www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

Riassunto Libro Sistemi Operativi

Informatica

Università degli Studi di Milano

77 pag.

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

SISTEMI OPERATIVI

COMUNICAZIONE DIGITALE

2017/2018

PROF. DARIO MAGGIORINI

RIASSUNTO LIBRO E LEZIONI

DI

MARCO DE CARLO

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

INTRODUZIONE 6

STORIA 6
HARDWARE 7
PROCESSORI 7
MEMORIA 7
DISPOSITIVI DI I/O 8
BUS 8
SPAZIO DI INDIRIZZAMENTO 8
SYSTEM CALL 8
SISTEMI MONOLITICI 9
SISTEMA MICROKERNEL 9
SISTEMI DISTRIBUITI (CLIENT-SERVER) 9

PROCESSI E THREAD 9

CREAZIONE DEI PROCESSI 9
TERMINAZIONE DI UN PROCESSO 10
CICLO DI VITA DEI PROCESSI 10
IMPLEMENTAZIONE DEI PROCESSI 11
MULTIPROGRAMMAZIONE 12
IMPLEMENTAZIONE DELLA MULTIPROGRAMAZZIONE 12
THREAD 13
THREAD IN JAVA 13
Creazione *13*
Terminazione *13*
Ereditarieta' *14*
COME SI USA RUNNABLE 14
IMPLEMENTAZIONE CLASSICA DEI THREAD 14
REGIONE CRITICA 15
MUTUA ESCLUSIONE IN JAVA 15
MUTUA ESCLUSIONE CON BUSY WAITING 15
ALGORITMO DI PATERSON 16
SLEEP AND WAIT 17
PRODUTTORE E CONSUMATORE 17
Strutture dati evolute *18*
Semaforo *18*
Semaforo binario (o mutex): *18*
MONITOR 19
SEMAFORI E MONITOR 19
Barriere *19*
IL PROBLEMA DEI 5 FILOSOFI 20
SCHEDULING 20
CATEGORIE DEI SISTEMI DI SCHEDULING 20
SCHEDULAZIONE NEI SISTEMI BATCH 21
Firts-Come First-Served (FCFS) *21*
Shortest Job First (SJF) *22*
Shortest Job First con prelazione (SRTF) *22*
SCHEDULING NEI SISTEMI INTERATTIVI 22
Scheduling Round Robin *22*
Scheduling con priorità *23*
Guaranteed Scheduling *23*
Lorrery *23*
Fair sharing *23*
Real-Time *23*
THREAD SCHEDULING 24

GESTIONE DELLA MEMORIA 25

MEMORIA SENZA ASTRAZIONE 25
REGISTRO BASE E REGISTRO LIMITE 25
ALLOCAZIONE DELLO SPAZIO 26
TRACCIAMENTO DELLA MEMORIA DISPONIBILE 26
FIRST FIT(il primo dove ci sta) *26*
NEXT FIT(il prossimo dove ci sta) *26*

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

BEST FIT(la taglia migliore) *27*
WORST FIT(la taglia peggiore) *27*
QUICK FIT *27*
MEMORIA VIRTUALE 27
ELEMENTI DELLA TABELLA DELLE PAGINE 28
TABELLA DELLE PAGINE SU PIÙ LIVELLI 29
TABELLA DELLE PAGINE INVERTITE 29
PAGINAZIONE SU RICHIESTA (DEMAND PAGING) 30
ALGORITMO SUL RIMPIAZZAMENTO DELLE PAGINE 30
ALGORITMO OTTIMO 30
FIRST IN FIRST OUT (FIFO) 30
ANOMALIA DI BELADY *30*
PROPRIETÀ DI INCLUSIONE *30*
SECONDA CHANCE 30
OROLOGIO 31
LRU 31
NOT RECENTLY USED (APPROSSIMAZIONE DI LRU) 31
NOT FREQUENTLY USED (NON È LRU) 31
WORKING SET 32
WSCLOCK 32
ALLOCAZIONE LOCALE E GLOBALE 32
PAGING DAEMON 33
DIMENSIONE OTTIMALE DELLE PAGINE 33
ISTRUZIONI SEPARATE E SPAZI DEI DATI 34
PAGINE CONDIVISE 34
LIBRERIE CONDIVISE 34
FILE MAPPATI *(IL FILE DI TESTO DIVENTA UN PEZZO DELLA MEMORIA)* 34
GESTIONE DEI PAGE FAULT 34
MEMORIA SECONDARIA 35
SEPARAZIONE TRA POLITICA E MECCANISMO 35
MEMORIA SEGMENTATA 36
CONFRONTO TRA POLITICHE 36

FILE SYSTEM 37

LIVELLO SOFTWARE DEL FILE SYSTEM 37
Nomi dei file *37*
Struttura interna *37*
Tipi di file *37*
Accesso ai file *38*
Attributi dei file *38*
Operazioni sui file *38*
LE DIRECTORY 38
Sistemi di directory a livello singolo *38*
Sistemi di directory gerarchici *38*
Path name *38*
Operazioni possibili *39*
LAYOUT DEL FILE SYSTEM 39
LOGICAL VOLUME MANAGER (LVM) 40
LVM E RAID 40
IMPLEMENTAZIONE DEI FILE 41
FILE CON ALLOCAZIONE CONTINUA 41
FILE CON ALLOCAZIONE A LISTE COLLEGATE 41
I-NODE 42
UNIX FILE SYSTEM 42
IMPLEMENTAZIONE DI DIRECTORY 43
FILE CONDIVISI TRA DIRECTORY 44
VARIANTI POSSIBILI DEL FILE SYSTEM 44
File System basati su log strutturati *44*
Journaled File System *44*
Virtual File System *45*
DIMENSIONE DEI BLOCCHI DI DATI 46
GESTIONE DEI BLOCCHI LIBERI 46

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

ANALISI DELLA CONSISTENZA DEL FILE SYSTEM 47
USO DELLA CACHE (DISK CACHING) 48

SISTEMA DI INPUT/OUTPUT 49

DISPOSITIVI I/O 49
Dispositivi a blocchi 49
Dispositivi a caratteri 49
Altri dispositivi 49
I/O MAPPATO IN MEMORIA 49
DIRECT MEMORY ACCESS (DMA) 50
INTERRUPT RIVISITATO 51
PRINCIPI DEL SOFTWARE PER I/O 52
I/O PROGRAMMATO 52
I/O GESTITO DA INTERRUPT 53
STRATIFICAZIONE DELL'I/O 53
GESTIONE DEGLI INTERRUPT 53
DEVICE DRIVER 53
SOFTWARE PER I/O INDIPENDENTE DAL DISPOSITIVO 54
Buffering 54
Error reporting 54
Allocazione e rilascio dei dispositivi dedicati 54
Dimensione dei blocchi indipendente dal dispositivo 54

DEADLOCK 55

I 5 FILOSOFI 55
UN MODELLO PER DEADLOCK 57
ALGORITMO PER IDENTIFICAZIONE DEI CICLI 57
COME TROVARE UN DEADLOCK CON RISORSE MULTIPLE 57
RISOLUZIONE DI UN DEADLOCK 58
EVITARE UN DEADLOCK 59
ALGORITMO DEL BANCHIERE (SINGOLA RISORSA) 59
ALGORITMO DEL BANCHIERE (RISORSE MULTIPLE) 59
PREVENIRE UN DEADLOCK 59
ALTRE QUESTIONI 63

CLOUD COMPUTING 64

CARATTERISTICHE DELLA VIRTUALIZZAZIONE 64
SOLUZIONI 64
LIVELLI DEI CLOUD (AS A SERVICE) 64
SaaS 64
PaaS 64
IaaS 65
UTILITY COMPUTING 65
CARATTERISTICHE DEL CLOUD 65
TRE TIPI DI CLOUD 65
IL LATO OSCURO DEL CLOUD 65

SISTEMI MULTIMEDIALI 66

IL DATO MULTIMEDIALE 66
CARATTERISTICHE DELLE INFORMAZIONI MULTIMEDIALI 66
(ALL'INTENRO DI UN SISTEMA OPERATIVO) 66
CODIFICA DI UN DATO MULTIMEDIALE 66
DUMB CODING 66
OTTIMIZZAZIONE DELLO SPAZIO 66
RIDONDANZA 67
IL MEZZO DI FRUIZIONE 67
PARAMETRI FISIOLOGICI 67
PARAMETRI PSICOLOGICI 67
AUDIO E VIDEO 67
MPEG 67
RIDONDANZA SPAZIALE 68
RIDONDANZA TEMPORALE 68

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

GOP – GROUP OF PICTURES 68
DEFINIZIONI 69
STREAMING 69
BUFFERING 69
Buffer underrun *69*
Buffer overrun *69*
E SE PERDIAMO DEI DATI? 69
FUNZIONALITA' VCR 69
PROCESSI MULTIMEDIALI 70
RMS (RATE MONOTONIC SCHEDULING) 70
EDS (EARLIEST DEADLINE FIRST SCHEDULING) 70
FILE SYSTEM CON SUPPORTO MULTIMEDIALE 71
ASSISTENTE 72
PRODUTTORE E CONSUMATORE 72
Utilizzando i monitor *72*
Utilizzando i semafori e i mutex *74*

I paragrafi in corsivo sono alcune possibili domande che vengono fatte all'esame orale.

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

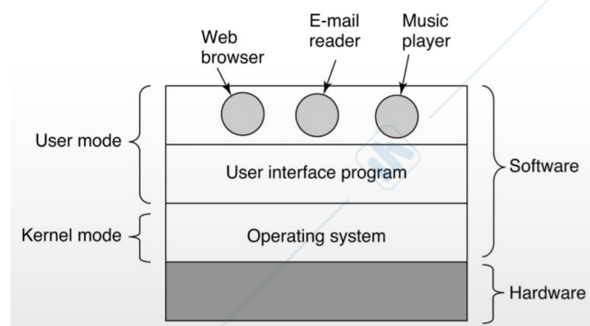
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

INTRODUZIONE

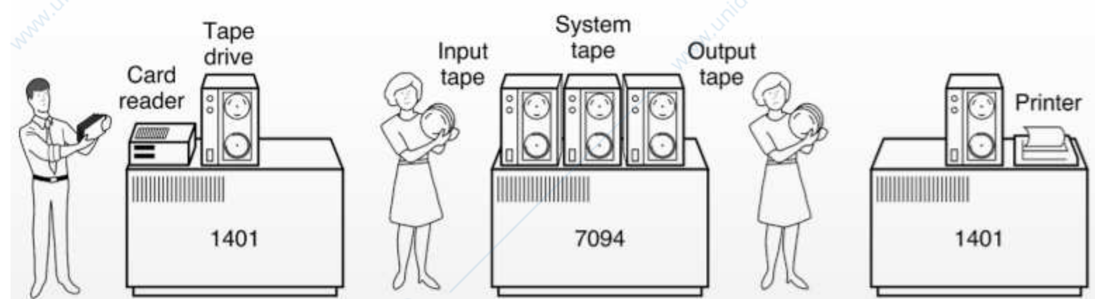
Il sistema operativo (SO) ha il compito di gestire tutte le risorse presenti all'interno dell'elaboratore. Sono possibili due modalità operative: modalità Kernel e modalità Utente. Il sistema operativo viene eseguito in modalità Kernel, in questo modo ha accesso a tutto l'hardware. Il resto del software gira in modalità utente, in cui è disponibile solo un sottoinsieme di istruzioni macchina.



In particolare ai programmi in modalità utente sono vietate quelle istruzioni che pregiudicano il controllo della macchina o gestiscono I/O. Il sistema operativo ha anche il compito di gestire le risorse. Si hanno un insieme di algoritmi che decidono a chi allocare le risorse e come queste devono essere utilizzate.

Storia

Il primo sistema di calcolo è stato il **mainframe**, esso era un sistema di calcolo utilizzato soprattutto da grandi aziende. Non esistevano monitor, quindi i sistemi erano sistemi batch. Essi servivano principalmente per fare calcoli, il sistema era basato su; un linguaggio di programmazione chiamato Fortran e l'input e output utilizzavano le schede perforate. All'interno del sistema di calcolo venivano caricate le schede perforate e queste erano lette in sequenze per esser tradotte su un nastro magnetico. Il nastro veniva caricato sulla CPU dove avveniva l'esecuzione.



L'elaboratore rappresentato nell'immagine è un unico "computer" suddiviso in tre parti, partendo da sinistra sono:

1401: è il pezzo di mainframe che traduce le schede perforate in segnali magnetici
7094: è la CPU, che elaborava e scriveva sul nastro il risultato. Sono presenti più alloggi per i nastri, quindi la possibilità di inserire più programmi. La memoria era preziosissima e c'era un software chiamato Resident monitor che si occupava di acquisire il contenuto del nastro e inserirlo in memoria. L'ultima istruzione utile era un JUMP alla locazione 0 dove il resident monitor sganciava il nastro, caricava il prossimo e riniziava.
1401: leggeva il nastro e stampava l'output.

Con l'aumentare della memoria e delle periferiche si è pensato di progettare **sistemi batch multi programmati**. Dei tre alloggi dei nastri presenti nella CPU, è possibile che la CPU esegue il primo nastro e mentre lo esegue legge il secondo nastro, ecc.. così quando termina l'esecuzione del primo job inizia subito il successivo, senza perder tempo. Un sistema multi programmato significa che all'interno della memoria sono presenti più job contemporaneamente, come questi si gestiscono la CPU è un altro problema.

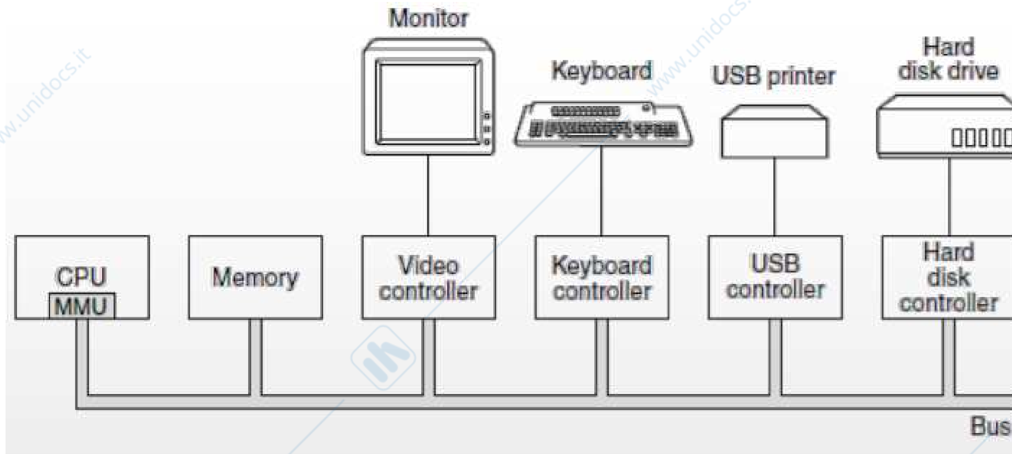
Ad un certo punto è arrivata la necessità di proteggere i dati, senza che l'esecuzione di un job andasse ad interferire con un altro. In più è arrivata l'esigenza di riuscire a passare da un processo all'altro, progettando così i **sistemi time-sharing**. Il sistema time-sharing è un sistema multi-programmato in cui il sistema di allocazione della CPU cambia da un job all'altro senza che nessuno di questi sia effettivamente terminato.

Oggi si è arrivati a costruire sistemi in grado di essere acquistati da famiglie.

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

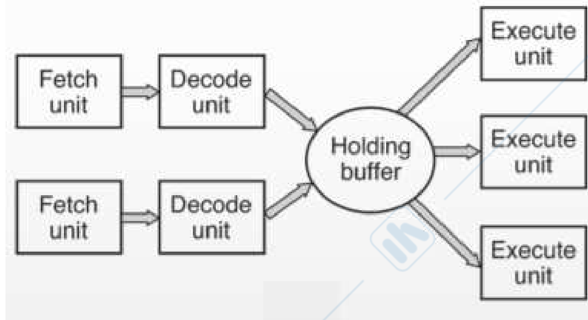
Hardware



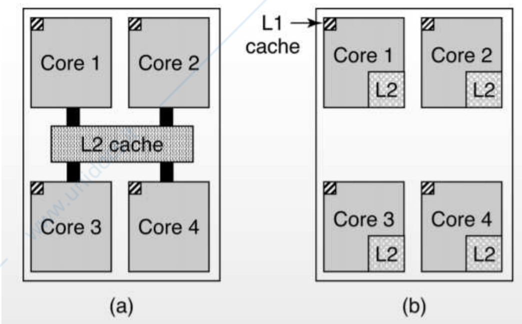
Un personal computer è generalmente costituito da dalle risorse fondamentali che nonostante l'evoluzione della tecnologia non sono cambiate, queste sono la CPU, il Bus che trasporta le informazioni (si suddivide in bus dati e bus indirizzi) e collega gli elementi all'interno del calcolatore, la memoria, e le varie periferiche che si collegano attraverso un controller che si occupa di raccordare il bus con il sistema di comunicazione con la periferica.

Processori

La CPU lavora principalmente in tre fasi, preleva, decodifica, esecuzione. L'organizzazione di queste fasi è chiamata **pipeline**, cioè la CPU ha a disposizione più dispositivi separati per prelevare, decodificare ed eseguire le istruzioni.



Le CPU contengono all'interno alcuni registri per memorizzare variabili importanti o risultati temporanei. Alcuni dei registri più importanti sono il **program counter** che contiene l'indirizzo di memoria in cui si trova la successiva istruzione da eseguire. Lo **stack pointer** che punta alla cima dello stack attuale in memoria, permette le operazioni di pop e push. Il **PSW**, questo registro contiene bit di controllo, come un bit che indica quale modalità è attiva se utente o del kernel.



Il Core contiene i transistor che determinano la capacità di elaborazione. Possiamo avere due varianti

a) Un chip a quattro core con memoria cache di tipo L2 condivisa
b) Un chip a quattro core con memorie cache di tipo L2 separate

La cache L1 è sempre all'interno della CPU e fornisce istruzioni codificate al motore di esecuzione della CPU. Una seconda cache chiamata cache L2, contiene istruzioni usate recentemente. La differenza tra L1 e L2 sta nelle tempistiche.

Memoria

Possiamo descrivere le memorie in una gerarchia piramidale, dove troviamo in cima la memoria che ha un costo ed una velocità di trasferimento dei dati elevata, mentre alla base della piramide troviamo quella memoria che ha un costo e una velocità bassa.

Typical access time		Typical capacity
1 nsec	Registers	<1 KB
2 nsec	Cache	4 MB
10 nsec	Main memory	1-8 GB
10 msec	Magnetic disk	1-4 TB

- Iniziando dalla cima della piramide troviamo i **registri** interni alla CPU, si accede in frazioni di nano secondi.
- Sempre all'interno della CPU è presente la **cache associativa**, essa è una zona di memoria all'interno della CPU e viene adoperata

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

per memorizzare gli ultimi dati che sono stati utilizzati. Se vale il principio di località allora è molto vantaggiosa.

- Successivamente troviamo la **memoria centrale** (RAM) ad essa si accede attraverso il bus dati e ha un tempo di acceso meno di 10 nanosecondi. Tutte le richieste della CPU che non possono essere soddisfatte dalla cache vanno alla memoria principale. Oltre alla memoria principale molti computer hanno una piccola quantità di memoria ad accesso casuale non volatile chiamata ROM. Essa è programmata dal costruttore e non può essere cambiata e solitamente in essa risiede il programma di caricamento usato all'avvio del computer.
- Infine troviamo come memorie il **disco** e le **memorie flash** che non sono memorie volatili, ma possono essere cancellate e riscritte.

Al prof non piace questa classificazione in quanto la cache associativa è una cache della memoria centrale, e a sua volta la memoria centrale è una cache del disco e a sua volta il disco è una cache della rete. (contento lui, contenti tutti!)

Dispositivi di I/O

I dispositivi di I/O sono costituiti da due parti: un controller e il dispositivo stesso. Il **controller** è un chip o un insieme di chip che fisicamente controllano il dispositivo. Accetta i comandi dal sistema operativo ed ha come compito di fornire un'interfaccia più semplice possibile al sistema operativo. Dato che ogni controller è diverso è necessario un software diverso per controllarne ciascuno. Il software che comunica con un controller, impartendogli i comandi e accettando le risposte è chiamato **driver del dispositivo**. Ogni produttore di controller deve fornire un driver per ogni sistema operativo che supporta. Il driver viene inserito all'interno del sistema operativo, in questo modo può girare in modalità kernel. Ci sono tre metodi per inserire un driver nel kernel e sono:

1) Collegare il kernel con il nuovo driver e riavviare il sistema.
2) Creare una voce in un file del sistema operativo indicando che necessita del driver e poi riavviare il sistema.
3) Il sistema operativo accetta i nuovi driver mentre è in esecuzione e li installa in corsa senza necessità di riavvio.

Bus

Il bus è un insieme di piste che porta i bit avanti e indietro. È condiviso da tutte le periferiche all'interno del calcolatore e ha bisogno di un sistema di arbitraggio. Esistono all'interno del calcolatore tanti tipi di bus con dell'elettronica (dei chip) che servono per convertire i segnali di un bus su un altro tipo di bus.

Spazio di indirizzamento

Ogni processo ha un suo spazio di indirizzamento, è il quantitativo totale di byte che il programma può utilizzare. Oggi gli indirizzi possono variare tra 32 o 64 bit, determinando uno spazio degli indirizzi rispettivamente di 2^{32} o 2^{64}byte. Nel caso in cui un processo richiede più spazio rispetto alla memoria principale viene utilizzata una tecnica chiamata memoria virtuale.

System call

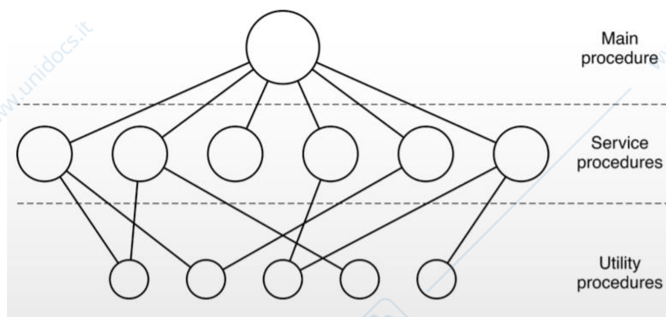
Una system call è una funzionalità che il sistema operativo mette a disposizione per poter accedere a dei servizi che coinvolgono le sue strutture dati interne. La system call è diversa dalle altre chiamate perché porta il processore in uno stato differente. Si hanno delle system call per ogni funzionalità, vengono raggruppate in famiglie. System call per la:

- Gestione processi; forck() nuovo processo, exit() terminare un processo,
- Gestione memoria;
- Gestione file system; aprire il file, chiudere il file, scrivere, creare nuove directory

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

Sistemi monolitici



Nei sistemi monolitici non era presente la distinzione tra modalità utente e kernel. Tutte le funzionalità erano tutte sullo stesso livello. Dato che questa funzione non andava bene si è optato nel raggruppare tutte le funzionalità simili su diversi livelli, dando vita ai SISTEMI A LIVELLI (stratificato). Successivamente si è pensato di assegnare delle funzionalità speciali solamente al kernel. Si andò a sviluppare un kernel molto piccolo con il compito di poter svolgere alcune funzionalità, mentre le altre nella modalità utente.

Sistema microkernel

È presente un piccolo kernel che mette a disposizione due funzionalità, un metronomo che scandisce il tempo e un sistema di comunicazione tra i processi. Tutto il resto sono processi, se ad esempio non c'è la stampante non esisterà il processo relativo.

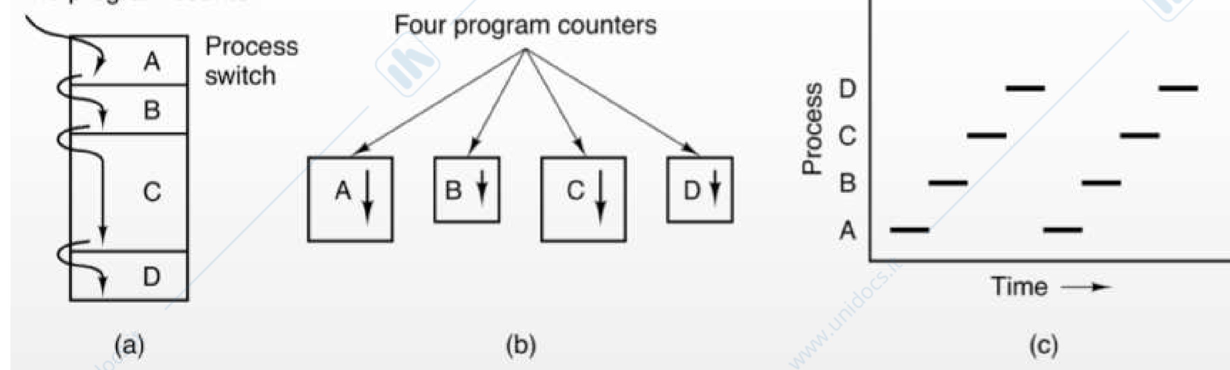
Sistemi distribuiti (client-server)

È possibile separare il kernel e farli parlare attraverso la rete.

Processi e Thread

Un processo è un programma abbinato ad uno stato esecutivo. Un processo è un'entità che svolge del lavoro all'interno del calcolatore. Un processo nello svolgere il suo lavoro, può fare uso di uno o più thread al suo interno a cui demandare delle esecuzioni parallele.

Tre modi per vedere concettualmente i processi:



a) Dal punto di vista della <u>memoria</u>. **Sheduling**. Esistono tanti processi in memoria, e ho un solo program counter. È presente un solo processore e viene eseguito una sola istruzione per volta.
b) Dal punto di vista del <u>programmatore</u>. **Programmazione concorrente**. Abbiamo una sola memoria, ma ogni processo ha il suo spazio di indirizzamento. È presente un solo program counter fisico, così quando ogni processo è in esecuzione, il suo contatore program counter logico è caricato nel program counter fisico. Al termine il program counter fisico viene salvato in memoria nel program counter logico archiviato del processo. Quindi ogni processo ha un proprio program counter.
c) Dal punto di vista del <u>sistema operativo</u>. **Time-sharing**. In ogni istante di tempo viene eseguito solo un processo.

Creazione dei processi

Gli eventi che possono causare la creazione di un processo sono (importante i primi due punti):

1. All'inizializzazione del sistema
2. All'esecuzione di una system call (FORK) da parte di un altro processo
3. Un utente richiede di creare un nuovo processo (lo chiede ad un processo già esistente…posso riportarlo al punto 2)

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

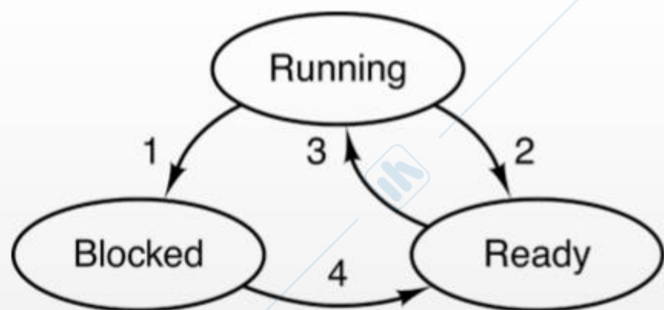
4. Inizializzazione di un job batch (processo programmato, come backup alle 3 di notte. Cmq è presente un processo che crea un processo di backup…per questo posso riportarlo al secondo punto.)

Terminazione di un processo

Un processo potrebbe terminare a causa del verificarsi di una delle seguenti condizioni.

1. Il processo termina in modo volontaria utilizzando una system call chiamata “exit” e questa permette al processo di richiedere al sistema operativo di eliminare tutte le sue strutture dati.
   La system call exit può essere:
   a) Esplicita: chiamando exit all’interno del programma.
   b) Implicita: quando l’esecuzione del main termina.
2. Il processo termina in modo volontariamente a fronte di un errore. (Error exit)
   a. Dal punto di vista pratico, viene effettuato una system call exit dato un parametro numerico, tipicamente un valore negativo.
   b. Dal punto di vista del sistema operativo, quando termino un processo segnalando un errore il sistema operativo deve notificare il processo padre che ci è stato un errore per cui il processo rimane congelato (processo zombie) fino a quando il processo padre vede l’errore e prende una decisione (la decisione può essere o non me ne frega niente oppure ho visto l’errore ora puoi ripulire le strutture dati)
3. Fatal error, il processo termina involontariamente senza che lui l’abbia richiesto. Il processo ha fatto un errore no al livello di codice, ma al livello di sistema operativo. (esempio divisione per zero)
4. È possibile uccidere un processo attraverso un altro processo in modo involontario (esempio quando usiamo il Task Maneger). Con UNIX esiste una system call che prende il nome di KILL (*non serve ad uccidere i processi, ma serve per mandare dei messaggi/segnali di vario tipo*) che viene usato dal sistema operativo come segnale per eliminare il processo, in questo caso.

Ciclo di vita dei processi



1. Processo bloccato a causa di una system call di I/O
2. Torno allo stato di pronto perché il quanto di tempo del processo in esecuzione è terminato. Il SO interviene utilizzando lo Scheduler per scegliere un altro processo.
3. Scheduler (schedulazione): il sistema operativo decide quale processo deve utilizzare la CPU. Il SO fa partire un pezzo di software al suo interno che si chiama DESPACHER(o Scheduler) e questo alloca la struttura dati del processo.
4. Questa traslazione dallo stato di blocco a quello di pronto, avviene principalmente a fronte di un Interrupt. La CPU riceve un interrupt dalla periferica, l’interrupt prevede lo spostamento del processo collegato a quell’evento di I/O dallo stato di bloccato allo stato di ready

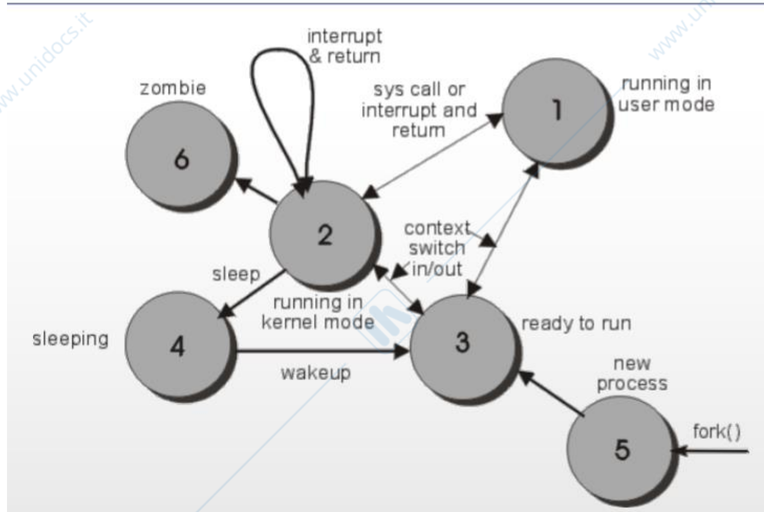
Un processo vive principalmente in tre macro strati:

Ready (pronto): il processo esiste, è all’interno della memoria. Vuole utilizzare la CPU, ma è in attesa che arrivi il suo turno.
Running (esecuzione): quando il processo possiede la CPU. È possibile che venga effettuata una system call di I/O portando il processo in uno stato di blocco.
Blocked(blocco): il processo chiede dati di I/O, al termine si rimette in coda.

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

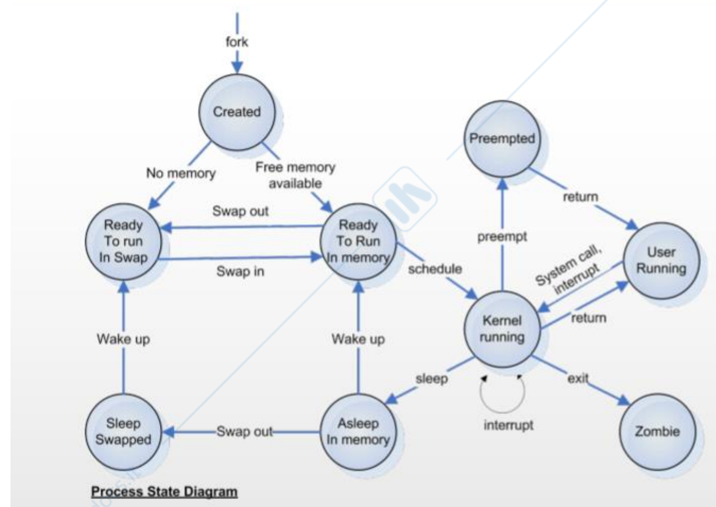
Ciclo di vita (già un po' meglio)



Ho due modalità di esecuzione:
1. Esecuzione del processo in modalità utente, quando il processo sta eseguendo il mio codice. (**User mode**)
2. Esecuzione del processo, mentre sto eseguendo il codice della system call. (**Kernel mode**)

New process: Prima di andare in READY ho una fase per allocare le risorse (riservare la memoria, verificare che non ci siano errori, ecc..) viene utilizzato questo stato per la creazione del processo e per verificare che tutte le operazioni vadano a buon fine. Solo allora viene inserito in READY il processo, e quindi tutte le strutture dati del processo saranno corrette.
Zombie: ci arrivo dallo stato del kernel mode a causa di una system call e aspetto che il SO mi spazza via. Il processo in questo stato occupa ancora la memoria.
Context switch I/O: se il quanto di tempo scade nel momento in cui sono in kernel mode, avviene un cambio di contesto in ready.

Ciclo di vita (quello quasi vero)



La parte destra è quella descritta in precedenza.
La parte sinistra tiene conto di quei processi che non fanno niente per molto tempo. Il SO utilizza delle politiche di SWAPPING, cioè quando il processo non fa nulla per lungo tempo allora il processo viene preso e messo sul disco (il processo viene SWOPPATO). Quando il processo viene richiamato, viene riportato il processo in memoria. Questo nel ciclo di vita impone di individuare due modalità che sono

a) Processi pronti in memoria
b) Processi pronti sul disco

Oppure

a) Processi che sono in attesa di I/O in memoria
b) Processi che sono in attesa di I/O sul disco.

Implementazione dei processi

I dati relativi di un processo vengono memorizzati nel **Process control block** (**PCB**). Per ogni processo viene inserita una entry, contente il PCB, all'interno di una tabella dei processi chiamata **process table** che viene memorizzata all'interno del kernel. Il PCB di un processo, è la struttura dati di un processo che contiene le informazioni essenziali per la gestione del processo stesso. Avremo un PCB per ogni processo allocato.
Esistono tre categorie principali all'interno di un PCB e queste sono:

1. **Process management**. Tutte quelle operazioni che sono funzionali per allocare la cpu. Metterò una fotografia dello stato della CPU dell'ultima volta che l'ho utilizzata. Ad esempio: che registri c'erano, dove si trovava il program counter, qual era la configurazione dello stato interno, il valore dello stack pointer, ecc… sono tutte informazioni che quando il processo potrà riutilizzare la CPU verranno copiate ed incollate nei registri appositi e da lì il program counter tornerà ad eseguire quel processo.
2. **Memory management**. Una famiglia di parametri che servono per capire dove sono i dati del processo. Sono presenti tre segmenti importanti
   a) Text Segment: dove si trova il binario da eseguire.
   b) Data Segment: dove si trovano i dati allocati dinamicamente

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

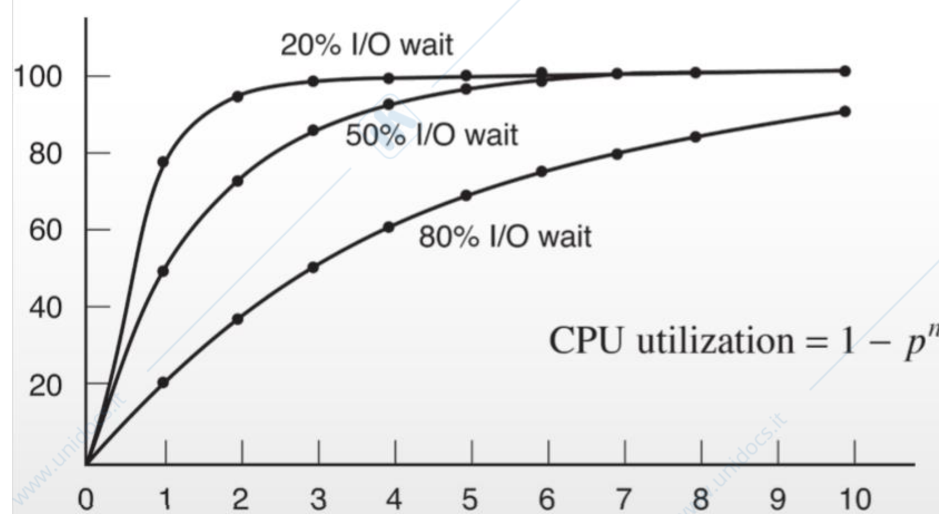
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

    c) Steck segment a cui punta lo stack pointer.
3. **File management**. L'ultima categoria riguarda il rapporto che è presente tra il processo e il file system. È necessario identificare per ogni processo qual è la directory a cui si fa riferimento.

La CPU passa continuamente da un processo all'altro, questo rapido cambio di contesto è chiamato multiprogrammazione.

Multiprogrammazione

Il grado di multiprogrammazione indica quanto "affollato" può essere il SO dal punto di vista dei processi.



Sull'asse delle x vediamo il numero dei processi. Sull'asse delle y vediamo l'utilizzo della CPU.
Se il livello di I/O, quindi la probabilità di trovare un processo nello stato di bloccato è alto. Allora è possibile inserire molti processi nel sistema.
Con una probabilità bassa di trovare un processo nello stato di blocco, il sistema raggiunge la saturazione con pochi processi.
Mediamente l'uso della CPU è data da $\mathbf{1-p^n}$,con n=numero dei processi e p=è la probabilità di trovare ciascun processo nello stato di wait.

Questo grafico non tiene conto che è presente un altro elemento che utilizza la CPU, ovvero il kernel. Se io aumento il livello di multiprogrammazione, aumento anche in maniera drastica il quantitativo di CPU che è necessario al kernel per gestire tutti i processi. Nel tempo la curva ad un certo punto andrà a scendere perché il kernel avrà bisogno del tempo ad esempio per gestire il PCB. Quindi il livello di multiprogrammazione è possibile aumentarlo, ma non esiste una formula esatta.

Implementazione della multiprogramazzione

È necessario istruire la CPU per implementare la multiprogrammazione. Ho una serie di operazioni da fare, dal momento in cui devo passare da un processo all'altro. La CPU riceve un segnale asincrono, che fa partire lo Scheduler. Il quale blocca il processo ed esegue un programma che salva il processo nel PCB. Il programma ha bisogno della CPU e anche lo stato del processo è presente nella CPU, per questo non posso utilizzarla altrimenti andrei ad inquinare i dati che devo salvare. Per questo tutte le CPU disponibili per la multiprogrammazione supportano un Interrupts di clock, cioè un interrupt che viene generato automaticamente ogni tot. Quando arriva viene trattato da un altro meccanismo presente nella CPU che ha due copie dei registri, la copia che uso per eseguire e l'altra copia di backup. Quando arriva l'interrupt i registri della copia di lavoro finiscono nella copia di backup (interrupt prevede questo).

1. Si riceve un interrupt di clock e la CPU si blocca.
2. La CPU (senza la richiesta di nessuno) prende tutto e li mette nei registri di backup.
3. Viene caricato il nuovo contatore di programma dal vettore dell'interrupt
4. Viene impostato il nuovo stack
5. Viene eseguita la procedura di servizio dell'interrupt
6. Lo scheduler decide quale processo eseguire come successivo
7. Avvia il nuovo processo corrente

*Come faccio a sapere quale interrupt ho ricevuto? L'interrupt è un bit e il tipo è identificato da un numero e non posso rappresentare un numero con un bit. Sul BUS dati appare l'identificativo dell'interrupt. Viene usato questo numero come indice all'interno della tabella che punta a delle funzioni all'interno del kernel e faccio una JUMP per eseguire il codice per gestire quel specifico interrupt. La tabella è chiamata **Interrupt table** e il codice è un **interrupt service routine**.*

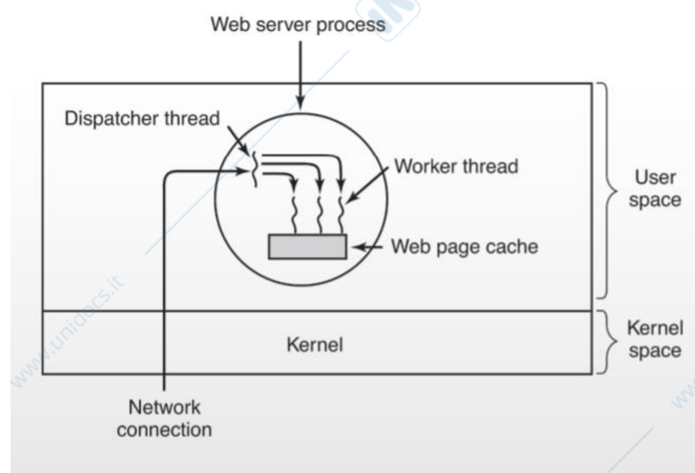
Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

1. Ricevo un interrupt
2. Salvo lo stato della CPU, significa mettere i registri nella copia di backup dei registri
3. Metto in sicurezza il processo, significa innescare una procedura in linguaggio macchina che prende i registri backup e li inserisce nel PCB.
4. Decido quale nuovo processo devo eseguire (SCHEDULER)
5. Recupero lo stato del nuovo processo
6. Ritorno a far esecuzioni di user mode

Tutto questo è il DISPATCHER

Thread



Un Thread è un flusso esecutivo all'interno di un processo

La *differenza tra un Thread e un processo* è che i processi sono *entità isolate*, mentre i thread *condividono lo spazio di indirizzamento* (quindi vedono le stesse variabili), ma ognuno di loro ha il suo flusso esecutivo.
In più i thread sono più leggeri dei processi e per questo sono più facili da creare e cancellare.
Dato un processo web server, che risiede su un calcolatore raggiungibile dalla rete. Esso aspetta un client che lo chiami e apra un canale di comunicazione. Il server riceve delle richieste sotto forma di stringa. Il web server restituisce quello che il client gli richiede. È possibile che ci siano molte richieste in poco tempo per questo è presente un thread che riceve le richieste, e le richieste ricevute vengono rimbalzate al primo thread libero che andrà ad interrogare le periferiche presenti per soddisfare la richiesta. Questo funziona solo se i thread possono condividere tra di loro le informazioni. Il vantaggio dei processi è la sicurezza.

Thread in Java

Per gestire i thread in java esiste una classe astratta Thread e i principali metodi sono:

- Metodo **run**()
  Sono le istruzioni che 'compongono' il thread va quindi ridefinito per specializzare il thread
- Metodo **start**()
  Non devo riscrivere questo metodo e vien utilizzato per avviare il thread
- Metodo **sleep**()
  È usato per sospendere per un determinato periodo di tempo l'esecuzione

CREAZIONE:

1. Definisco una nuova classe che estenda THREAD
   Class Cipolla extends Threads
2. Creo una istanza di questa nuova classse
   Cipolla t=new Cipolla()
3. "avvio" con il metodo start()
   t.start()

TERMINAZIONE:

Il thread termina spontaneamente e viene deallocato dalla JVM nel momento in cui termina il metodo run()

Esistono alcuni metodi, come suspend(), resume(), stop(), che vengono evitati perché possono portare a stati inconsistenti.

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

EREDITARIETA':
L'ereditarietà multipla non è supportata (infatti non è possibile dichiarare due volte extends nella stessa classe)
Per risolvere questo problema è possibile implementare l'interfaccia **Runnable** e poi creare un oggetto Thread a partire da tale oggetto. Oppure posso usare una classe innestata.

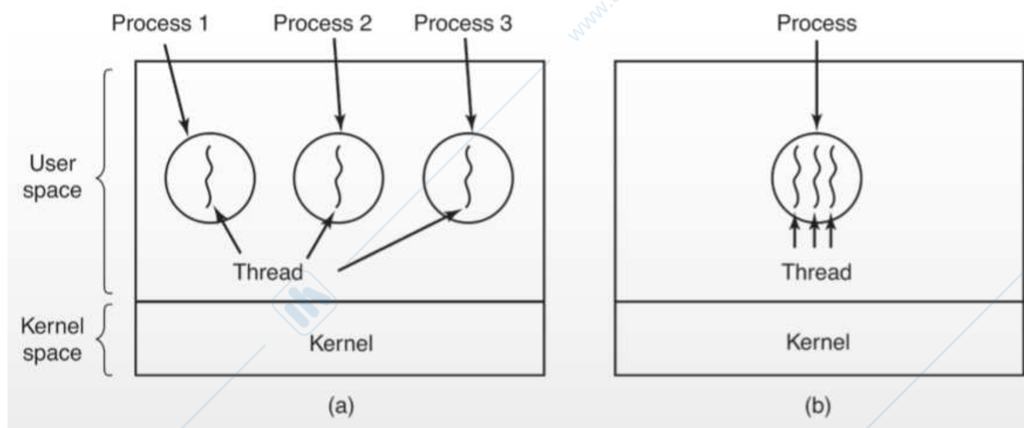
Come si usa Runnable

Un oggetto che estende Runnable non è un thread, ma descrive che cosa deve fare un Thread. Runnable si usa:

1. Faccio implementare Runnable a una classe esistente o una sua sottoclasse
   class MyRun implements Runnable
   oppure
   class MyRun extends Threads implements Runnable
2. Creo un oggetto Thread usando l'istanza come parametro.
   Thread t = new Thread(new MyRun());
3. "avvio" il thread con start()
   t.start();

Se più thread accedono alla stessa risorsa, sarà necessario gestirli in quanto la scelta di quale processo eseguire dipenda dalla schedulazione.

Implementazione Classica Dei Thread

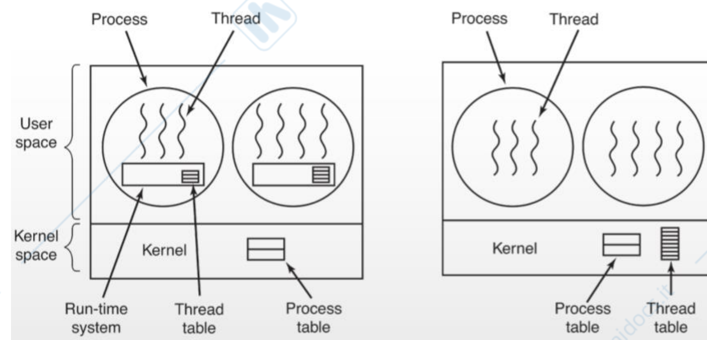


Possiamo rappresentare i thread nello spazio utente in due modalità:
a) Più processi classici, cioè con un thread singolo
b) Un processo con più thread

Le informazioni del processo vengono viste da tutti i thread che sono: spazio di indirizzamento, variabili globali, i file aperti, quali sono i processi figli, quanta CPU che sta usando.
Ci sono informazioni che ha il thread e il processo non li vede. Queste informazioni sono inerenti al flusso esecutivo del thread. Quando eseguo un thread esso avrà le sue informazioni locali che sono:

- il program counter
- i registri
- lo stato, cioè in quale stato il processo si trova nel suo ciclo di vita.

È possibile implementare i thread in diversi modi, cioè o nello spazio utente, o nel kernel oppure utilizzare una soluzione ibrida.



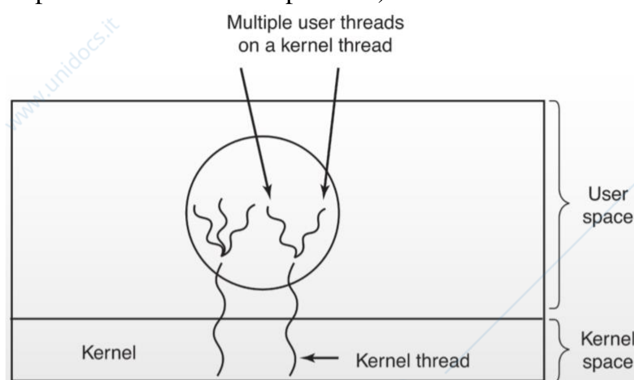
Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

Nello spazio utente: È compito del processo conservare le informazioni riguardanti i thread, è necessaria una Tabella dei Thread all'interno del processo presente nello spazio utente e questa tabella sarà nascosta al kernel, mentre nel Kernel sarà presente una tabella dei processi dove avremo le informazioni del PCB inerenti ai processi (o del thread principale che gestisce il processo).
È necessario implementare nel processo una libreria Run-Time System, che serve al processo per gestirsi in autonomia i thread (ed è quello che accade con la Java Virtual Machine, cioè la JVM conosce di avere dei thread al suo interno e decide quale eseguire)

Nello spazio kernel: Le informazioni riguardanti i thread sono presenti in una tabella dei thread all'interno dello spazio Kernel. Quando un thread si blocca, il kernel può eseguire o un altro thread dello stesso processo oppure un thread di un processo diverso. Dipende se il prossimo flusso esecutivo che userà la CPU sarà in base al processo o in base al thread.

Eseguire un cambio di contesto fra thread di livello utente richiede una manciata di istruzioni macchina, mentre con i thread di livello kernel richiede un completo cambio di contesto. D'altro canto, con i thread a livello kernel, avere un thread bloccato su un'operazione di ingresso/uscita non sospende l'intero processo come accade a livello utente. (in quanto all'interno del singolo processo non ci sono interrupt e pre questo è impossibile schedulare i processi)



È presente una soluzione ibrida. Possiamo dividere le operazioni presenti nel kernel in più flussi esecutivi paralleli. Un thread all'interno del kernel può occuparsi di coordinare uno o più thread a livello utente. Possiamo divere i thread, ad esempio, sulla base di quanta CPU utilizzano.
In questo modo è difficile individuare se l'eventuale bug è presente nel kernel o nello spazio utente.

Regione critica

Più processi possono condividere una parte di memoria comune che ciascuno può leggere e scrivere. Questa memoria condivisa può trovarsi nella memoria principale o può essere un file condiviso.
Situazioni dove due o più processi leggono o scrivendo i medesimi dati condivisi, il risultato finale dipende dall'ordine con cui vengono eseguiti i processi, sono chiamate condizioni di **corsa critica**.

La **regione critica** è quella porzione di codice del programma in cui i processi sono in competizione per accedere alla risorsa condivisa.
La **mutua esclusione** viene utilizzata per risolvere il problema della corsa critica, ovvero per proibire a più di un processo di leggere e scrivere dati condivisi contemporaneamente. Quindi in questo modo tutti gli accessi alla regione critica diventano transazionali. Servono quattro condizioni, che sono:

1. Due processi o più non devono mai trovarsi contemporaneamente nelle loro regioni critiche.
2. Non è possibile fare assolutamente nessuna assunzione su il numero di CPU e sulla loro velocità (non è possibile far affidamento sulla velocità della CPU)
3. Nessun processo al di fuori della sua regione critica deve esser in grado di bloccare gli altri processi che vogliono entrare nella regione critica
4. Nessun processo dovrebbe attendere all'infinito per entrare nella sua regione critica

Mutua esclusione in Java

In java la mutua esclusione avviene decorando un metodo con *sychronized.* Se un thread si trova all'interno di una zona di codice delimitata con *sychronized* di un certo oggetto nessun altro thread può accedere ad una zona delimitata dello stesso oggetto.
La JVM, che ha al suo interno la gestione dei thread, ha il compito di capire quando ogni thread si trova nella sua regione critica e di conseguenza blocca gli altri thread.

Mutua esclusione con Busy Waiting

Come viene tradotta la mutua esclusione in istruzioni adatte al kernel e per la CPU.
Disabilitare gli interrupt: la CPU ha all'interno della sua configurazione il **Program Status Word** (o **PSW**), cioè un registro di stato che contiene informazioni sullo stato dei programmi in

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

esecuzione. Essa detiene anche l'informazione su quali interrupt sono stati accettati e quali sono stati ignorati. È possibile disattivare gli interrupt quando il processo entra nella sua regione critica e li riabilito quando il processo esce dalla regione. Questo perché la CPU passa da un processo all'altro solo sulla base di interrupt del clock o altri interrupt Questo comporta la disattivazione dello scheduler e di conseguenza il processo potrebbe impiegare molto tempo e potrebbe fare quello che gli pare.
Se invece il sistema è multiprocessore, disabilitare gli interrupt interessa solo la CPU (il core) che ha eseguito la interruzione degli interrupt, ma le altre (core)CPU continuerebbero l'esecuzione e potrebbero accedere alla regione critica.

Variabili di lock: ogni regione critica ha una sua variabile booleana che segna se la regione critica è occupata o meno. Prima di entrare nella regione critica, è necessario controllare se la variabile è nello stato false. Il problema che potrebbe sorgere è che un processo riesca ad entrare nella regione critica senza far in tempo a modificare il lock, così da permettere l'ingresso ad un altro processo nella regione.

Test and Set Lock (TSL): L'istruzione TSL è un'istruzione a livello assembly e ha il compito di indicare al processo se in una certa locazione di memoria è presente il valore 1, in tal caso lo imposta a 0 e va avanti (lo fa in un colpo solo a differenza del metodo precedente). Se la locazione di memoria che uso come parametro della TSL è 0 allora il processo prova ad acquisire la CPU in un secondo momento. Non è una situazione ideale perché tutte le volte che il processo può avere la CPU deve eseguire la TSL e quindi è necessario chiedere allo scheduler di ricaricare le strutture dati per eseguire la TSL e questo comporta una perdita di tempo.

Alternanza stretta: tramite un busy waiting, mi garantisce un'alternanza stretta.

```
1. WHILE(TRUE){
2.  WHILE(TURN !=0) /*FIN TANTO NON È IL MIO TURNO ASPETTO NEL WHILE*/ ;
3.   CRITICAL_REGION();  // TURN È 0
4.   TURN=1;
5.   NONCRITICAL_REGION();
6. }
```

```
1. WHILE(TRUE){
2.  WHILE(TURN !=1) /*LOOP*/ ;
3.   CRITICAL_REGION();
4.   TURN=0;
5.   NONCRITICAL_REGION();
6. }
```

Questo fa uso di busy waiting e si verifica nel while(turn). È possibile complicare il codice per far gestire più processi e prevede che tocchi ai processi in modo alternato e questo non è sempre la situazione ottimale.

Algoritmo di Paterson

Soluzione con N processi, mentre quella precedente era solo con due processi

```
1. #DEFINE FALSE 0
2. #DEFINE TRUE  1
3. #DEFINE N     2                                   /* NUMERI DI PROCESSI*/
4.
5. INT TURN;                                          /* A CHI TOCCA
6. INT INTERESTED[N];                      /* TUTTI I VALORI INIZIALIZZATI A 0 (FALSE)
7.
8. VOID ENTER_REGION(INT PROCESS);                 /* IL PROCESSO È O 0 OPPURE 1
9. {
10.  INT OTHER;                                       /* NUMERO DELL'ALTRO PROCESSO
11.
12.  OTHER=1-PROCESS;                               /* IL NUMERO DEL PROCESSO OPPOSTO
13.  INTERESTED[PROCESS]=TRUE;                        /* MOSTRA CHI È INTERESSATO
14.  TURN=PROCESS;                                      /* IMPOSTA IL FLAG
15.  WHILE(TURN == PROCESS && INTERESTED[OTHER]==TRUE)    /* ISTRUZIONE VUOTA, SE ENTRAMBE VERE ALLORA ENTRA
16. }
17.
18. VOID LEAVE_REGION(INT PROCESS)                   /* IL PROCESSO CHE È PARTITO
19. {
20. INTERESTED[PROCESS]=FALSE;                  /* INDICA L'USCITA DALLA REGIONE CRITICA
21. }
```


Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

Sleep and Wait

Per evitare il busy waiting, in quanto esso comporta diversi problemi come
1. Uso eccessivo e inutile della CPU
2. **Problema dell'inversione della priorità**, cioè quando è presente un processo con priorità molto alta che viene condizionato da un processo con priorità bassa.

La soluzione è quella di creare delle system call (**sleep** e **wait**) che ci permettono di sospendere il processo senza occupare la CPU.
In Java abbiamo a disposizione
- **Wait()**: dall'interno di una regione critica posso chiedere di sospendere il processo fino a che un altro processo non eseguirà una notify (riabilitando lo scheduling) dalla stessa regione critica. Il thread che fa uso di wait() rinuncia al lock (in questo modo altri thread possono accedere alla regione critica) e nel caso venga risvegliato deve acquisirlo di nuovo. Nel caso ci siano più processi in wait non possiamo prevedere quale sarà quello che verrà risvegliato e che acquisterà il lock.
- **Notify()**: permette di risvegliare un thread (non è possibile specificare quale) in stato di waiting, lo scheduler ne sceglie uno autonomamente. Il therad che esce dallo stato di waiting ha la possibilità di riacquisire il lock.

Sia wait che notify non serve a garantirmi la mutua esclusione senza busy waiting, in quanto questo lo faccio con il syncronized. Essi mi aiutano a garantire il punto 4, cioè a garantire che nessun processo aspetterà in eterno per poter accedere alla risorsa condivisa.

Produttore e Consumatore

Consiste in due processi che condividono lo stesso buffer, di dimensione fisse. Il produttore ha il compito di inserire le informazioni nel buffer e il consumatore di prelevare le informazioni.

La situazione si complica quando il produttore vuole inserire un nuovo elemento nel buffer quando questo è già pieno. La soluzione del produttore è quella di "andare a dormire" ed essere risvegliato quando il consumatore ha rimosso uno o più elementi.

Mentre se il consumatore vuole rimuovere un elemento dal buffer e vede che il buffer è vuoto "va a dormire" finché il produttore non inserisce qualcosa nel buffer e lo risveglia.

```
1.  #define N 100                          /* numero di posti nel buffer
2.  int count=0;                           /* numero di elementi nel buffer
3.
4.  void producer(void){
5.   int item;
6.
7.   while(TRUE){
8.    item=produce_item();                 /* produco un elemento
9.     if(count==N){                       /* se il numero di elementi nel buffer è N
10.     sleep();                           /* chiedo allo scheduler di addormentarmi
11.    }                  /* quando lo sleep termina o se non ho il buffer pieno,allora:
12.   insert_item(item);                   /* inserisco l'oggetto nel buffer condiviso
13.   count=count+1;                   /* aumento il numero di oggetti presenti nel buffer
14.    if(count==1) wakeup(consumer);      /* se il buffer era vuoto allora ci sarà sicurament
    e un consumatore
15.  }
16. }
17. void consumer(void){
18.  int item;
19.
20.  while(TRUE){
21.    if(count==0){                       /* se il buffer è vuoto
22.     sleep();                           /* vado a dormire
23.    }
24.   item=remove_item();                  /* toglie un elemento dal buffer
25.   count=count-1;                       /* decrementa il numero di elementi nel buffer
26.    if(count==N-1){              /*se il buffer era pieno,allora ci sarà un prod in sleep
27.     wakeup(producer);                  /* sveglio un produttore
28.    }
29.   consumer_item(item);                 /* stampa elemento
30.  }
31. }
```

Questa soluzione non funziona perché non rendo atomiche le operazioni e in questo caso la variabile condivisa è *count*.

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

Possiamo infatti considerare questo scenario:
Il consumatore legge la variabile count e verifica che è uguale a 0 e sta per entrare nell'if.
Prima di invocare lo sleep, il consumatore viene interrotto dal dispatcher, che riattiva il produttore
Il produttore crea un nuovo dato da elaborare, lo mette nel buffer e incrementa count.
Poiché il buffer era vuoto prima di questa operazione, il produttore invoca wakeup per risvegliare i processi consumatori addormentati.
Sfortunatamente, l'interruzione del consumatore è avvenuta prima che il consumatore invocasse sleep e per questo wakeup non ha alcun effetto. Non appena il controllo tornerà al consumatore, la procedura sleep verrà completata, impedendo l'esecuzione del consumatore.
Il produttore andrà avanti fino a che non riempirà il buffer, dopodiché eseguirà anch'esso sleep.
In questo modo entrambi i processi rimarranno per sempre addormentati, raggiungendo una situazione di stallo (deadlock).

Strutture dati evolute

Per riuscire a trovare una soluzione comoda che riesca a risolvere il problema senza scomodare la CPU e senza fare busy waiting abbiamo bisogno di strutture dati evolute che riescano ad integrarsi con il kernel. Possiamo utilizzare il semaforo o il monitor.

Semaforo

Un semaforo è una struttura dati su cui ho due operazioni atomiche, cioè che sono garantite dal sistema operativo. Un'operazione di **DOWN** e un'operazione di **UP**.
DOWN: controlla che il valore sia maggiore di 0, allora decrementa il valore, altrimenti va in sleep.
UP: Incrementa il valore del semaforo, risvegliando i processi bloccati.

Semaforo binario (o mutex):

è una variabile che può assumere due stati; bloccato(1) e non bloccato (0). Utile per gestire la mutua esclusione di alcune risorse.

L'operazione di DOWN si interfaccia con il sistema operativo e viene garantita come operazione atomica. I semafori possono essere implementati con una TLS.

Quanti semafori servono per gestire un produttore o consumatore? O due o tre semafori. Uno per garantire la mutua esclusione e la struttura dati e uno per mettere in sleep il produttore o il consumatore quando serve.

```
1.  #define N 100                      /* numero di posti nel buffer
2.  typedef int semaphore;             /* i semafori sono un tipo speciale di input
3.  semaphore mutex=1;                 /* è un semaforo binario e mi garantisce la mutua e
    sclusione, cioè controlla l'accesso alla regione critica
4.  semaphore empty=N;                 /* il numero di posizioni da riempire nel buffer

5.  semaphore full=0;                  /* conta il numero di posti occupati nel buffer
6.
7.  void producer(void){
8.   int item;
9.
10.  while(TRUE){
11.   item=produce_item();             /* produco un elemento
12.   down(&empty);                    /* decrementa il contatore empty
13.   down(&mutex);                  /* blocco tutto per entrare nella regione critica
14.   insert_item(item);               /* inserisco l'elemento nel buffer
15.   up(&mutex);                      /* lascio la regione critica
16.   up(&full);                       /* incremento il contatore dei posti pieni
17.  }
18. }
19. void consumer(void){
20.  int item;
21.
22.  while(TRUE){
23.   down(&full);                    /* decrementa il contatore full
24.   down(&mutex);                /* blocco tutto per entrare nella regione critica
25.   item=remove_item();             /* prende l'elemento dal buffer
26.   up(&mutex);                     /* lascio la regione critica
27.   up(&empty);                     /* incremento il contatore dei posti vuoti
28.   consume_itemm(item);            /* consuma l'elemento
```


Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

```
29.  }
30. }
```

Monitor

```
1.  static class our_monitor{                       // monitor
2.  private int buffer[]=new int[N];                // dimensione buffer
3.  private int count=0,lo=0,hi=0;                  // lo è l'indice del posto nel buffer da dove
     deve essere prelevato l'elemento successivo. hi è l'indice del posto nel buffer dove dev
    e essere messo l'elemento successivo.
4.   public synchronized void insert(int val){
5.    if(count==N) go_to_sleep();                   // se il buffer è pieno, vai a dormire
6.     buffer[hi]=val;                              // inserisce un elemento nel buffer
7.     hi=(hi+1)%N                                  // calcolo il posto dove devo inserire l'elem
    ento successivo
8.     count=count+1;                               // aumento il contatore che segna il numero d
    i elementi presenti nel buffer
9.    if(count==1) notify();                        // Se il consumatore dorme allora lo sveglia

10.  }
11.
12.  public synchronized void insert(int val){
13.   int val;
14.    if(count==0) go_to_sleep();                  // Se il buffer è vuoto allora vai a dormire

15.   val=buffer[lo];                               // Preleva un elemento dal buffer
16.   lo=(lo+1)%N;                                  // posto da cui prendere l'elemento successiv
    o
17.   count=count-1;                                // decremento il contatore
18.    if(count==N-
    1) notify();                  // Se il produttore dorme allora lo svegli
19.   return val;
20.  }
21.
22.  private void go_to_sleep(){
23.   try{
24.    wait();
25.   }catch(InterruptedException exc){}
26.  }
27. }
```

Un monitor è una raccolta di procedure che vengono raggruppati in un modulo rappresentano la mia regione critica. I monitor vengono utilizzati da Java. In un monitor può essere attivo un solo processo. Vengono utilizzati in sostituzione di sleep e wakeup le operazioni di wait() e notify(). Etichettare una funzione o una variabile con synchronized vuol dire che questa fa parte del monitor e solo all'interno del monitor è possibile utilizzare wait() e notify().
Al contrario dei semafori, nei monitor è il compilatore a gestire la mutua esclusione (mentre nei semafori è compito del sistema operativo), quindi con meno probabilità che qualcosa vada storta.

Semafori e Monitor

È possibile implementare un semaforo usando un monitor, cioè la regione critica andrà da down ad up ed è dove inserirò il synchronized e dopo faccio down implementato come: controllo la variabile interna, se la variabile è uguale a 0 faccio sleep. Up controllo la variabile interna se questa è uguale a 1 allora faccio notify. Per implementare il DOWN andrò a controllare la variabile interna e se è uguale a 0 allora la sostituisco con un sleep(), altrimenti se è uguale a 1 allora faccio la notify().

È possibile implementare un monitor usando i semafori, cioè l'ingresso a tutti i pezzi di codice che compongono la regione critica sono soggetti al mutex.

Barriere

Le barriere sono riferite a gruppi di processi in quanto la loro esecuzione deve essere allineata. Questa soluzione potrebbe servire in un flusso multimediale (audio, video e sottotitoli devono esser allineati).
Alcune applicazioni sono suddivise in fasi e hanno la regola che nessun processo può proseguire nella fase successiva finché tutti i processi non sono pronti a procedere alla fase successiva. Questo è possibile se aggiungiamo una barriera alla fine di ogni fase. Quando un processo raggiunge la barriera è boccato finché tutti i processi non raggiungono la barriera.

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

In Java questo si ottiene con notifyAll(), mentre i thread che raggiungono la barriera si mettono in wait(). Quando tutti i thread sono in wait() un thread esterno esegue una notifyAll() e li risveglia tutti insieme, ma non sarà noto l'ordine di schedulazione e i thread dovranno ri-acquisire il lock per entrare nella regione critica

Il problema dei 5 filosofi

Sono presenti 5 filosofi e ognuno ha un piatto di spaghetti che vuole mangiare, ma per farlo ha la necessità di acquisire due forchette. Il piatto è la risorsa locale, le forchette sono le risorse condivise. Nel momento in cui il filosofo riesce ad acquisire la forchetta di destra e quella di sinistra allora potrà mangiare. Nel caso in cui tutti i filosofi prendono la forchetta di destra contemporaneamente, nessuno sarà in grado di prelevare la forchetta alla sua sinistra e si andrebbe a verificare una condizione di deadlock.
Una possibile soluzione è quella di utilizzare un array di semafori per ogni filosofo, così i filosofi che hanno fame possono essere bloccati se le forchette sono occupate. Un filosofo prima di mangiare deve controllare che i suoi vicini non stiano mangiando anche loro. È possibile far mangiare almeno due filosofi opposti.

Scheduling

Lo scheduling è un componente del sistema operativo che implementa gli algoritmi di scheduling il quale, dato un insieme di richieste di accesso ad una risorsa, stabilisce un ordinamento temporale per l'esecuzione di tali richieste.
Oltre a prelevare il processo idoneo da eseguire, lo scheduler deve preoccuparsi anche di fare un uso efficiente della CPU, dato che lo scambio di processi è un'attività onerosa.
Infatti è necessario cambiare dalla modalità utente alla modalità kernel, salvare lo stato del processo corrente, memorizzando i suoi registri in modo che possano essere richiamati in seguito. Successivamente viene scelto un nuovo processo attraverso l'utilizzo dell'algoritmo di scheduling corrente e viene caricata la MMU con la mappa di memoria del nuovo processo. Infine viene fatto ripartire il nuovo processo.
I processi si comportano in due differenti modi:

1. I processi chiamati **CPU baund**, che spendono la maggior parte del loro tempo in calcoli
2. I processi chiamati **I/O baund**, che spendono la maggior parte del loro tempo in attesa di input/output

Lo scheduling viene effettuato quando:

1. Alla creazione di un nuovo processo
2. Alla terminazione di un processo
3. Al blocco di un processo su un'operazione di I/O

Categorie dei sistemi di scheduling

Vi sono diversi algoritmi di scheduling per diversi ambienti.

BATCH: è un sistema che esegue un processo e quando termina esegue il processo successivo. Non ci sono utenti impazienti in attesa di una risposta veloce.

INTERATTIVI: il flusso della CPU rimbalza tra un processo e l'altro

REAL-TIME: è un sistema che si impone delle scadenze e risponde con dei tempi che sono compatibili con i tempi di operazione del suo utilizzatore. Si dividono in due famiglie e sono:

a) Hard real-time: essi assegnano una scadenza temporale per ogni attività, e l'attività deve esser terminata in quell'istante altrimenti provoca un danno irreparabile al sistema
b) Soft real-time: sono sistemi che assegnano delle scadenze, ma con una tolleranza. Il superamento della scadenza assegnata provoca un degrado delle prestazioni, ma no un danno irreparabile.

Gli obiettivi degli algoritmi di scheduling in diverse circostanze
Sono presenti dei tratti comuni tra **Tutti i sistemi**

Equità – tutti i processi, su un lungo periodo, avranno una equa condivisione della CPU

Regole di sistema – le regole applicate al sistema devono essere messe in atto senza eccezioni

Bilanciamento – il sistema di scheduling ha il compito di non sprecare risorse e di tener impegnate tutte le parti del sistema.

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

Ognuna delle famiglie di sistema ha i propri obiettivi

Sistemi Batch

Throughput	di massimizzare il throughput, cioè di completare un gran numero di job per ora. Quindi di minimizzare i tempi in cui la risorsa è inutilizzata.
Turnaround time	è il tempo che intercorre da quando un job si presenta al sistema a quando questo termina e si ha a disposizione l'output. Per cui si è sempre cercato di ottenere un tempo di turnaround il più breve possibile perché questo permette di massimizzare il numero di job.
Utilizzo della CPU	tener sempre occupata la CPU, all'epoca non era presente un grande numero di processi I/O

Sistemi Interattivi

Tempo di risposta	è il tempo che il processo impiega in coda prima di essere schedulato per la prima volta
Proporzionalità	i tempi di risposta devono essere compatibili a quelli che l'utente si aspetta.

Sistemi Real-time

Riuscire a soddisfare le scadenze senza perdere i dati

Prevedibilità	riuscire a predire il carico del sistema per prepararsi

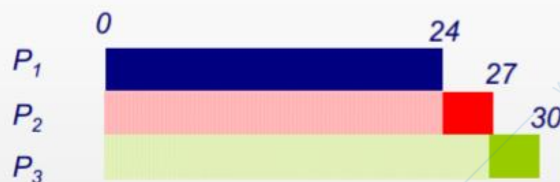
Schedulazione nei sistemi Batch

FIRTS-COME FIRST-SERVED (FCFS)

È il più semplice di tutti gli algoritmi di scheduling ed è un algoritmo <u>non premptive</u>, cioè viene scelto un processo da eseguire e viene eseguito finché questo non si blocca o termina, rilasciando volontariamente la CPU. Con questo algoritmo i processi sono assegnati alla CPU nell'ordine in cui la richiedono. È presente una singola coda dei processi in stato di pronto. L'equità di questo algoritmo è la sua caratteristica. Potrebbe essere ideale in accoppiato con l'algoritmo NCFS

Processo	Tempo di arrivo	Tempo di burst
P_1	0	24
P_2	0	3
P_3	0	3

Diagramma di Gantt



Tr: $P_1 = 0$; $P_2 = 24$; $P_3 = 27$ $\overline{Tr} = (0 + 24 + 27)/3 = 17$

Tta: $P_1 = 24$; $P_2 = 27$; $P_3 = 30$ $\overline{Tta} = (24 + 27 + 30)/3 = 27$

Tr= tempo di risposta, cioè quando il processo acquista la CPU

$\overline{Tr}$= tempo di risposta medio, in media i processi hanno aspetto per accedere alla CPU

Tta= tempo di turnaround, cioè quando viene rilasciato il processo

$\overline{Tta}$= tempo di turnaround medio

Tempo di attesa= il tempo trascorso da un processo nella coda di ready.

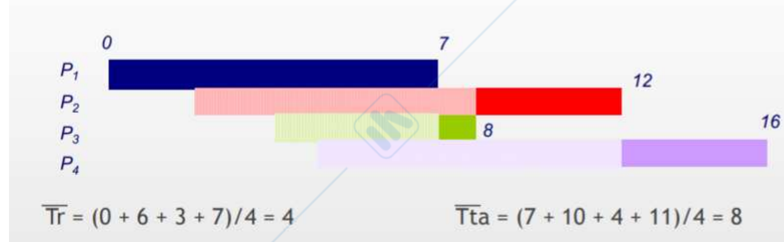
Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

SHORTEST JOB FIRST (SJF)

È un algoritmo non preemptive e parte con il presupposto che i tempi di esecuzione siano conosciuti in anticipo. Il job più breve viene servito per primo dopo che il processo in esecuzione è terminato. Questa politica è ottima solo nel caso in cui tutti i job siano disponibili contemporaneamente

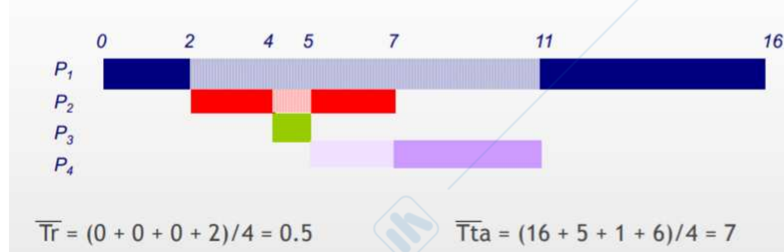
Processo	Tempo di arrivo	Tempo di burst
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4



SHORTEST JOB FIRST CON PRELAZIONE (SRTF)

L'algoritmo Shortest Remaining Time First. Lo schedulatore sceglie sempre il processo il cui tempo di esecuzione residuo è il più breve; anche in questo caso il tempo di esecuzione deve essere noto in anticipo. Quando arriva un nuovo job, il suo tempo totale viene confrontato con il tempo residuo dei processi concorrenti; se il nuovo job ha bisogno di meno tempo per terminare rispetto i processi concorrenti, questi vengono sospesi e viene iniziato il nuovo job.
Viene utilizzato soprattutto in un sistema interattivo.
È possibile utilizzare questo algoritmoo se si ha la necessità di utilizzare poca memoria, perché SJF con prelazione mantiene i job in memoria il meno tempo possibile.

Processo	Tempo di arrivo	Tempo di burst
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4



Scheduling nei sistemi interattivi

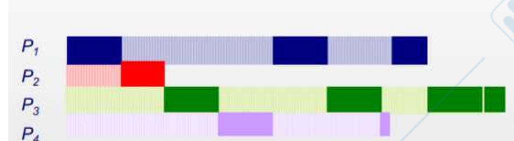
Questi algoritmi vengono utilizzati nei sistemi interattivi, ma possono essere usati anche nei sistemi batch

SCHEDULING ROUND ROBIN

È uno degli algoritmi più utilizzati. Ad ogni processo è assegnato un intervallo di tempo chiamato *quanto* durante il quale gli è consentito l'esecuzione di un job. Se alla fine del quanto di tempo il job non è giunto al termine, viene forzato inserendolo alla fine della coda e viene eseguito il job successivo.
Il vero problema di questo algoritmo è la scelta del quanto di tempo, perché se il quanto è troppo breve provoca troppi cambi di contesto e peggiora l'efficienza della CPU, ma assegnare un tempo molto grande può provocare tempi di risposta lunghi per job brevi.

Processo	Tempo di arrivo	Tempo di burst
P_1	0	53
P_2	0	17
P_3	0	68
P_4	0	24

Quanto di tempo = 20



Quanto di tempo = 10

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

Scheduling con priorità

Ad ogni processo viene assegnata una priorità e viene eseguito il processo con priorità più alta. Le priorità possono essere di diverso tipo, abbiamo:

- Priorità statica: viene applicato una priorità al processo da quando nasce e non cambia durante la sua vita
- Priorità dinamica: viene applicata una priorità al processo quando nasce e viene ricalcolata durante il suo ciclo di vita

Il compito che ha il sistema operativo è quello di fare una valutazione della priorità, cioè considerano tutti i processi che si trovano nello stato di ready (cioè quei processi che sono pronti per acquisire il processore) selezionano il processo con la priorità più alta. Se ho due processi con la stessa priorità viene scelto uno dei processi a caso.
Questa soluzione la possiamo applicare a tutti gli algoritmi, ad esempio

- FCFS: la priorità è il tempo di arrivo.
- SJF: la priorità è calcolata in maniera statica in base al tempo di burst. (con 0 priorità più alta e 100 con priorità bassa)
- SJF con prelazione: la priorità è calcolata in maniera dinamica in base al tempo di burst
- Round Robin: tutti i processi hanno la stessa priorità.

Per implementare le priorità i processi (all'interno del PCB) che hanno la stessa priorità vengono raggruppati in classi di priorità (array) e viene eseguito il processo, scelto in base allo scheduling, all'interno della classe con priorità più alta. Ogni volta che il processo termina il suo quanto di tempo subisce il ricalcolo della sua priorità. Se il processo ha terminato il suo quanto di tempo a disposizione, ma non ha finito il lavoro, allora viene ricalcolata la priorità spostandolo in una priorità più bassa. Solitamente si pensa a servire per primi i processi I/O baund, cioè quei processi con il burst breve.
In Windows ad esempio è in grado di spostare un processo da una classe all'altra mentre questi sono in attesa. Windows prevede 128 livelli di priorità divisi per banchi. Quando creo un processo, all'interno della system call devo dichiarare che tipo di processo è, se è un processo di sistema, se è un processo di I/O o un processo di calcolo. Windows farà alternare la sua priorità in un ranger limitato a bachi di 64 code. Se il processo è dichiarato I/O uso le 64 code più importanti, per i processi di calcolo uso le 64 code meno importanti, per i processi di sistema uso le code in mezzo. In più un processo ha la possibilità di risiedere in una coda che non lo appartiene semplicemente perché è da troppo tempo che aspetta la CPU.
Lo scheduler può diminuire la priorità ad ogni interrupt del clock e nel caso la priorità diventi minore del processo precedente allora ci sarà uno scambio.

Esistono altri metodi per implementare un sistema di scheduling, e sono:

Guaranteed Scheduling

Scheduling garantito, la CPU viene equamente divisa tra gli utenti. È necessario che il sistema tenga traccia di quanta CPU ogni processo ha ottenuto dalla sua creazione. Viene calcolato. L'algoritmo stabilisce di mandare in esecuzione il processo che ha il rapporto più basso, fintantoché il suo rapporto non sale al di sopra del rapporto del suo avversario più vicino.

Lorrery

È una politica che assegna ad ogni processo un "biglietto". Quando tutti i biglietti sono stati distribuiti lo scheduler estrae un biglietto a caso. Il processo che detiene quel biglietto andrà in esecuzione e il biglietto andrà distrutto. Ogni processo può ottenere più biglietti e quindi avrà più possibilità di acquisire la risorsa, in questo modo assegno una priorità maggiore.

Fair sharing

Ha il compito di sparire in modo equo l'utilizzo della CPU tra gli utenti, senza considerare il quantitativo di lavoro che ha ognuno di essi. Esempio, se si hanno 10 processi da schedulare, di cui 9 sono richiesti dall'utente A, mentre un processo è richiesto dall'utente B. Utilizzando il Fair-Share essendoci due utenti gli viene assegnato la stessa quantità di utilizzo della CPU, quindi il 50% ad entrambi. Al sistema non importa che 9 processi verranno schedulati al 50% della CPU.

Real-Time

Un processo nasce dichiarando che la sua schedulazione deve esser portata a termine entro un certo istante temporale. I sistemi Real-time vengono strettamente associati alla **Condizione di schedulabilità** cioè:

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$$

La sommatoria su *m* processi dove *i* è l'indice con cui conto i processi. Del tempo di burst richiesto diviso dal deadline (scadenza). La sommatoria tra tutti i processi deve essere minore di 1.

La frazione è quasi sempre inferiore a 1 perché il valore della deadline è quasi quanto il valore del burst (esempio, devo eseguire 3 unità di tempo di lavoro entro 10 unità di tempo. 3/10).
La possiamo leggere come la frazione di CPU che è necessario dedicare al processo al fine di poterlo schedulare entro la sua scadenza.
La **condizione di schedulabilità** richiede che in ogni istante la somma della frazione del calcolo di lavoro che devo dare a tutti i processi non ecceda il 100% della CPU.

- Se eccede il 100% della CPU, allora la sommatoria sarà maggiore di 1 e per questo non è possibile garantire la schedulazione real-time.
- Se la condizione di schedulabilità è verificata allora esiste una sequenza di allocazione alla CPU ai vari processi che mi garantisce il rispetto di tutte le scadenze, ma è possibile che l'algoritmo non riesca a trovare questa sequenza.

Thread scheduling

Lo scheduling si differenzia a seconda che siano thread utente o thread a livello del kernel.

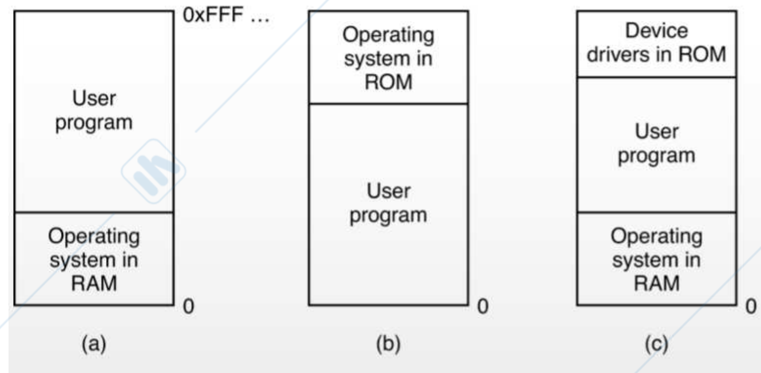
1. Se i thread sono implementati nello spazio utente allora il kernel non sarà a conoscenza dell'esistenza dei thread. Lo scheduling sarà a livello di processo, mentre all'interno sarà presente uno scheduler a livello dei thread.
2. Se i thread sono implementati nello spazio del kernel, allora sarà presente uno scheduler tra i thread. In questo caso si andrà a dare una priorità al singolo thread anziché al singolo processo

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

GESTIONE DELLA MEMORIA

La memoria è un array di locazioni in cui è possibile inserire delle informazioni. Una word è l'allineamento minimo con cui è possibile inserire dati all'interno della memoria. La memoria fisica (RAM) è organizzata da un insieme di indirizzi che vanno da zero a un numero massimo e ogni indirizzo corrisponde ad una cella contenente un certo numero di bit, solitamente 8.



Possiamo avere tre possibilità su come è possibile configurare la memoria.

a) È solitamente quello che capita quando si accende il PC, cioè all'accensione del pc vengono lette le prime tracce presenti sul disco. Troverà le informazioni del kernel il quale viene copiato in memoria a partire dalla posizione zero in memoria. Il resto della memoria è dedicata ai programmi utenti.
b) La seconda possibilità è quella di inserire il sistema operativo nella ROM, in questo modo se il dispositivo viene spento il SO rimane, cioè nella ROM. I processi applicativi iniziano dalla posizione fisica zero.
c) L'ultima soluzione è composta dal sistema operativo presente all'inizio della memoria. Mentre nella ROM sono presenti dei driver di dispositivi che non possono essere modificati, questa viene chiamato BIOS del sistema.

I modelli a) e c) hanno lo svantaggio che un difetto nel programma utente possa cancellare il sistema operativo.

Memoria senza astrazione

Per avere più processi nella memoria, in un sistema che non ha astrazione della memoria, è possibile caricare i processi uno dietro l'altro e viene utilizzata la RILOCAZIONE STATICA che consiste nel sommare la posizione assoluta a ogni indirizzo del programma. In questo modo si evita che un processo possa interferire con lo spazio di memoria di altri processi. (????ASSOLUTA O RELATIVA)

I problemi principali che si deve cercare di risolvere sono quelli di PROTEZIONE e di RIPOSIZIONAMENTO.

Una soluzione è quella di assegnare a ogni processo il suo spazio degli indirizzi. Uno **spazio degli indirizzi** è l'insieme degli indirizzi che un processo può usare per indirizzare la memoria.

Registro Base e Registro Limite

La CPU ha a disposizione due registri chiamati *registro base* e *registro limite*. Il registro base è l'indirizzo fisico da dove incomincia il programma in memoria, mentre nel registro limite è memorizzata la lunghezza del programma. Nel PCB vengono inseriti i registri base e il registro limite.

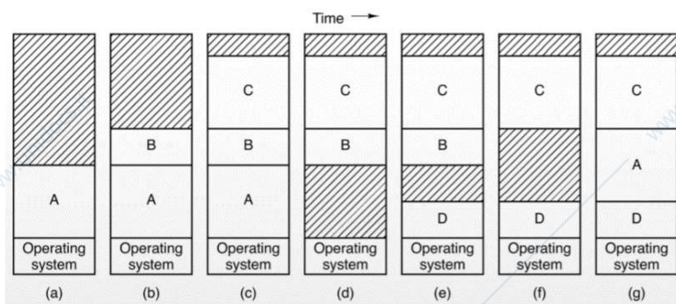
Ogni volta che la CPU tenta di scrivere, sul bus indirizzi viene sommato il valore del registro base e si effettua un confronto tra valore del registro limite e l'indirizzo generato, in modo tale che quest'ultimo non sia superiore al registro limite.

1- Nel caso in cui è superiore viene generato un errore chiamato Segment not found, il processo in questo caso non può andare avanti perché il processo sta invadendo una zona di memoria non dedicata a lui.

Questo meccanismo viene chiamato **uso della memoria per segmenti**, cioè viene assegnato ad ogni processo un segmento di memoria. Il problema è che questi segmenti sono difficili da gestire in quanto vengono creati e distrutti per molteplici processi.

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari



Questa operazione di **SWAPPING**, viene chiamata anche **scheduling a lungo periodo**. Esistono due diversi scheduling e sono:
1. Lo scheduling che alloca la CPU ad un processo, chiamato **scheduling a breve periodo**
2. Lo scheduling che decide che per motivi di prestazione un processo viene trasferito interamente sul disco.

Allocazione dello spazio

L'allocazione dello spazio è complicata in quanto ogni processo è dinamico. Infatti un processo durante il suo ciclo di vita può aumentare la sua dimensione di dati. Per questo motivo è necessario che il segmento in cui è ingabbiato possa crescere o restringere in base alla necessità. La memoria occupata da un processo è costituita da:

1. segmento di testo del programma. Dove è presente l'eseguibile
2. un pezzo di dati, che è diviso in due
   a. Lo *Stack* che parte dalla locazione più alta e scende quando vengono aggiunti altre informazioni (A-Stack)
   b. *Data segment* dove sono presenti le variabili, e questo spazio cresce man mano che si inseriscono altre informazioni (A-Data)

Nel caso in cui i due spazi, A-stack e A-Data, collidono si avrà un errore in quanto lo spazio disponibile tra i due sarà esaurito, senza esser riuscito ad aumentare lo spazio in tempi ragionevoli. Questo errore solleva un TREP che si chiama Sleep Underflow, cioè una sovrapproduzione sullo stack. In questa situazione il sistema operativo non riesce più a garantire la consistenza dei dati, perché lo stack potrebbe aver già sovrascritto dei dati.

Tracciamento della memoria disponibile

Per effettuare il tracciamento della memoria disponibile ho bisogno di strutture dati di supporto all'interno del kernel.

1- Una possibile soluzione è utilizzare un sistema dotato di una **BITMAP**, cioè la memoria è divisa in unità di allocazioni, ogni unità corrisponde a un bit della bitmap, dove 0 corrisponde se l'unità è libera e 1 se è utilizzata. Questa soluzione è stata utilizzata soprattutto nei sistemi mainframe e conviene utilizzarla se la memoria è piccola. Infatti la dimensione della tabella bitmap sarà proporzionale alla dimensione della memoria. Infatti se la dimensione della tabella bitmap è grande, il problema non è lo spazio che questa occupa in memoria, ma l'operazione di ricerca di uno spazio libero necessario al processo.

2- La seconda soluzione è quella di utilizzare una **lista collegata**, cioè sarà presente all'interno del kernel una struttura di liste di segmenti, ed ogni segmento contiene o uno spazio vuoto o un processo, l'indirizzo da cui parte, la lunghezza e il puntatore alla voce successiva. La lista solitamente occupa più spazio della bitmap, ma ci permette di velocizzare la ricerca degli spazi vuoti.

Allocare e deallocare continuamente zone di memoria provoca la **Frammentazione esterna**, cioè la presenza di spazio libero ma non contiguo. È possibile eseguire della compattazione della memoria, ovvero sposta tutti i processi, copiando il loro contenuto (molto costoso) e li sposta in cima alla memoria. Compattare la memoria è l'ultima soluzione, infatti si preferisce prevenire utilizzando algoritmi in grado di allocare della memoria per un nuovo processo. Questi algoritmi sono:

FIRST FIT(IL PRIMO DOVE CI STA)

alloco spazio, scorro la lista e il primo spazio sufficientemente grande viene utilizzato. Utilizzando questo algoritmo su lungo periodo non restituisce delle buone prestazioni, in quanto alloca lo spazio sempre all'inizio della memoria.

NEXT FIT(IL PROSSIMO DOVE CI STA)

per allocare dello spazio parto dall'inizio della memoria e trovo il primo vuoto disponibile sufficientemente grande e lo alloco (magari non tutto). La prossima allocazione parte da dove ero arrivato in precedenza. In questo modo tendo a sparpagliare lungo tutto lo spazio disponibile. Ha la prerogativa di abbassare i tempi e di sparpagliare uniformemente lo spazio allocato all'interno della mia memoria. Mi penalizza se ho molti processi grandi.

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

BEST FIT(LA TAGLIA MIGLIORE)
esso guarda prima tutta la lista collegata e prendo lo spazio disponibile più piccolo tra quelli che possono accomodare la richiesta. È possibile ordinarla per la dimensione libera, questo prevede un algoritmo di ordinamento e per questo rende il tutto più pesante. L'idea alla base del best fit è di ridurre lo spazio di memoria sprecata, creando frammenti di dimensione minima.

WORST FIT(LA TAGLIA PEGGIORE)
Questo algoritmo colloca il processo nello spazio di memoria libera di dimensione maggiore. È l'opposto del Best fit ed è basato sul presupposto che se lo spazio libero di memoria è grande, lo spazio che rimane libero dopo avere allocato un processo è abbastanza grande per contenere altri processi e per non essere sprecato. Ha prestazioni migliori al Best fit

QUICK FIT
Questo algoritmo mantiene liste separate per le zone di memoria con dimensioni maggiormente richieste. Ha il vantaggio di essere veloce nella ricerca di uno spazio libero di dimensioni fisse. Il problema sorge quando bisogna aggiornare le liste, in quanto queste operazioni richiedono tempo.

Memoria virtuale

Oggi i programmi sono più grandi della dimensione della memoria. Per risolvere questo problema, negli anni 60 si è pensato di utilizzare una tecnica chiamata **Overlay**.
Questa tecnica consisteva nel suddividere il programma in pezzi e questa suddivisione era affidata al programmatore. All'avvio di un programma, in memoria veniva caricato il gestore degli overlay il quale caricava e avviava l'overlay 0 in memoria. Al termine, il gestore degli overlay caricava l'overlay 1 o in modo contiguo all'overlay 0 (nel caso in cui c'era abbastanza spazio libero in memoria) oppure sovrascrivendo l'overlay 0 (nel caso in cui lo spazio in memoria non era sufficiente).
Gli overlay erano tenuti sul disco e portati dentro e fuori la memoria dal gestore dell'overlay. Suddividere grandi programmi in piccoli pezzi richiedeva molto tempo e molto spesso la suddivisione era piena di errori. Per queste problematiche si è pensata una soluzione alternativa, cioè la Memoria virtuale.

La **memoria virtuale** consiste che ogni processo ha il suo spazio degli indirizzi, per questo motivo non sono più utili i registri base e limite, ma è necessario aggiungere dell'hardware chiamato **MMU** (Memory management Unit) che ha il compito di mappare gli indirizzi virtuali con gli indirizzi fisici di memoria. Lo spazio degli indirizzi virtuali è suddiviso in unità di dimensioni fissa (non più segmenti) chiamate **Pagine**. Le unità corrispondenti nella memoria fisica sono chiamate **Frame**.
All'interno della MMU, per ogni processo, è presente una tabella delle pagine. La tabella delle pagine è costituita dagli indirizzi virtuali e fisici e da un bit che indica se la pagina è presente o assente. Quando la MMU intercetta l'indirizzo virtuale capisce a quale pagine stiamo facendo riferimento, trasforma la pagina in un frame trasformando l'indirizzo virtuale in indirizzo fisico. Per eseguire questa trasformazione vengono effettuati i seguenti passaggi:
- i (4) bit più significativi dell'indirizzo virtuale in ingresso sono utilizzati come indice nella tabella delle pagine.
- Viene controllato il bit presente/assente e se questo è uguale allo 0, allora si verifica un TRAP. Se il bit presente/assente è uguale a 1, il numero di frame trovato nella tabella delle pagine viene copiato nei (3) bit più significativi e vengono aggiunti i 12 bit dell'offset senza nessuna modifica nel registro di output.
- Al termine viene inviato sul bus l'indirizzo fisco di memoria.

Nel caso in cui viene fatto riferimento ad una pagina non mappata, la MMU causa un TRAP, chiamato page fault, della CPU verso il sistema operativo. Il compito del sistema operativo è quello di prelevare un frame poco utilizzato e trascriverlo sul disco (se non è già presente). La pagina appena referenziata viene collocata nel frame appena liberato, cambia la mappa e riavvia l'istruzione che era in trap.
Solitamente solo una piccola parte delle voci della tabella delle pagine viene letta frequentemente, il resto è poco utilizzato. In più la tabella delle pagine si trova in memoria, quindi viene usata un pezzo di memoria, per gestire la memoria stessa.
Per questo motivo si è aggiunto un componente, all'interno della MMU, chiamato **TLB** o **Memoria Associativa** che consente di mappare gli indirizzi virtuali con gli indirizzi fisici più utilizzati recentemente, senza utilizzare la MMU. In questo modo la TLB esegue una funzione di buffer. Il TLB consiste in un numero ridotto di voci rispetto la MMU, dove ciascuna voce è costituita dalle seguenti informazioni:
- Il numero di pagina virtuale
- Un bit se vengono effettuate delle modifiche
- Un bit di codice di protezione

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

- Il frame fisico dove si trova la pagina
- Un bit che indica se la voce è valida (cioè in uso) o meno

Quindi quando la MMU intercetta un indirizzo virtuale, guarda se è presente nella TLB.

a) Se è presente allora viene utilizzata la TLB senza passare dalla MMU.
b) Se l'indirizzo virtuale non è presente nel TLB, la MMU rileva una **TLB miss** e cerca nella sua tabella delle pagine. Se questa è presente allora l'indirizzo richiesto viene inserito nella TLB e inviato.

Oggi la maggior parte dei dispositivi preferiscono adottare una soluzione software piuttosto che far gestire le TLB miss dall'hardware della MMU. Infatti le voci della TLB vengono caricate dal sistema operativo e quando si verifica una TLB miss, invece di lasciare il compito alla MMU, viene generato un errore di TLB e il sistema operativo ha il compito di trovare la pagina, rimuovere una voce dalla TLB, inserire la nuova pagina e riavviare l'istruzione in errore.
Per migliorare le prestazioni che effettuano la gestione della TLB via software sono state sviluppate varie strategie, ad esempio quella di anticipare le possibili pagine che possono esser richieste successivamente e caricare in anticipo le loro voci nel TLB.
Possiamo avere due tipi di TLB miss:

1- **Soft miss**: avviene quando la pagina di riferimento non è nel TLB ma è nella memoria. In questo caso sarà necessario aggiornare la TLB.
2- **Hard miss**: avviene quando la pagina non è in memoria e nemmeno nel TLB. Per prendere la pagina serve un accesso al disco.

Elementi della tabella delle pagine

Caching disabled
Modified
Present/absent
Page frame number
Referenced
Protection

Gli elementi della tabella delle pagine sono costituti da bit che gestiscono le informazioni, escluso per il **Numero di frame**.
Esiste un bit presente/assente:

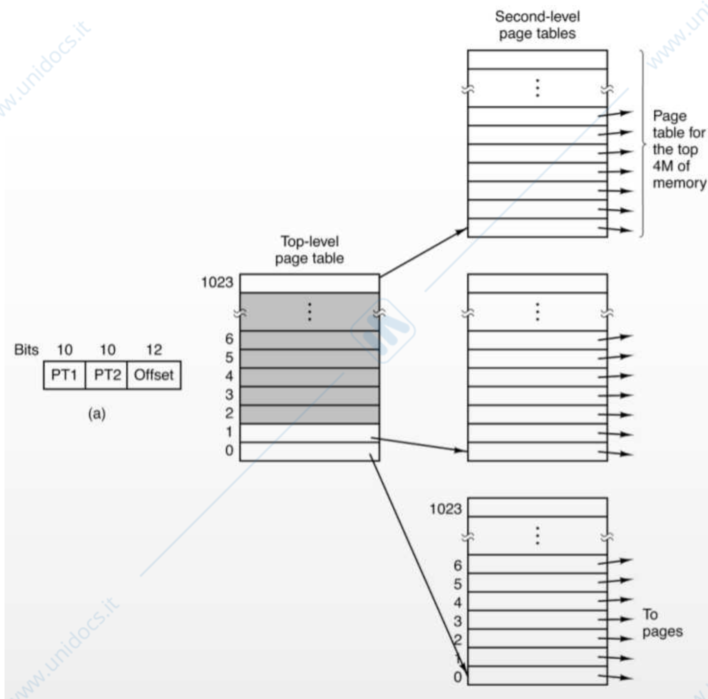
- 1 se è presente
- 0 non è presente

Il bit di **Protezione** specifica quali tipi di accesso sono consentiti, ad esempio solo in lettura o scrittura o entrambi.
Il bit **Modificato** viene impostato quando la pagina è stata modificata in memoria e deve essere copiata sul disco. Se non è stata modificata può esser scartata senza ricopiarla sul disco per risparmiare tempo.
Il bit **Referenziato** viene impostato ogni volta che si fa riferimento alla pagina.
Il bit **Caching** impostato ad 1 impedisce alla pagina di esser spostata sul disco. Simile allo SWAP, ma si chiama paginazione sul disco in quanto lo SWAP viene eseguito sull'intero processo, mentre in questo caso si fa riferimento alla pagina.

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

Tabella delle pagine su più livelli



Dato che ogni processo nel proprio PCB ha un riferimento alla propria tabelle delle pagine e in più la dimensione della tabella è molto grande perché dipende dallo spazio dell'indirizzamento che offriamo al processo e dalla dimensione delle pagine. Per questo è necessario diminuire la dimensione della tabella.
Una soluzione è quella di suddividere l'indirizzo in due parti, la prima parte (PT1) viene usata come indice ad una tabella di primo livello. All'interno di questa tabella sono presenti riferimenti a tabelle delle pagine di secondo livello.
La seconda parte (PT2) dell'indirizzo viene utilizzata come indice sulla tabella delle pagine di secondo livello. La tabella di secondo livello farà riferimento ad un frame.
Ho risparmiato spazio perché la tabella di secondo livello non è obbligatoria che esista, infatti essa esiste solo se fa riferimento ad un frame.
Per cui per il principio di località, cioè durante l'esecuzione di una data istruzione presente in memoria, con molta probabilità le successive istruzioni saranno ubicate nelle vicinanze di quella in corso, nel momento in cui incomincio ad allocare spazio inizierò a allocare spazio in modo contiguo.
Questo approccio ha funzionato fin quando si sono utilizzati dispositivi a 32bit. Successivamente con i dispositivi a 64bit anche questo metodo è fallito perché la dimensione delle tabelle è aumentata drasticamente. La possibile soluzione per continuare ad utilizzare questo metodo era quella di utilizzare molte tabelle di piccole dimensioni (a 5 livelli di tabelle), ma questo comportava uno spreco di tempo per passare da una tabella all'altra ogni volta che era necessario recuperare il frame non presenta nella TLB.
Un'altra soluzione poteva essere quella di allargare la dimensione delle pagine, ma questa causa il problema della **frammentazione interna**. Ovvero se ho una pagina molto grande è difficile che questa venga utilizzata tutta, per questo si avrebbe molto spazio sprecato. Per ridurre la frammentazione interna è possibile ridurre la dimensione delle pagine.
Queste soluzioni non sono adatte e per questo motivo si è pensato di utilizzare la tabella delle pagine invertite.

Tabella delle pagine invertite

La tabella delle pagine invertite è utilizzata soprattutto nei calcolatori a 64bit. Essa consiste in una tabella per processo ed è molto piccola. All'interno della tabella costruisco una funzione chiamata HASH, cioè una funzione che partendo da un dominio mi restituisce una voce, cioè un valore numerico associato ad ogni elemento del dominio all'interno di un insieme limitato. Ovvero presa una pagina ottengo dalla funzione di HASH un numero che va da 0 a 128. Viene diviso l'insieme delle pagine utilizzate in 128 gruppi di pagine omogenei. Quando devo trasformare una pagina in un frame, vengono esaminati l'identificativo della pagina, applico la funzione di HASH che restituisce il numero della famiglia che viene usato come indice nella tabella delle pagine invertite. Questa punterà ad una lista concatenata di record in cui sarà presente il numero di pagina e il numero di frame. Se esiste è proprio in quella lista, per cui l'unica cosa che bisogna fare è necessario scorrere la lista. L'unico vero problema è lo scorrere la lista perché questo comporta dello spreco di tempo, in alcuni sistemi operativi come ad esempio linux invece della lista viene inserito un albero binario

La tabella delle pagine invertite è utilizzata soprattutto nei calcolatori a 64bit. È presente una sola voce per frame nella memoria reale, piuttosto che una voce per pagina. Ogni voce è composta da (processo, pagina virtuale) che si trova nel frame. Lo svantaggio di questa politica si presenta quando lo spazio virtuale degli indirizzi è superiore rispetto alla memoria fisica, in quanto la traduzione da virtuale a fisica diventa difficile. Quando il processo referenzia la pagina virtuale, l'hardware non può più trovare la pagina fisica usando l'indirizzo virtuale nella tabella delle pagine. È necessario che cerchi la voce costituita da (processo, pagina virtuale) nell'intera tabella delle pagine invertite. Possiamo avere due soluzioni.
Se il TLB può contenere tutte le pagine più usate, la traduzione può avvenire velocemente come con le normali tabelle delle pagine.

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

Nel caso di una TLB miss deve essere fatta una ricerca sulla tabella delle pagine invertite. Questa ricerca è possibile grazie all'utilizzo di una ***Hash table*** *sull'indirizzo virtuale. Tutte le pagine virtuali attualmente in memoria che hanno lo stesso valore di hash sono legate insieme. Ogni voce della hash table punterà ad una lista concatenata di record in cui sarà presente la corrispondenza tra la pagina virtuale e il frame. L'unico problema è lo scorrere la lista perché questo comporta dello spreco di tempo, in alcuni sistemi operativi come ad esempio in Linux invece della lista viene utilizzato un albero binario.*

Paginazione su richiesta (Demand Paging)

I frame che associo alle tabelle delle pagine si possono trovare sia in memoria (RAM) che sul disco. In questi casi è necessario marcare nella voce all'interno della tabella delle pagine il bit di caching il quale segnale che il frame è situato sul disco. In questo modo la memoria viene liberata per far spazio ad altri frame. Per selezionare quale frame spostare sul disco o da eliminare è necessario adottare una politica di rimpiazzamento.
Il Demand Paging è quel meccanismo che fa partire questa scelta.

1. Nel momento in cui la MMU non trova una coppia pagina-frame nella TLB viene sollevato una TRAP chiamato page fault, il quale avvisa il sistema operativo. Questa trap viene intercettata dal kernel. Quando viene sollevata una trap il kernel non conosce la sua entità e potrebbe essere o una trap a causa di una divisone per zero, una chiamata (system call) oppure una page fault.
2. Viene bloccato il processo in esecuzione, cioè metto in sicurezza la CPU, lo stato del processo e il registro tutto il necessario nel PCB.
3. Quando viene identificato che la TRAP riguarda una page fault, allora viene interpellato il pezzo di kernel che si chiama Memory manager. Esso trova la pagina che manca nella TLB.
4. Nel caso in cui la memoria è esaurita è necessario liberare un frame in uso. Questo viene fatto solo se non ci sono frame già liberi e viene eseguito un algoritmo che mi dà una politica di rimpiazzamento delle pagine.
5. La pagina che non era presente nella memoria (RAM) viene caricata nel frame che si è liberato.
6. Aggiorno la tabella delle pagine.
7. Ridò il controllo al processo.

Algoritmo sul rimpiazzamento delle pagine

Quando si verifica una page fault è necessario far spazio alla pagina richiesta. Il S.O. deve scegliere quale pagina eliminare e se è stata modificata, mentre era in memoria, deve essere riscritta.

Algoritmo ottimo

viene sostituita la pagina che verrà referenziata il più tardi possibile. Questo algoritmo non è possibile implementarlo perché prevede il futuro, in quanto richiede in anticipo la sequenza delle pagine che devono essere referenziate, e questo non è possibile. Questo algoritmo viene usato come punto di riferimento nello studio di altri algoritmi.

First In First Out (FIFO)

viene sostituita la pagina che è presenta da più tempo nel sistema.

ANOMALIA DI BELADY: L'algoritmo FIFO all'aumentare del numero di frame disponibili aumenta il numero di page fault, anziché diminuirli.

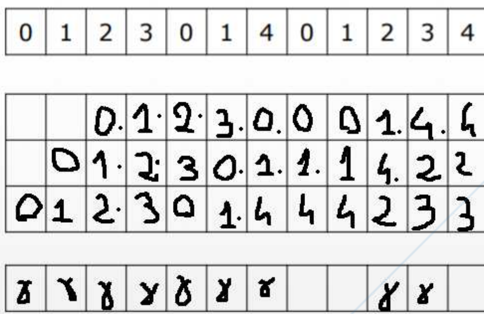
PROPRIETÀ DI INCLUSIONE: l'insieme delle pagine presenti in memoria caricate all'istante t con m frame è sempre un sottoinsieme di quelle caricate all'istante t con m+1 frame. Gli algoritmi che soddisfano la proprietà di inclusione non sono soggetti all'anomalia di Belady.

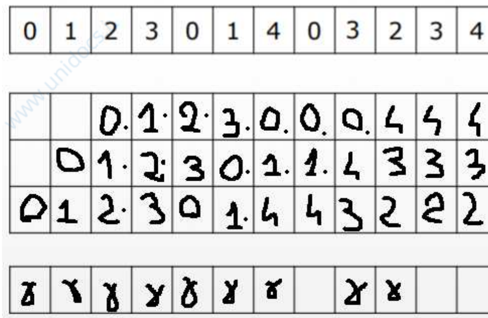
Seconda Chance

viene data una seconda possibilità prima di sostituirla. È necessario ricordare se le pagine sono state referenziate da quando gli è stata assegnata una seconda possibilità. Se la pagina viene referenziata e aveva la seconda chance, questa viene annullata.

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari





Orologio

tutte le pagine si trovano su una lista circolare a forma di orologio. La prima pagina indicata dalla lancetta viene controllata. Se il bit R è 0, allora viene sfrattata, la nuova pagina viene inserita al suo posto nell'orologio e la lancetta spostata in avanti di una posizione. Se R è 1 viene azzerato (gli viene data una seconda chance) e la lancetta avanzata alla pagina successiva, fino a quando non trova una pagina con il bit R uguale a 0.

LRU

viene sostituita la pagina che è in memoria da più tempo e quindi sarà la meno utilizzata. In caso di poche pagine LRU si approssima al FIFO. LRU migliora se allargo lo spazio di memoria perché non subisce le anomalie di belady. (Sposto le pagine in base all'ordine di arrivo)
Questo non viene utilizzato in quanto andrebbe a compromettere le prestazioni del sistema in quanto dovrebbe ordinare spesso le pagine secondo la configurazione.

NRU-Not Recently Used (approssimazione di LRU)

ad ogni pagina associo due bit, un bit che segna se la pagina è stata referenziata, mentre il secondo bit indica se la pagina è stata modificata. Definisco quattro classi per criticità

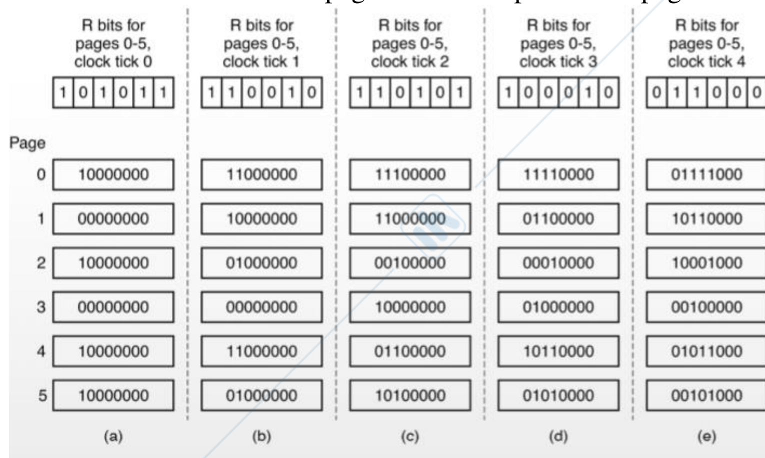
Classe 00: not referenced, not modified. – è la prima che viene scartata
Classe 01: not referenced, modified.
Classe 10: referenced, not modified.
Classe 11: referenced, modified. – probabilmente mi servirà successivamente

Come mai una pagina potrebbe essere modificata e non referenziata? Perché viene applicato un processo di invecchiamento, ovvero periodicamente il kernel seleziona tutte le pagine e azzera il bit di referenziazione

Not Frequently Used (non è LRU)

cerca di assegnare un ordine alle pagine che sia semplice da conservare.

- Ad ogni pagina viene associato un numero identificativo.
- Ad ogni clock il numero identificativo scorre a destra introducendo il bit di referenza come bit più significativo.
- Al momento di un errore di pagina viene rimpiazzata la pagina con numero identificativo minimo.



In questo modo vado ad effettuare una raffinazione rispetto al NRU, in quanto utilizzo un numero maggiori di classi. Ho perso il bit di modifica.

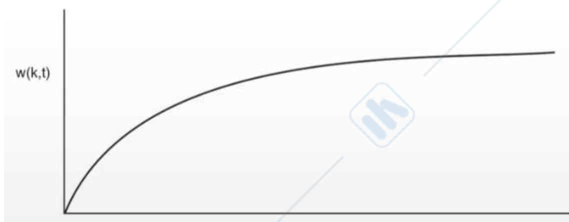
Il rettangolo identifica la pagina.

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

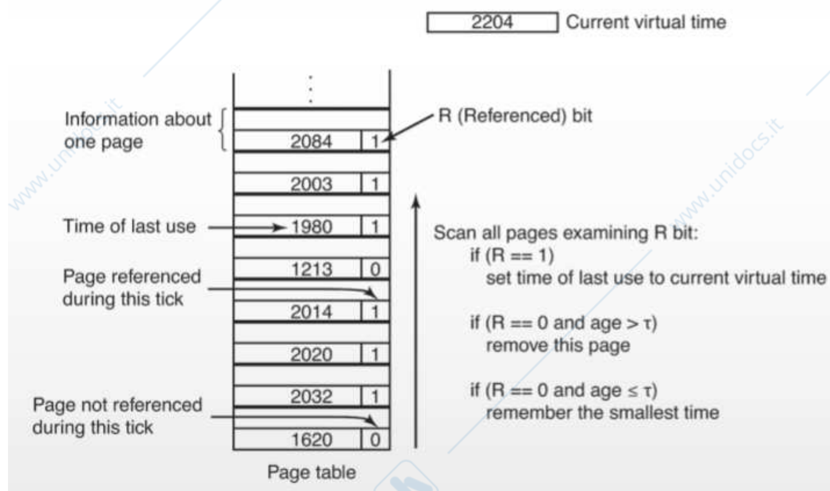
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

Working Set

il working set è l'insieme delle pagine presenti in memoria che un processo ha bisogno per poter essere eseguito senza generare page fault. Sono presenti solo quelle pagine che sicuramente al processo serviranno nel prossimo futuro, quindi non tutte le pagine.



Il WS, sull'asse delle y, ha due dimensioni nel tempo evolve seconda una parabola. Inizialmente sono presenti molti page fault, capita soprattutto quando parte il programma. Nel tempo il WS arriva a una dimensione tale per cui il numero di page fault sono contenuti. L'idea consiste che se siamo a conoscenza di quelle pagine che compongono il WS, allora la pagina vittima sarà quella pagina che non è contenuta nel WS. Non è facile calcolarsi il WS in quanto non è possibile predire le pagine che saranno referenziate nel futuro.



Per questo si è pensato di scrivere del software per capire se una pagina fa parte del WS o meno.

Viene usato l'algoritmo basato sul working set, cioè associamo ad ogni pagina due informazioni di cui uno è un flag da un bit che indica se la pagina è stata referenziata oppure no, la seconda è un'informazione di tipo numerico che indica quando è stata l'ultima volta che la pagina è stata usata. All'interno del calcolatore è presente il valore del tempo virtuale, cioè da quanto tempo la CPU è attiva. È importante che tutti i valori del tempo di ultimo utilizzo di ogni pagina siano inferiori al tempo virtuale e da quanto tempo sono distanti dal tempo virtuale. In questo modo possiamo classificare le pagine in tre categorie:

- **R=1**: se il bit R è uguale a 1 aggiorno il tempo di ultimo utilizzo con il valore del tempo virtuale attuale. In più azzero il bit R dandogli una seconda chance. Poiché la pagina è stata referenziata durante l'ultimo ciclo di clock e chiaramente nel WS.
- **R=0 e età>τ** : le pagine che sono state referenziate e hanno un'età maggiore di τ (TAU), queste non faranno parte del WS
- **R=0 e età<τ**: le pagine che hanno un'età inferiore al valore di τ, queste potrebbero far parte del WS

τ è calcolata su delle stime, mentre l'età è uguale alla differenza tra il tempo virtuale attuale e il tempo di ultimo utilizzo. L'hardware imposta i bit R e M di modifica. Periodicamente viene generato un interrupt del clock che provoca l'azzeramento del bit di referenziato. Ad ogni errore di pagina viene fatta la scansione della tabella delle pagine (le pagine non sono in ordine) per cercare una pagina da rimuovere.

Con questo metodo è possibile approssimare un LRU, lo svantaggio è il tempo impiegato per eseguire la scansione della lista non mi conviene effettuare un algoritmo di ordinamento delle pagine in quanto il confronto per vedere se il bit R è uguale a 1 è molto veloce.

WSClock

corrisponde alla versione del Working set precedentemente annunciato, ma con il clock. In questo modo riprendo la scansione della tabella da dove ho eliminato l'ultima pagina.

Allocazione locale e globale

Gli algoritmi di rimpiazzamento delle pagine possono operare in modalità locale oppure globale.

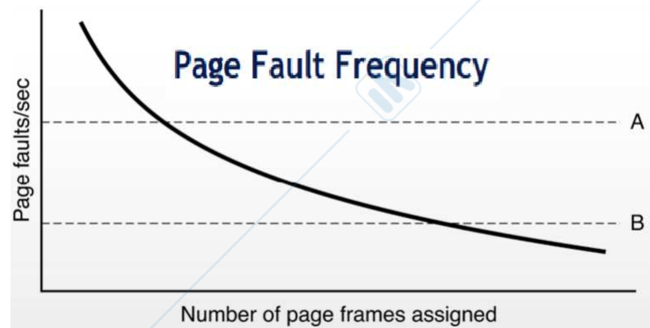
Locale è quando rimpiazzo una pagina all'interno dello stesso processo che ha causato il page fault,

Globale è quando rimpiazzo una qualsiasi pagina su tutti i processi all'interno del sistema.

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

Ad ogni processo gli viene assegnato un Working set di base, questo implica che i processi hanno un working set diverso tra loro, ed è un parametro che assegna il SO. Il WS può allargarsi o restringersi, quando il processo incomincia ad eseguire molti page fault significa che il WS è diventato piccolo. Allargare il WS di un processo implica il ristringimento di un WS di un altro processo.
All'interno del PCB ho una contabilizzazione di quanti page fault sono stati generati dal processo in un'unità di tempo. Nel momento in cui si verificano troppi page fault e supera una certa soglia significa che si è sottostimato il WS del processo e quindi gli è stato assegnato poco spazio per questo motivo si cerca di aumentare la dimensione del WS rimuovendo pagine da un altro processo (modalità globale).
Sotto la soglia vengono usate politiche di rimpiazzamento locali, mentre sopra la soglia vengono usate politiche di rimpiazzamento globali.



Nel caso in cui tutti i processi superano la soglia, quindi hanno il WS troppo grande. Se la somma dei WS dei processi all'interno del sistema supera la capacità della memoria virtuale (cioè tutte le pagine che posso mettere in memoria più tutte le pagine che è possibile inserire sul disco). In questo caso il sistema va in **thrashing** e l'unico modo è quello di uccidere un processo, ma questo è difficile in quanto non è possibile eseguire un processo che uccide un altro processo. Il SO elimina un processo a caso sperando che la situazione migliori.

Paging Daemon

È un processo in background che viene risvegliato periodicamente per ispezionare lo stato della memoria. Nel caso in cui la memoria è saturata il paging daemon si occupa di attuare l'algoritmo di rimpiazzamento delle pagine in uso per recuperare dello spazio.

Dimensione ottimale delle pagine

Non esiste un risultato ottimale per scegliere la dimensione della pagina.
Data p la dimensione della pagina, per ogni pagina ho un record all'interno della tabella delle pagine e chiamo e la dimensione del record all'interno della tabella delle pagine.
Chiamo s la dimensione media del processo e in un dato istante sono presenti n processi.
Date queste informazioni possiamo calcolarci lo spazio sprecato.
Per calcolarci la frammentazione interna, cioè lo spazio sprecato all'interno di una pagina, sarà data da:

$$\frac{np}{2} \qquad (a\ livello\ di\ sistema)$$

Dove per ogni processo ho mediamente mezza pagina sprecata, perché se un processo ha allocate per sé un insieme di pagine e quando inizia a utilizzare lo spazio in memoria le prime pagine saranno riempite interamente mentre l'ultima pagina avrà dello spazio sprecato.
Successivamente avremo lo spazio sprecato indotto dalla tabella delle pagine e dipende dal processo.

$$overhead = \frac{se}{p} + \frac{p}{2}$$

(cioè il tempo medio di CPU necessario per eseguire i moduli del kernel)
Dove:
$\frac{s}{p}$= il numero delle pagine allocate per il processo.
$\frac{se}{p}$=la dimensione della tabella delle pagine
$\frac{p}{2}$=frammentazione interna.
L'idea è quella di trovare la dimensione ottimale della pagina p. Sarà necessario effettuare la derivata prima e la pongo uguale a 0.

$$-\frac{se}{p^2} + \frac{1}{2} = 0 \xrightarrow[trovo\ p]{} p = \sqrt{2se}$$

Se il SO ha dei processi molti grandi mediamente è più corretto allocare delle pagine più grosse, ma non in modo lineare ma proporzionale secondo la radice. Il valore di p deve essere una potenza di due e nel caso in cui non lo è lo si approssima.

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

Istruzioni separate e spazi dei dati

È necessario distingue due tipi di pagine. Le pagine che contengono solo il codice del programma (**I-space**) e le pagine che contengono i dati (**D-space**). Anche a livello di indirizzamento le pagine I-space e D-space vengono separate
È possibile eliminare una pagina che contiene il testo del programma in quanto è possibile recuperarla dall'eseguibile, mentre non è possibile eliminare le pagine che contengono i dati in quanto questi sono stati ricavati durante l'esecuzione. Questa separazione ci permette di utilizzare le **pagine condivise**.

Pagine condivise

Infatti se vengono eseguiti più processi dello stesso programma per evitare due copie delle stesse pagine è opportuno condividerle. Possono essere condivise solo le pagine in sola lettura, quindi quelle che si trovano nello spazio I-space. Mentre la tabella delle pagine appartenenti al D-space rimane separata.
Per rendere possibile questo è necessario che ogni processo, presente nella tabella dei processi, abbia due puntatori di cui uno che fa riferimento alla tabella delle pagine I-space, che può essere condiviso da altri processi, mentre l'altro puntatore alla tabella delle pagine D-space.

È possibile ottenere le pagine condivise anche senza la suddivisione in D-space e I-space. Ciascun processo punta alle stesse pagine, ma le pagini dati vengono mappati solo in modalità lettura. Finché i processi leggono solamente non viene causata nessuna TRAP, ma appena un processo tenta di aggiornare una pagina con bit di protezione di solo lettura causa una TRAP al SO. Il quale esegue una copia della pagina che ha causato l'errore in modo che ogni processo abbia la sua copia privata. In questo modo entrambe le copie vengono impostate come read-write, così le successive scritture su una delle due copie verranno eseguite senza TRAP. Questo comporta che le pagine che non vengono mai modificate non necessitano di essere copiate. Tale approccio viene chiamato **Copy On Write**.

Librerie condivise

Oggi vengono realizzati degli eseguibili che si compongo da un corpo principale realizzato dal programmatore e da un altro pezzo di codice che proviene da librerie esterne.
Le librerie, solitamente, vengono caricate quando è caricato il programma o quando le funzioni che sono contenute sono richieste per la prima volta. Se un altro programma in precedenza aveva caricato la libreria, non è necessario caricarla nuovamente.
Quando una libreria è richiesta questa non viene caricata l'intera libreria, ma solo le funzioni necessarie saranno portate nella RAM.
Le librerie condivise hanno preso vari nomi in Linux shared object in Windows DLL. Il compilatore quando compila le librerie utilizza degli indirizzi relativi e non assoluti.

File mappati *(il file di testo diventa un pezzo della memoria)*

Le librerie condivise in realtà rappresentano un caso speciale di una situazione più generale, chiamata **Memory-Mapped File** (**file mappati in memoria**). L'idea è che un processo possa inviare una chiamata di sistema (mMap) per mappare un file di testo all'interno del suo spazio degli indirizzi virtuali. Questo permette ai processi di accedere direttamente ad alcuni buffer del SO come se fossero buffer privati del processo. Il SO inserisce dei fleg nella tabella delle pagine e indicano che la pagina nel momento in cui sarà aggiornata, allocata, ecc.. andrà scritta all'interno del file.
Abbiamo due possibili utilizzi:

1) Accedere a un file come se fosse un array di caratteri. Infatti il contenuto del file viene mappato nello spazio di indirizzamento del processo. Questo metodo viene usato per accedere rapidamente a porzioni di un file o per ottimizzare l'accesso a un file da parte di più processi.
2) Condividere con altri processi un'area di memoria. Questa è la tecnica più efficiente per la comunicazione tra processi., ma se gli accessi da parte dei processi non sono correttamente sincronizzati, è soggetta ad errori a causa degli accessi simultanei.

Gestione dei page fault

Ecco cosa accade nel kernel quando avviene una page fault

1) Nel momento in cui la MMU si rende conto di non avere a disposizione una pagina nella TLB, genera una TRAP. La MMU invia una TRAP al kernel.
2) Metto in sicurezza lo stato della CPU, quindi salvo i registri.
3) Il SO capisce qual è la pagina di memoria virtuale che è richiesta
4) Il SO verifica i permessi di accesso

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

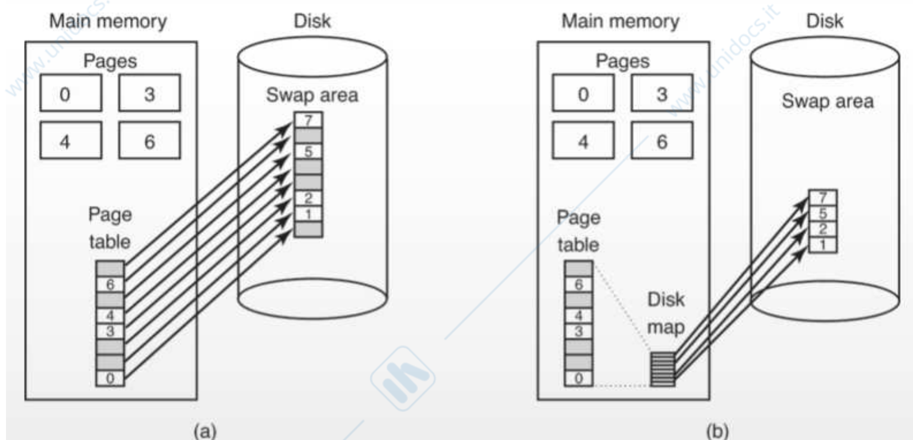
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

5) Se il bit di modifica è attivo devo aggiornare l'informazione, questo comporta che devo aggiornare la pagina sul disco. L'aggiornamento comporta delle operazioni di I/O che consistono in un cambio di contesto
6) Il SO cerca la pagina richiesta
7) Viene aggiornata la tabella delle pagine
8) Recuperata l'istruzione dal backup
9) Il processo che ha generato il page fault viene inserito nello scheduling
10) Vengono ricaricati i registri e le vari informazioni, quindi recupero lo stato della CPU
11) Continua l'esecuzione

Se il page fault avviene durante il **ciclo di fetch** (cioè durante il normale funzionamento della CPU che esegue tre operazioni principali: *preleva* un'istruzione dalla memoria principale, la *decodifica* con cui interpreta l'istruzione, infine la *esegue*) è possibile risolverlo soltanto in modalità hardware, cioè la CPU deve effettuare un **checkpoint** dell'ultima istruzione che è riuscita a prelevare completamente e quando arriva la trap di page fault, allora devo ritornare all'ultimo checkpointer e successivamente devo ricominciare la fase di fetch. Ricominciare la fase di fetch comporta la perdita di alcune operazioni che avevamo precedentemente già eseguito.

Memoria secondaria

È quella zona di disco dove vengono inserite le pagine solitamente sono le D-space. Le pagine vanno fisicamente sul disco nella stessa partizione in cui si effettua lo swap dei processi. Solo che nel caso dei processi si chiama *SWAP* e nel caso delle pagine si chiama *Demand Paging*.

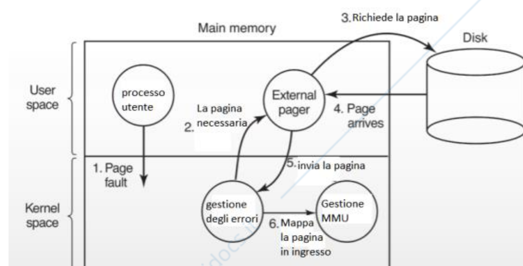


È necessario avere una mappa tra le pagine presenti in memoria e la loro posizione sul disco. Lo posso fare in due modi

1) Posso avere un'immagine del processo sul disco, quindi potrebbero esserci dei buchi che si riferiscono a quelle pagine non presenti sul disco. Una pagina in memoria principale ha sempre una copia su disco (*copia ombra*), ma potrebbe non essere aggiornata.
2) È possibile crearsi una mappa separata all'interno della RAM ed effettuare una mappatura soltanto dei blocchi che sono sul disco. Una pagina in memoria non ha una copia sul disco.

Separazione tra politica e meccanismo

Durante la progettazione è importante distinguere tra la politica e i meccanismi, questo perché non si vuole riprogettare tutto solo per aggiornare, ad esempio, un nuovo algoritmo di sostituzione delle pagine.
Una **politica** è un modo per affrontare il problema, ad esempio gli algoritmi per il rimpiazzamento delle pagine.
Meccanismi sono quelle funzioni che ci permettono di attuare le politiche, ad esempio il flag per controllare che la pagina sia presento o meno, il bit di ultima modifica, ecc…
All'interno dello spazio kernel vengono implementate delle politiche, mentre il software implementa il meccanismo sia nello spazio utente e nel kernel. Questo ci permette di sostituire le politiche senza cambiare i meccanismi.



Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

Memoria segmentata

La segmentazione consiste nel fornire alla macchina molti spazi degli indirizzi completamente indipendenti, chiamati Segmenti. I segmenti possono avere dimensioni differenti tra di loro e possono cambiare durante la loro esecuzione (allargarsi e restringersi).
Il programma deve fornire un indirizzo costituito da due parti; un **numero di segmento** e un **indirizzo all'interno del segmento**. Solitamente all'interno del segmento non sono presenti mix di tipologie di dati differenti e segmenti differenti possono avere diverse tipologie di protezione.

Symbol table: è una tabella che associa i nomi delle funzioni alla posizione dove si trova il codice rispettivo.
Source text: è il codice del programma in esecuzione
Constants: dove sono conservate le costanti, cioè informazioni che non cambiano.
Parse tree: struttura dati ad albero
Call stack: sono elencate le chiamate effettuate

L'implementazione della segmentazione differisce dalla paginazione per un motivo fondamentale, cioè le pagine sono di dimensione fissa e i segmenti no.
Il problema che si verifica con la segmentazione è la **frammentazione esterna** in quanto la rimozione e l'aggiunta di nuovi segmenti in memoria porta alla creazione di spazio libero non compattato. È possibile fare una ottimizzazione spostando un segmento alla volta partendo dai più piccoli.

Confronto tra politiche

Considerazione	Paginazione	Segmentazione
Il programmatore deve conoscere quale TECNICA è in uso?	NO, perché si vede un unico spazio di indirizzamento e la MMU che si occupa di tutto.	SI, perché per specificare un indirizzo nella memoria segmentata è necessario fornire sia l'indirizzo del segmento che l'indirizzo all'interno del segmento.
Quanti spazi di indirizzi lineari ci sono?	1 o 2, gli indirizzi di I-Space e D-Space	MOLTI
Lo spazio degli indirizzi totali può superare la dimensione della memoria fisica?	SI	SI, perché i segmenti possono essere paginati. Cioè all'interno del segmento ho la pagina. Un segmento non deve stare tutto in memoria e non deve essere contiguo
Le procedure e i dati possono essere distinti e protetti separatamente?	NO	SI
Le tabelle la cui dimensione varia possono essere disposte facilmente?	NO, perché la dimensione della pagina è fissa ed è uguale ai frame	SI, perché il segmento può aumentare e restringere la sua dimensione
La condivisione delle procedure fra utenti è facilitata?	NO, perché (librerie condivise, ecc. sono procedure fornite dal SO e sono codice protetto gestite dal kernel.) nel caso di procedure utente è necessario creare una libreria condivisa	SI, perché è possibile condividere il segmento con un altro processo. In un segmento ci sono più processi
Perché è stata inventa questa tecnica?	Per avere uno spazio degli indirizzi grande senza dover acquistare ulteriore memoria fisica.	Per consentire a programmi e dati di essere spezzati in spazi degli indirizzi logicamente indipendenti e per facilitare la condivisione e la protezione.

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

FILE SYSTEM

Per molte applicazioni è fondamentale che le informazioni non vadano perse, l'obiettivo è quello di salvare le informazioni a lungo termine e i requisiti essenziali sono:

1- Deve essere possibile salvare una grandissima quantità di informazione
2- Le informazioni devono permanere oltre la fine del processo che la usa
3- Le informazioni devono poter essere acquisite contemporaneamente da molteplici processi.

Per permettere la memorizzazione persistente vengono utilizzati i dischi. I quali li possiamo immaginare come una sequenza lineare di blocchi dalle dimensioni fisse, che supporta due operazioni: quella di lettura e di scrittura.

Il file system viene utilizzato perché a RAM ha un costo elevato ed è volatine. Per questo è necessario definire una struttura dati che sia permanente a lungo periodo. Per fare questo è necessario intervenire sia sul livello software che hardware.

- Software: come dispongo i dati su una periferica di memorizzazione stabile
- Hardware: come gestisco fisicamente la periferica.

Livello software del file system

NOMI DEI FILE

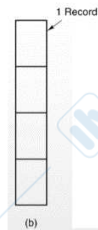
I file sono unità logiche create dai processi. I processi possono leggere file esistenti e crearne di nuovi se necessario. Le informazioni salvate nei file sono **Persistenti**.

Alla creazione del file gli viene assegnato un nome, la denominazione dei file cambia da sistema a sistema. La parte che segue il nome è detta **Estensione dei file** e indica il tipo del file. In UNIX le estensioni sono solo convenzioni e non sono forzate dal SO. I processi possono accedervi al file utilizzando il suo nome.

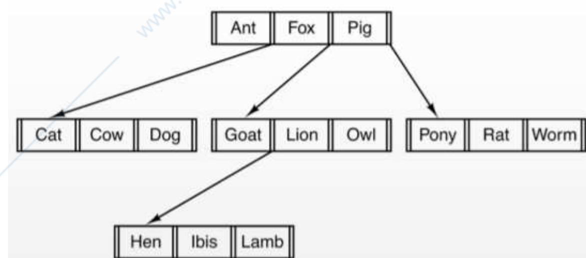
STRUTTURA INTERNA

Possiamo distingue tre tipi di strutture

1- Il file è una sequenza di **record** a lunghezza fissa, ciascuna con una certa struttura interna. L'operazione di lettura restituisce un record e l'operazione di scrittura sovrascrive o aggiorna un record. Nessun sistema attuale utilizza questo modello.



2- Il sistema operativo non conosce e non è interessato a cosa stia nel file, vede solo una **sequenza di byte**. Qualsiasi significato deve essere imposto dai programmi a livello utente. Questo modello è utilizzato.



3- Un file è composto da un **albero di record**, non necessariamente della stessa lunghezza, ognuno contenente un *Campo Chiave* in una posizione fissa del record. L'albero è ordinato sul campo chiave per permettere una ricerca rapida. Dei nuovi record possono essere aggiunti al file, con il SO decide dove posizionarli.

TIPI DI FILE

Possiamo avere diversi tipi di file

File normali: sono quei file che contengono informazioni dell'utente. Possono essere sia file ACII sia file binari. I file ASCII sono composti da linee di testo e il vantaggio è che se sono utilizzati da una gran quantità di programmi sia per l'input che per l'output. I file binari hanno una struttura conosciuta solo dai programmi preposti ad usarli.

Directory: sono tipi di file di sistema usati per mantenere la struttura del file system

File speciali a caratteri: sono file relativi all'I/O

File speciali a blocchi sono usati per modellare i dischi

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

Un file ha 5 sezioni:
1) Intestazione: inizia con un magic numer che lo identifica come eseguibile.
2) Testo
3) Dati
4) Bit rilocazione
5) Tabella simboli

Accesso ai file

Possiamo distinguere l'accesso ai file in:
Accesso Sequenziale: il sistema può leggere tutti i byte o i record in ordine, ma non può saltarli o leggerli in maniera disordinata
Accesso Casuale: è possibile leggere i byte o i record di un file in maniera disordinata o accedere ai record secondo una chiave. Per specificare da dove iniziare a leggere possono essere usati due metodi:
a) Ogni operazione di READ fornisce la posizione dove iniziare a leggere nel file
b) È fornita un'operazione speciale, SEEK per impostare la posizione attuale per passare alla lettura. Questo è il metodo più utilizzato.

Attributi dei file

Ad ogni file il So associa delle informazioni, come la data, il nome, la dimensione, ecc... Tutte queste informazioni vengono chiamati Attributi o Metadati.
Alcuni attributi vengono denominati FLAG, questi sono bit che controllano o abilitano alcune proprietà specifiche.

Operazioni sui file

Le operazioni più comuni sono:
Create: creazione del file
Delete: eliminazione de file
Open: apertura di un file, questo serve per portare nella memoria principale gli attributi e l'elenco degli indirizzi del disco.
Close: pulizia della memoria con la chiusura del file
Read: lettura dei dati dal file
Write: i dati sono scritti sul file
Append: è una forma limitata di write, essa infatti permette di scrivere i dati solo alla fine del file.
Seek: indica da quale posizione iniziare a prelevare i dati.
Get attributes: per leggere gli attributi dal file
Set attributes: alcuni attributi è possibile impostarli dall'utente.
Rename: utente cambia il nome di un file esistente.

Le directory

La directory è un file che tiene traccia dei file presenti nel sistema.

Sistemi di directory a livello singolo

La forma più semplice di directory è quella di avere una sola directory contenenti tutti i file, chiamata Directory Principale. Il vantaggio di questo schema è la semplicità e la capacità di localizzare i file rapidamente.

Sistemi di directory gerarchici

La directory ramificate ad albero ha la caratteristica principale che ogni utente può avere una directory principale privata per la propria gerarchia.

Path name

Per specificare un nome del file in un sistema ad albero, vengono utilizzati due metodi:
1) A ogni file viene assegnato un **Path Name Assoluto**, composto dal percorso della directory principale al file. Questi sono univoci
2) Il **Path Name Relativo** parte dalla cartella di lavoro corrente. Descrive la posizione di un elemento a partire da un'altra posizione dell'albero del file system.

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

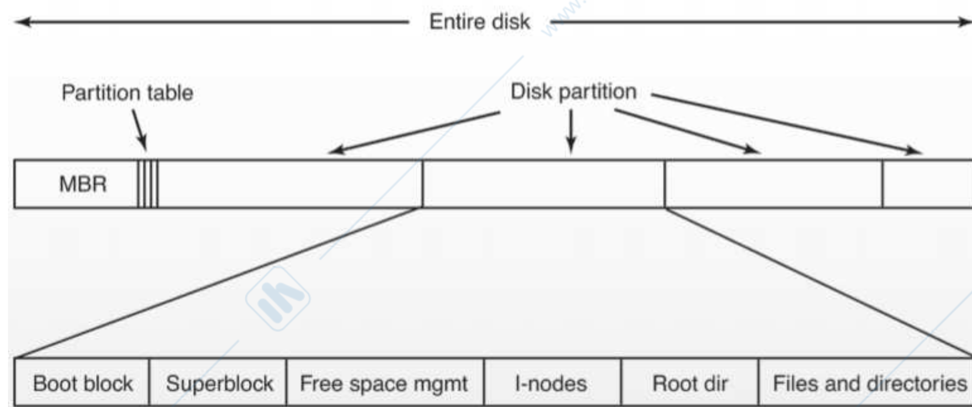
OPERAZIONI POSSIBILI

Create: viene creata una directory vuota
Delete: viene cancellata una directory, questo può avvenire solo se la directory è vuota
OpenDir: viene aperta una directory e successivamente è possibile leggera record per record
CloseDir: viene chiusa la directory e si libera lo spazio della tabella interne.
ReadDir: restituisce la voce successiva all'interno di una directory.
Rename: rinomina una directory.

Write
- **Link**: permette ad un file di apparire in più di una directory. Questo aumenta il contatore nell'i-node del file (per tener traccia del numero di directory contenete il file) è detto Hard Link.
- **Unlink**: viene rimossa una voce dalla directory. Se il file è presente in una sola directory, viene rimosso dal file system. Se è presente in molteplici directory viene rimosso solo il path name specificato gli altri rimangono.

Get attribute
Set attribute

Una directory ha degli attributi come i file che possono essere i diritti di accesso (anche più elaborati dei file) chi l'ha creata, quando è stata creata, l'ultima modifica, ecc…



La maggior parte dei dischi è suddivisa in più partizioni, ed ognuna ha il suo file system. Il settore 0 del disco è chiamato **MBR** (master boat record) che è usato per avviare il computer. Alla fine del blocco MBR è presente una tabella delle partizioni. Quando il computer è avviato il BIOS legge ed esegue l'MBR e la prima cosa che fa è leggere la partizione attiva ed eseguire il primo blocco, chiamato blocco di boot.
Il programma nel blocco di boot carica il SO contenuto in quella partizione. Ogni partizione inizia con il blocco di boot anche se non contiene un sistema operativo avviabile. Il layout di una partizione del disco cambia molto da file system a file system. Spesso il file system contiene i seguenti elementi:

Super blocco: indica la strutturazione del file system. Contiene tutti i parametri chiave riguardati il file system ed è letto in memoria all'avvio del computer o quando il file system viene usato per la prima volta
Gestione dello spazio libero: contiene le informazioni sui blocchi liberi nel file system ad esempio sotto forma di bitmap.
I-node: un array di strutture dati, una per file che raccolgono tutto riguardo il file
Directory principale: contiene la cima dell'albero del file system.
File e directory: contiene tutte le altre directory e file.

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

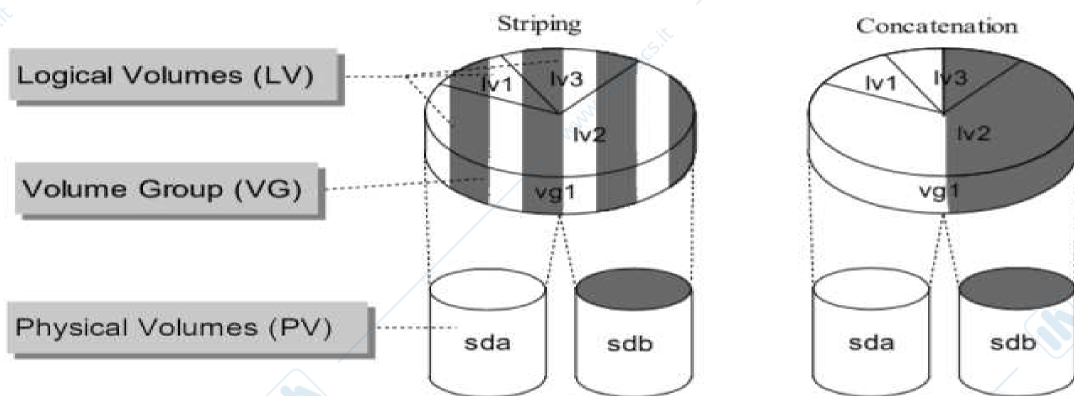
Logical Volume Manager (LVM)

Il LVM è un'astrazione hardware del disco fisso. Permette al SO di utilizzare dei dispositivi logici virtuali per l'accesso al disco che mascherano la natura dei dispositivi fisici su cui risiedono i dati. Questo permette di cambiare la configurazione dello storage senza dover intervenire su tutte le applicazioni che usano lo storage.

Come funziona

- Senza LVM
  Sono presenti dei dischi fisici con le partizioni. In queste partizioni ci sono i file system

- Con LVM
  Ogni disco è diviso in unità discrete e queste fanno a capo a un Physical Volumes (PV) il quale viene suddiviso in blocchi di dati, chiamati Physical Extent (PE). Questi sono associati con i blocchi di dati con cui è diviso un Logical Volumes (LV) chiamati Logical Extent (LE).
  I LE vengono mappati alternamente su PE di diversi Physical Volumes (PV) in questo modo le performance potrebbero migliorare. Questo metodo è chiamato **STRIPED.**
  Un altro metodo è chiamato **LINEAR**, dove i LE di un LV vengono mappati sequenzialmente ai PE di uno o più PV.



Il Physical Volumes può essere o l'intero disco oppure una partizione del disco. L'insieme di questi formano il Volume Groups (VG), il quale utilizza questo insieme di PV come un unico volume di archiviazione, ma questo per essere utilizzato viene suddiviso in diversi Logical Volume (LV). Possiamo pensarla che i VG siano gli hard-disk, mentre i LV sono come le partizioni e ogni LV avrà il proprio file system.

In questo modo possiamo modificare il VG e i LV mentre il sistema è attivo. Possiamo aumentare il numero di LV oppure aumentare la stessa dimensione dei LV già presenti.
È possibile creare un sistema di backup senza l'utilizzo di software aggiuntivo.
Questo sistema di backup consiste in due batterie di dischi e agganciamo la seconda batteria alla prima. Il LV si mette nella modalità **snapshot**. Verrà effettua una fotografia al file system in quel momento e eseguirà una copia byte a byte della LV sulla nuova batteria di dischi. Intanto gli utenti lavorano e anche gli applicativi continueranno a funzionare. Tutte le modifiche fatte al file system vengono inserite in uno **stato di virtualizzazione a tampone**, per cui gli applicativi vedono le modifiche ma non viene effettuato il backup. Quando il backup finisce il LVM esce dalla modalità snapshot e vengono applicate tutte le modifiche avvenute nel lasso di tempo. Il vantaggio è quello di applicare la batteria di dischi che ha effettuato il backup come nuovi dischi e questi saranno subito pronti ad iniziare dall'ultimo backup fatto senza nessuno sforzo.

LVM e RAID

Prima della creazione del LVM, la soluzione migliore era quella di comprare un controller chiamato RAID che lavorava a livello hardware (mentre il LVM opera a livello software). Il controller RAID è un controller in cui si accagliavano due dischi e a livello hardware risultava un solo disco, ma il compito era del controller di inserire i dati all'interno dei due dischi. Questo comportava delle limitazioni, come:

a) i dischi devono essere identici, mentre nel LVM no
b) è possibile fare lo Striped, ma con prestazioni inferiori al LVM

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

il RAID 5 è costituito da quattro dischi in cui un disco contiene un'informazione detta di parità, cioè un'informazione che fa da controllo delle prime tre. Usiamo i dischi in sequenza e questo comporta una velocità maggiore, ma se uno dei dischi si rompe comunque gli altri tre contengono gli stessi dati.

Implementazione dei file

Per tener traccia di quale blocco del disco è associato ad un determinato file è possibile utilizzare due metodi, che sono allocazione contigua oppure allocazione a liste collegate.

File con allocazione continua

I file vengono memorizzati in una sequenza contigua di blocchi del disco. Questo potrebbe causare della *frammentazione interna* in quanto non è certo che l'ultimo blocco venga utilizzato del tutto.

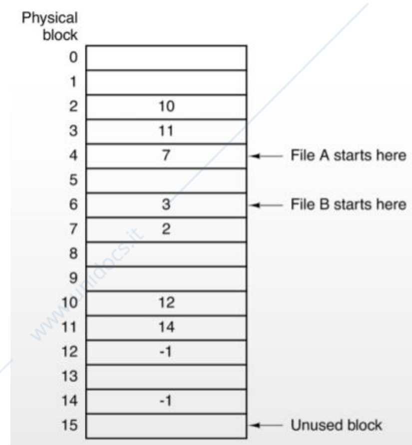
- *I vantaggi* di questa tecnica sono la facilità di implementazione in quanto è necessario ricordare l'indirizzo del disco del primo bocco e il numero dei blocchi occupati dal file. L'accesso casuale del file è nettamente più veloce. I dati vengono letti alla massima velocità del disco.
- *Gli svantaggi* nell'utilizzare l'allocazione continua dei file sono che nel corso degli anni i dischi possono soffrire di *frammentazione esterna*.

L'allocazione contigua è realmente usata nei CD-ROM in quanto la dimensione dei file è conosciuta in anticipo e non cambierà nel tempo.

File con allocazione a liste collegate

Questo metodo consiste nel mantenere ciascun file attraverso una lista collegata di blocchi del disco. All'interno di ciascun blocco è presente un puntatore che punta al blocco successivo.
I vantaggi di questa tecnica sono nella riduzione della frammentazione esterna
Gli svantaggi delle liste collegate è che l'accesso casuale è estremamente lento in quanto è necessario leggere gli n-1 blocchi prima di arrivare a blocco n desiderato. Inoltre, la quantità di spazio per i dati di un blocco non è più una potenza di due, poiché il puntatore occupa alcuni byte. Dato che alcuni programmi leggono e scrivono in blocchi la cui dimensione è una potenza di due, questo comporta che quando è richiesta la lettura del blocco è necessario concatenare le informazioni da due blocchi del disco eliminando i byte occupati dai puntatori per costruire così un blocco la cui dimensione è una potenza di due e quindi compatibile con i programmi. Questo genera ulteriore sovraccarico, in quanto per ogni lettura di blocco sarà necessario fare due accessi.

Si è pensato allora di inserire tutti i metadati che erano all'interno dei data-block in una tabella apposita.



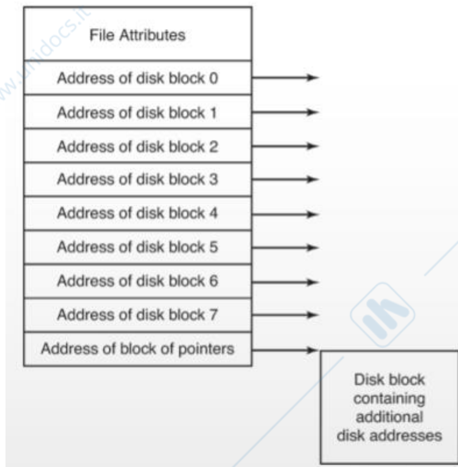
Questa tabella viene inserita nella prima traccia del disco. In questo modo è nato il **File allocation table**, **FAT**. La FAT è costituita da una tabella con un elemento per ogni blocco del disco ed è indicizzata dal numero di blocco. L'elemento di directory contiene il numero del primo blocco del file, che a sua volta contiene il numero del blocco successivo del file. L'ultimo blocco ha come elemento della tabella un valore di fine file. I blocchi inutilizzati sono contrassegnati da un valore di default.
Questa tabella è necessaria che venga aggiornata spesso e che sia presente nella memoria (RAM), questo comporta che la FAT non si presta molto bene a dischi di grandi dimensioni. È possibile ottenere una FAT 16 o da 32, la cifra 16 si intende il numero di bit con cui si costruisce la tabella. Questo comporta che il numero di data_block all'interno del file system non può essere superiore a 2^{16} perché non è possibile indicizzare. Per questo motivo quando si utilizza il FAT, file più grandi di 4GB non è possibile memorizzarli.

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

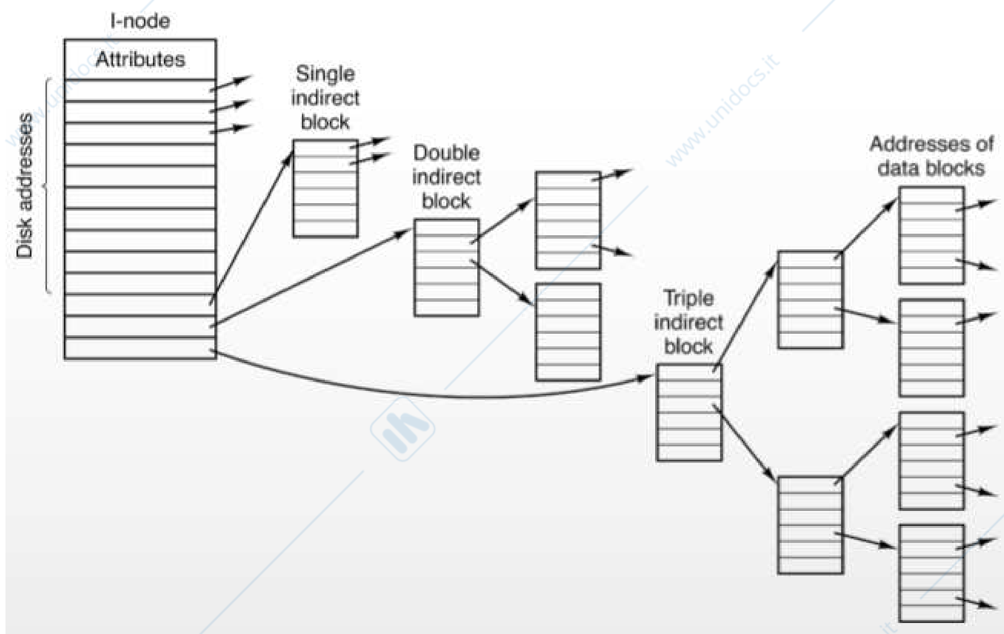
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

i-node

Oggi per tener traccia quali data block appartengono ad un file viene utilizzata un'allocazione ad albero. Essa è una strategia adottata nel mondo UNIX, ed è costituita principalmente da una struttura dati chiamati i-node. L'**i-node** elenca quasi tutti gli attributi (no il nome del file che si trova nella directory) relativI al file e anche gli indirizzi dei **data block** del file relativo, che vengono usati per memorizzare i dati. Alla fine dell'i-node è presente un data block che ha come riferimento un data block che contiene riferimenti ad altri indirizzi del disco, in questo modo si cerca di estendere la struttura. Come viene implementato quest'ultimo blocco dell'i-node dipende dal file system. In generale viene utilizzato come base l'Unix File System L'i-node si trova in memoria solo quando il file corrispondente è aperto.



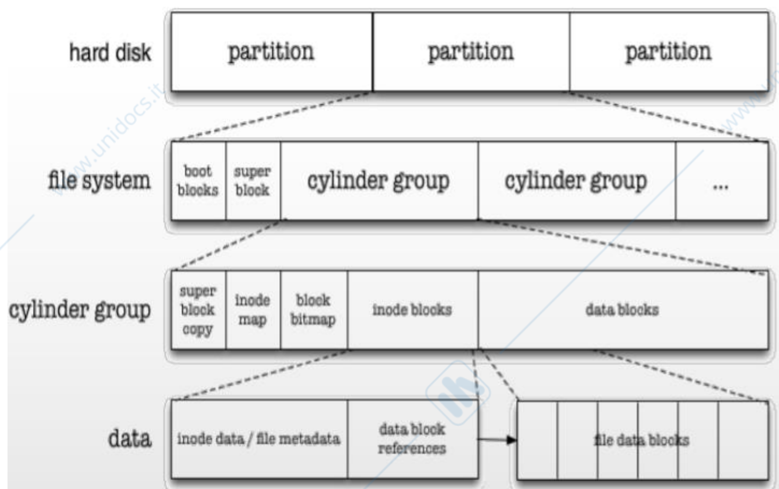
Unix File System



L'i-node ha un'intestazione con i metadati relativi al file. Secondo lo standar UFS, i primi 10 campi sono rifermenti a data block che contengono dati, seguono tre campi finali con riferimenti a data block che non contengono dati. Il primo viene chiamato data block a singola direzione (single indirect block), dove ci sono riferimenti a data block che contengono dati (del file allocato).
Quando viene allocato anche l'ultimo riferimento del data block a singola direzione, allora viene usato il penultimo puntatore dell'i-node il quale si chiama blocco a doppia direzione (double indirect block). Il quale alloca un data block che fanno riferimento ad altri data block che a loro volta hanno riferimenti a dati del file.
L'ultimo riferimento dell'i-node è a tripla direzione.
Non è possibile usare un i-node molto grande dove inserire tutti i dati senza fare delle separazioni in quanto andremmo in contro ad una frammentazione interna molto significativa.
Solitamente sul disco sono presenti un numero maggiore di file di dimensione minima rispetto a file con una dimensione molto grande. Per questo motivo si preferisce allocare i file di piccola dimensione nei primi 10 riferimenti dell'i-node in quanto si prevede che verranno utilizzati più spesso, mentre i file di grande dimensione vengono allocati in tripla direzione e questo comporta una maggiore lentezza, ma giustificata in quanto non ci si aspetta delle buone prestazioni.

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari



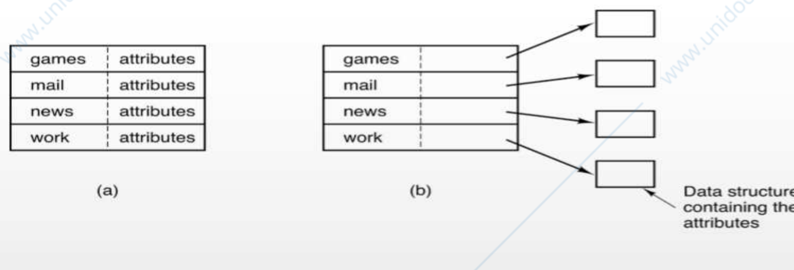
Il disco è suddiviso in partizioni ed ogni partizione ha il suo file system…..

GUARDARE PARAGRAFO LAYOUT DEL FILE SYSTEM

Implementazione di directory

La directory è un file e può essere organizzata in due modi differenti, che sono:

a) Ogni voce contiene il nome e gli eventuali attributi del file o un i-node.
b) Ogni voce contiene il nome e un puntatore ad una struttura separata che contiene gli attributi del file o un i-node.

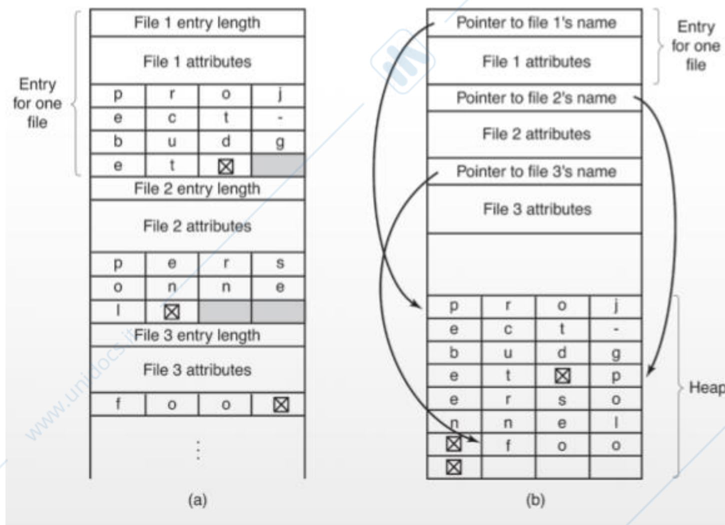


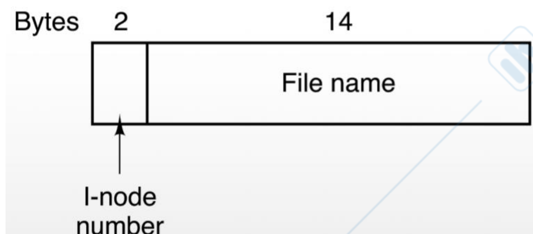
Il nome del file può essere lungo a piacimento. Abbiamo due possibilità per gestire il nome del file.

a) ciascun elemento della directory inizia con degli attributi tra cui la lunghezza del nome del file. In coda agli attributi viene inserito il nome del file, il quale termina con un carattere speciale. Il problema è che non si ha molto spazio per gestire il cambio di nome.

b) Ciascuna voce della directory hanno una lunghezza fissa, all'interno del record si ha un riferimento ad una zona che si trova alla fine dello spazio della directory chiamato HEAP dove sono presenti i nomi dei file. Il vantaggio è che è possibile rinominare il nome del file aggiungendo il nuovo nome in coda nell'HEAP e cambiando il riferimento presente nel record.

In **Unix File System**, gli attributi del file si trovano nell'i-node. Per cui nella directory è presente il nome del file e il numero di i-node a cui si riferisce.

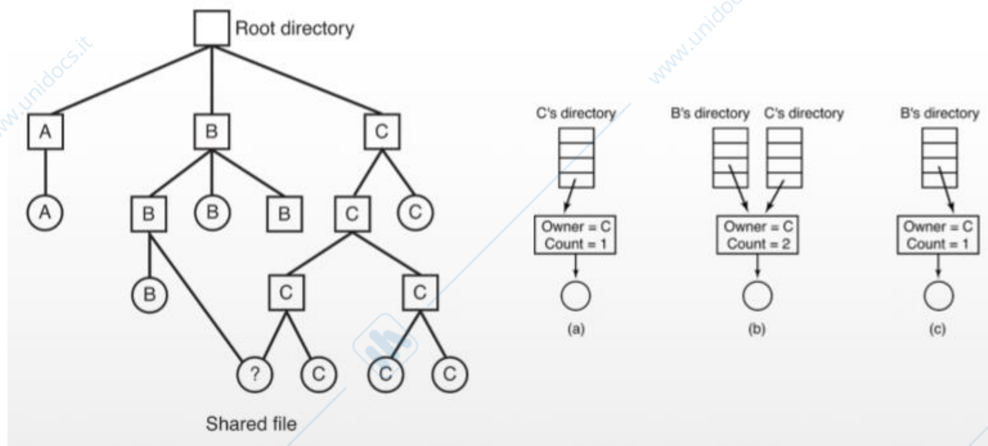




Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

File condivisi tra directory



Non è consigliabile inserire il nome del file all'interno dei metadati dell'i-node in quanto questo può essere raggiungibile da più directory con nomi differenti. È necessario introdurre il concetto di **LINK**, ovvero la connessione che si verifica tra la directory e il file.
Possiamo implementare i file condivisi in due modi:

1- Considerando due diverse directory che puntano allo stesso i-node. Il problema si verifica quando si elimina il proprietario di un file condiviso. In quanto il sistema vede dal conteggio dei LINK COUNT che il file è ancora in uso da altre directory, quindi elimina la directory proprietaria e lascia invariato l'i-node decrementando il conteggio. (ma B non è il proprietario)
2- Viene creato un file di tipo LINK e viene inserito nella directory non proprietaria. Questo file contiene solo il path name del file a cui si collega. Solo il proprietario ha il puntatore all'i-node. Quando un proprietario rimuove il file, esso è distrutto, mentre la rimozione di un link non influisce sul file stesso. Questa tecnica sovraccarica il lavoro in quanto è necessario leggere, analizzare ed eseguire il file contenente il path name. Inoltre serve un i-node extra per ciascun link simbolico, ma è possibile ottimizzare memorizzando il path name sull'i-node stesso.

In **UFS** i file condivisi vengono implementati in modo tale che nelle due directory lo stesso i-node è implementato con nomi diversi ed è presente un parametro che segna il numero di volte che è stato referenziato, quando questo è uguale a 0 allora viene eliminato.

Varianti possibili del File System

File System basati su log strutturati

il File System chiamato **LFS** consiste nel conservare un determinato numero di dati nel buffer cache, riferendoci al principio di località in cui se si usa un file allora si utilizzeranno i data block vicini. Se la struttura dati non viene aggiornata frequentemente allora al primo problema (interruzione della corrente elettrica, problemi del kernel, inconsistenze, ecc..) tutte le modifiche presenti nel buffer verranno perse. Mentre se l'immagine che è presente nel buffer cache viene aggiornata troppo frequentemente allora si ha un calo di prestazioni.
Si è pensato di aggiungere dei periodi chiamati SINC (è una chiamata di sistema sinc) che obbliga l'aggiornamento del buffer cache sul disco. Questo comporta un rischio nel caso in cui si verifica un problema e il periodo di aggiornamento non è stato eseguito. È presente una zona libera in cui vengono scritte tutte le operazioni fatte, chiamato **LOG**. Nel caso in cui si verifica un problema e non si è eseguita la SINC, allora si verifica la consistenza del file system aiutandomi con il log, cioè si confrontano le operazioni presenti nel log con quelle eseguite e se nel caso in cui non sono stata svolte allora si eseguono. Portando così il file system all'ultimo stato presente nel log.

Altri file system basati sul log sono

Journaled File System

Esso raccoglie tutte le operazioni che devono essere eseguite sul file system in un dato istante. Nel momento in cui il SO esegue la SINC, dato che l'immagine in memoria è uguale a quella sul disco, viene marcato come checkpoint. Quando si verifica un problema, il SO guarda nell'area riservata al Journaled e controlla l'ultimo checkpoint. Tutte le operazioni presenti nel checkpoint vengono eseguite, queste operazioni devono essere idempotenti. Le **operazioni idempotenti** sono quelle operazioni che applicate più di una volta, mantengono inalterato il risultato. Ad esempio aggiungere un carattere alla fine del file

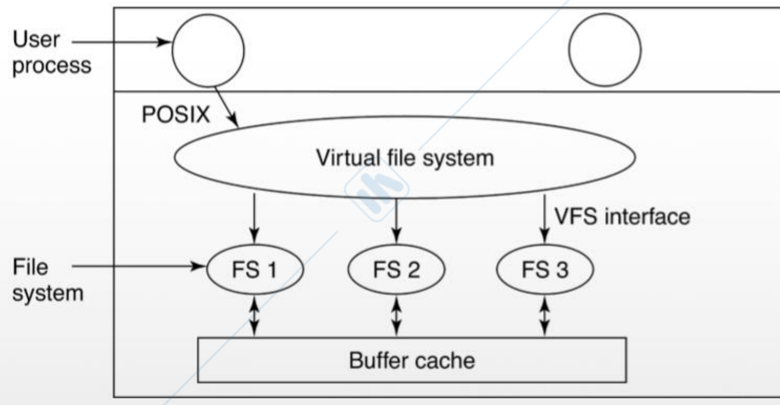
Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

non è un'operazione idempotente, mentre assegnare una dimensione fissa ad un file e scrivere in una certa posizione un carattere, è un'operazione idempotente

Virtual File System

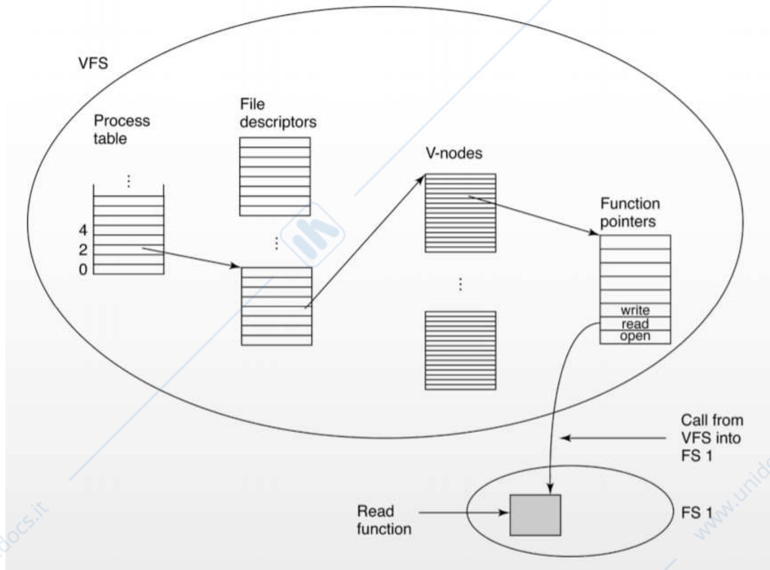
Molte volte, anche sullo stesso PC, sono in uso molti file system diversi, anche per lo stesso SO.
In Windows i file system vengono identificati con una lettera differente di unità.



In Unix si cerca di unire molti file system in una struttura singola. Infatti viene utilizzato il concetto di VFS per cercare di integrare più file system in una struttura uniforme. Tutte le funzionalità comuni ai file system vengono inseriti in un file system virtuale il quale richiama i file system reali. Questo permette ad un processo di vedere un'interfaccia uniforme verso file system diversi.

Serve a dare continuità e trasparenza, si pone come file system sopra l'implementazione reale, per cui rimappa delle funzionalità sue su delle funzionalità reali del file system. Sopra il file system reale è presente un file system virtuale che ha delle funzionalità che vengono rimappati su file system specifici

Saranno presenti dei nodi virtuali, che non sono gli i-node, che rimappano le system call su delle system call adatte al file system.

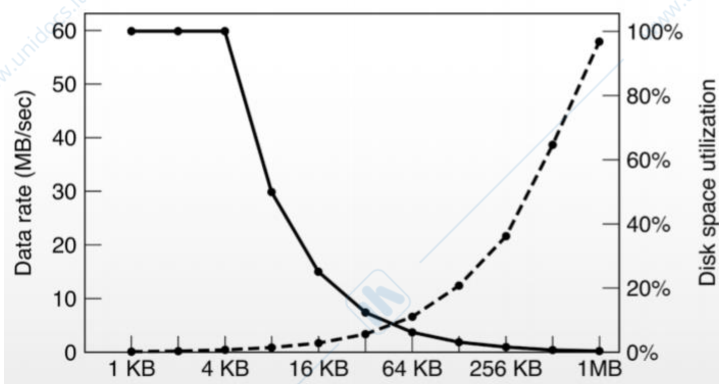


Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

Dimensione dei blocchi di dati

È necessario definire una dimensione dei blocchi di dati (data block), se questa è troppo grande ci si impiega poco tempo a leggerlo, ma sarà presente una forte frammentazioni interna.

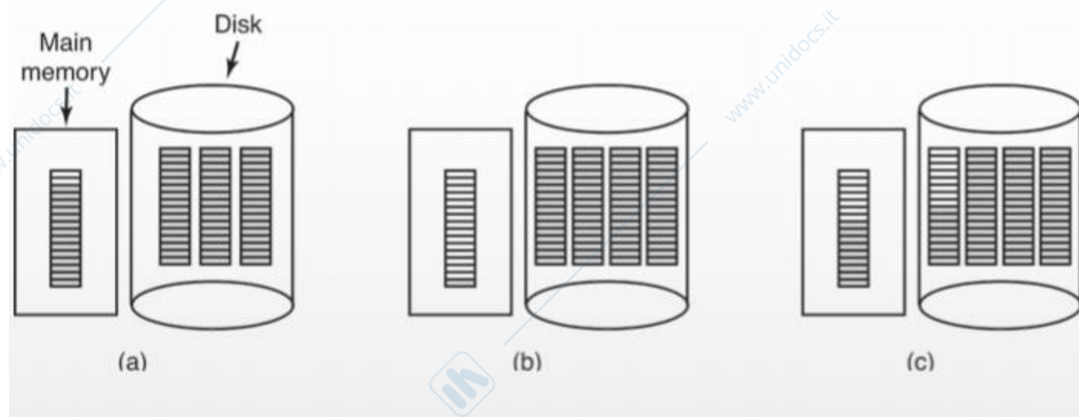


Nel caso in cui il data block è piccolo, allora la frammentazioni interna sarà al minimo, il problema è che se devo leggere un file molto grande sarà necessario fare molti accessi al disco.
È possibile fare delle supposizioni sul rendimento se si hanno dei data block con dimensione crescente, per cui sull'asse delle x abbiamo la dimensione dei data black che varia da 1KB a 1MB. Si è visto che la frammentazione interna cresce in maniera esponenziale (curva tratteggiata) all'aumentare del data block.
Facendo la simulazione sul throughput di sistema, cioè il quantitativo di byte che si riesce a trasferire in una unità di tempo (per cui se è molto efficiente si avrà un throughput molto alto) si ha una curva inversa a quella precedente (curva spezzata continua)
Con data block molto piccoli si ha un throughput molto alto, mentre al crescere dei data block si ha un throughput molto basso.
La possibile soluzione su questo sistema, in queste condizioni è identificare un punto ottimale in cui mi favorisca troppo il throughput e non mi penalizza la frammentazione, in questo caso è tra i 32KB e 64KB. La dimensione del data block deve essere sempre una potenza di due

Gestione dei blocchi liberi

È necessario tener traccia dei data block liberi, è possibile utilizzare due metodi. All'interno di ogni partizione è presente uno spazio dedicato alla gestione dello spazio libero.

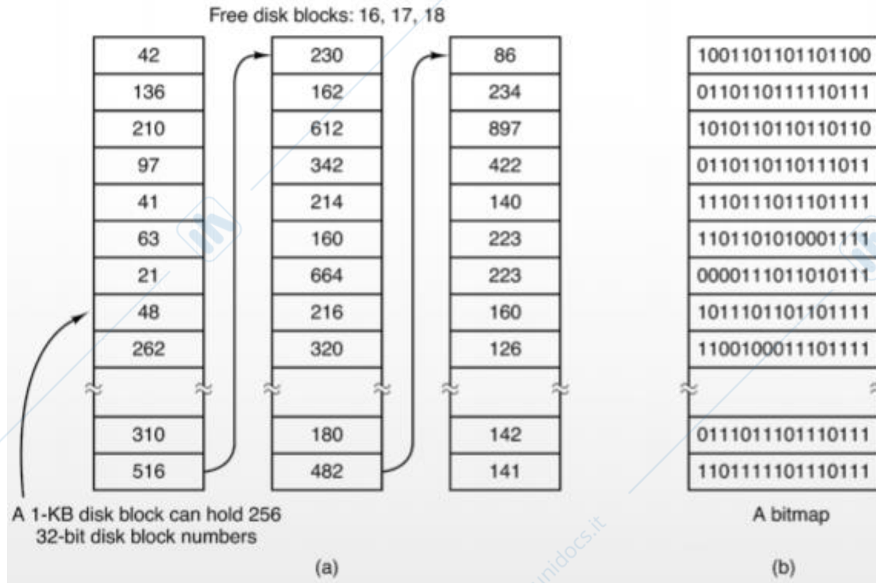
a. È possibile utilizzare una **free list** (lista collegata) che è un elenco dei numeri di data block che sono disponibili. Questa lista collegata ha il vantaggio che alla creazione di un file vengono prelevati i blocchi necessari. A differenza della bitmap che è presente sul disco, la lista si trova in memoria. È possibile inserire solo una parte della lista in memoria, il restante in una zona di swap. Quando i blocchi vengono liberati essi vanno aggiunti alla lista. Il problema che si potrebbe verificare è quando una lista è quasi vuota si potrebbe avere un aumento di I/O su disco, perché potrebbe servire immediatamente la lista presente sul disco e no quella in memoria. Una possibile soluzione è quella di leggere in memoria solo mezza lista in questo modo si ha piena la maggior parte della lista per cercare di ridurre l'I/O sul disco.



Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

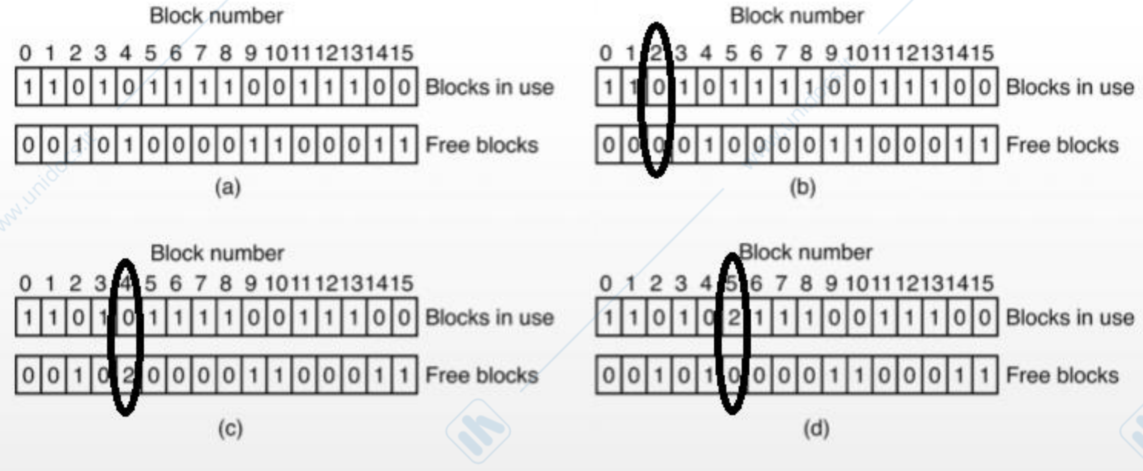
b. È possibile utilizzare una **bitmap**, che a differenza della bitmap utilizzata per gestire la memoria (capitolo 3) è accettabile perché per la bitmap in memoria era necessario trovare una sequenza di bit consecutivi a zero mentre in questo caso si hanno n bit per n blocchi. I blocchi liberi sono indicati dal valore 1, mentre con 0 i blocchi occupati nella mappa. Se i blocchi liberi sono consecutivi, il sistema può tener traccia di sequenze di blocchi piuttosto che di singoli blocchi.



Analisi della consistenza del file system

I file system possono leggere, modificare e riscrivere i blocchi, ma se si verifica un crash prima di aver riscritto i blocchi modificati, si può verificare uno stato inconsistente del sistema. Viene utilizzata un utility, in UNIX si chiama fsck (file system check) mentre in Windows scandisk. Essa viene eseguita a ogni avvio del sistema, specialmente dopo un crash. È possibile eseguire due tipi di controlli di consistenza: quello dei blocchi e quello dei file.

- Quello dei blocchi consiste in due tabelle, ciascuna contenente un contatore per ogni blocco. I contatori della prima tabella tengono traccia di quante volte ogni blocco è presente in un file (per fare ciò sarà necessario leggere tutti gli inode dal quale sarà possibile costruire una lista di tutti i numeri di blocco utilizzati nel file corrispondete), mentre i contatori della seconda tabella registrano quanto spesso ogni blocco sia presente nella lista dei bocchi liberi (per fare ciò sarà necessario analizzare la lista collegata dei blocchi liberi o della bitmap per trovare tutti quelli non in uso).



la differenza tra i due array è che il primo è un array di interi, mentre il secondo è un array di bit.

a) In condizione di stabilità mi trovo in questa situazione, il blocco o è libero oppure è occupato una sola volta. Quindi avrò un 1 nella prima o nel secondo array
b) È possibile che il blocco non appare in nessun array (blocco 2) e sarà riportato come blocco mancante. Essi sprecano spazio e riducono la capacità del disco. per risolvere il problema vengono aggiunti alla lista dei blocchi liberi.
c) È possibile che un blocco appaia due volte nella lista dei blocchi liberi (con una bitmap questo non può accadere). La soluzione è ricostruire la lista.

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

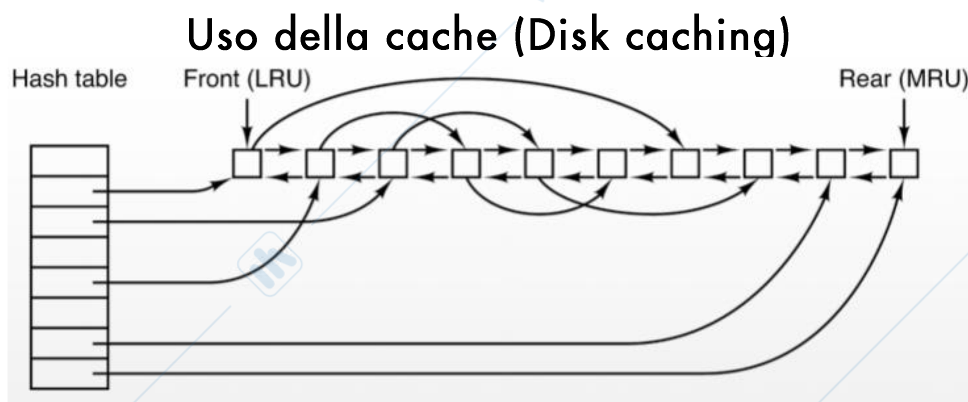
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

d) Potrebbe accadere che lo stesso blocco di dati sia presente in due o più file. Possiamo avere due scenari:
   - Se uno di questi file è cancellato si avrebbe una situazione in cui il blocco è contemporaneamente impiegato in un file (primo array) e presente nella lista dei blocchi liberi (secondo array)
   - Se fossero rimossi entrambi i file, il blocco sarebbe messo due volte nella lista dei blocchi liberi. (secondo array)
     La possibile soluzione sarebbe quella di duplicare il file referenziato due volte e spostarlo. In questo modo la struttura del file system è resa consistente, ma andrebbe reso noto all'utente il tipo di errore, per verificare il danno.

- Oltre al controllo per verificare che i data block siano conteggiati correttamente, vengono verificate anche le directory. Anche in questo caso si utilizza una tabella con dei contatori per file no per blocco. Vengono ispezionate tutte le directory partendo dalla directory principale. Si ottiene una lista indicizzata per numero di i-node, che indica in quante directory un file è contenuto. Successivamente questa lista viene confrontata con i contatori dei collegamenti memorizzati negli i-node stessi. Possono verificarsi due tipi di errori:
  a) Il contatore dei LINK è troppo alto del numero delle voci delle directory. Questo errore spreca spazio su disco con dei file che non si trovano in alcuna directory.
  b) Se due voci di directory sono collegate a un file, ma l'i-node indica che ve n'è solo una, quando una delle voci è cancellata, il conteggio dell'i-node va a zero. Questa azione comporterà che una delle directory punterà a un i-node non utilizzato, i cui blocchi potrebbero presto essere assegnati ad altri file.

  Quando un file è cancellato, tutto ciò che accade è che viene impostato un bit nella directory o nell'i-node a contrassegnare il file come rimosso. Non viene restituito alcun blocco alla lista dei blocchi liberi, finché non siano effettivamente necessari.

Uso della cache (Disk caching)



Un data block del disco viene portato in memoria e il file system legge e scrive nella zona in cui è presente il data block, successivamente si esegue la sinc che aggiorna la copia sul disco.
Questo permette di aumentare le prestazioni del sistema. Il problema si verifica quando si ha un file system molto grande e di conseguenza anche la cache diventa grande. La tecnica migliore per ridurre i tempi di accesso al disco è il BUFFER CACHE. In questo caso per cache intendiamo una raccolta di blocchi appartenente logicamente al disco ma che sono tenuti in memoria per ragioni di prestazioni.
Per questo è necessario trovare dei meccanismi per determinare rapidamente se un data block è presente nella cache oppure no. Due sono le operazioni da fare.
La prima è quella di generare una **tabella hash**, cioè una funzione che in base al numero di data bock mi divide tutti i data block in famiglie. Quando si ha bisogno di un data block basta identificare a quale famiglia appartiene e così ci si ricava una lista più breve di data block dove andare a cercare.
Quando un bocco deve essere caricato in una cache piena, ne deve essere rimosso qualcuno. Per questo motivo ogni blocco è presente in due liste, una lista che parte dalla hash table e un'altra lista che tiene i data block in ordine di ultimo utilizzo. In fondo alla coda c'è il blocco utilizzato meno recentemente e quando sarà necessario la sua immagine verrà aggiornata sul disco e successivamente eliminato per far spazio.

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

Sistema di Input/Output

Un SO ha anche il compito di controllare tutti i dispositivi di I/O del computer, inviando comandi ai dispositivi, intercettando gli interrupt e gestendo gli errori.

Dispositivi I/O

I dispositivi di I/O possiamo distinguerli in due categorie.

Dispositivi a blocchi

È un dispositivo che archivia le informazioni in blocchi di dimensioni fisse (cioè non è possibile inviare un carattere per volta, ma un blocco di caratteri), ognuno con il proprio indirizzo. Ciascun blocco può essere letto o/e scritto indipendentemente da tutti gli altri. Esempi hard disk, USB, CD.

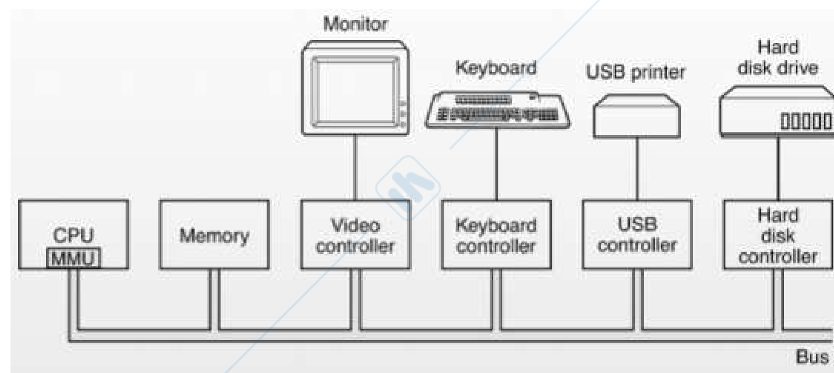
Dispositivi a caratteri

Questi dispositivi utilizzano un'unità a caratteri che rilascia o accetta un flusso di caratteri. Esempi stampanti, mouse. (leggo o scrivo un byte alla volta)

Altri dispositivi

I clock, non sono né dispositivi a blocchi e manco a caratteri, tutto ciò che fanno è produrre degli interrupt a intervalli definiti. Esempi touch screen.

È presente un bus di comunicazione, su cui si affacciano tutti i dispositivi hardware. I dispositivi di I/O sono costituiti da una parte meccanica e una elettronica. La componente elettronica è detta **Controller del dispositivo**. Il controller è un hardware che effettua una conversione delle informazioni in modo tale che siano compatibili con il dispositivo. In più il controller esercita un controllo sulla periferica, infatti la CPU interagisce con il controller per istruire la periferica su cosa deve fare. Nel momento in cui la CPU chiede ad un dispositivo di fare operazioni di lettura o scrittura dei dati, la CPU fa altro mentre il dispositivo lavora. Quando l'operazione di I/O termina allora arriva un interrupt che notifica l'arrivo dei dati richiesti. Come faccio a trasferire i dati dalla periferica alla memoria? Infatti i dati della periferica saranno presenti nel buffer di questa e dovranno esser trasportati in memoria (RAM).



I/O Mappato in memoria

Ogni controller ha a disposizione dei registri oppure un buffer dati (dipende dal dispositivo) con cui comunicare con la CPU. La CPU comunica con il controller in due possibili modi:

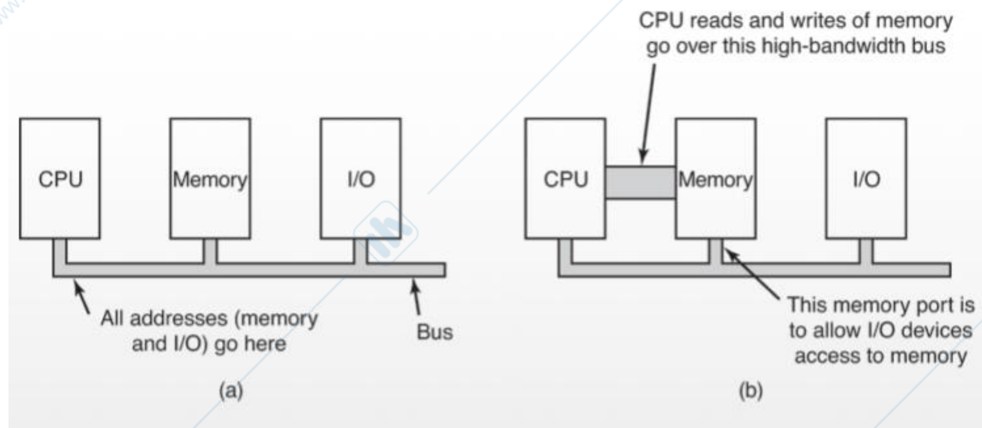
a) A ciascun registro è assegnato un numero di porta. L'insieme di tutte le porte viene chiamato spazio delle porte di I/O. Essi non sono mappati nello spazio degli indirizzi della memoria. La CPU parla con il controller e gli dice cosa deve fare. Questo metodo comporta che la CPU supporti delle operazioni specifiche per fare l'I/O. L'istruzione che esegue l'I/O con la porta è **int**, non significa interrupt (il prof si incazza). **L'I/O con porta** non funziona con i dispositivi a blocchi perché se ho un'istruzione a basso livello bisogna coinvolgere anche la MMU e per questo viene scelto il metodo successivo.
b) **I/O mappato in memoria**, vengono mappati tutti i registri di controllo nello spazio della memoria con un indirizzo univoco. Viene utilizzata la MMU ed essa è a conoscenza di quella zone di memoria che non fa riferimento a delle pagine. Quando si richiede di leggere o scrivere in quelle zone di memoria, in realtà la MMU farà riferimento al flag del controller. Solitamente vengono assegnati gli indirizzi superiori della memoria.
c) È presente uno schema ibrido, con buffer dei dati dei dispositivi di I/O mappati in memoria e porta di I/O separati per i registri di controllo.

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

Quando la CPU vuole leggere dalla memoria o dalla porta di I/O, inserisce l'indirizzo di cui ha bisogno nelle linee degli indirizzi del bus e dichiara un segnale di READ sulla linea di controllo del bus.

Per migliorare le prestazioni viene aggiunto un bus dedicato tra la memoria e la CPU, in quanto tutte le richieste che invio al controller devono passare sull'unico bus presente (disco, stampante, muse, tastiera,ecc..) e viene utilizzato anche per la comunicazione tra la CPU e la memoria. Quest'ultimi dati sono tanti perché per ogni micro istruzione ho la fase di fetch, decodifica ed esecuzione.

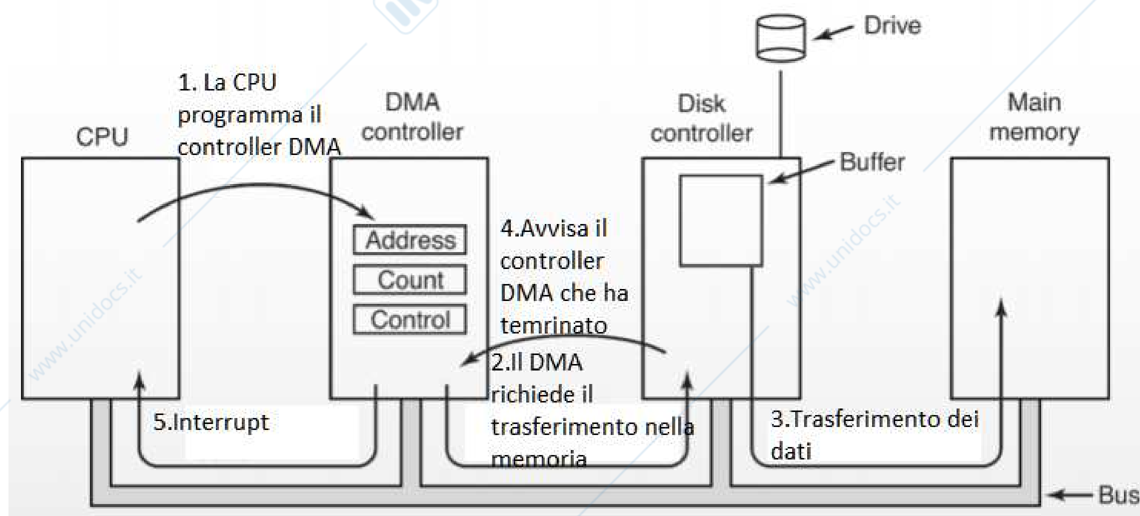


All'interno della CPU è presente un dispositivo filtro che decide quale bus utilizzare, se quello generale o quello dedicato tra la CPU e la memoria. Quest'ultimo bus può essere molto breve e quindi elettricamente non è difficoltoso da gestire e in più ha una capacità di trasferimento molto più alta del bus dati standard.

Direct Memory Access (DMA)

La CPU può richiedere dati ad un controller di I/O un byte alla volta, ma questa soluzione è troppo lenta per questo viene utilizzato la **DMA**. Il controller DMA contiene molti registri che possono essere scritti e letti dalla CPU, tra cui un registro *degli indirizzi di memoria*, *un registro dei conteggi* dei byte e un *registro di controllo*. Quest'ultimo è utilizzato per specificare le porte di I/O da usare, la direzione del trasferimento, l'unità di trasferimento e il numero di byte da trasferire alla volta.

Come funziona



1. La CPU invia una richiesta al controller DMA e imposta i registri.
2. Il DMA aggrega le richieste. Invia sul bus una richiesta di lettura al controller del disco e inserisce sul bus degli indirizzi, l'indirizzo di memoria su cui scrivere.
3. Il disco esegue e inserisce i dati richiesti nel proprio buffer. Al termine utilizza il bus e scrive i dati presenti nel buffer del controller del disco in memoria centrale utilizzando l'indirizzo di memoria indicato dal controller DMA.
4. Quando l'operazione di scrittura in memoria è completa, il controller del disco manda un segnale di conferma al DMA. Il DMA incrementa l'indirizzo di memoria da usare e diminuisce il conteggio dei byte. Ripetendo così le istruzioni 3 e 4 finché il conteggio è uguale a 0.
5. Quando il registro dei conteggi è uguale a 0 il DMA manda un interrupt alla CPU per avvisarla che il trasferimento è completato.

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

COME ADOPERANO I BUS:

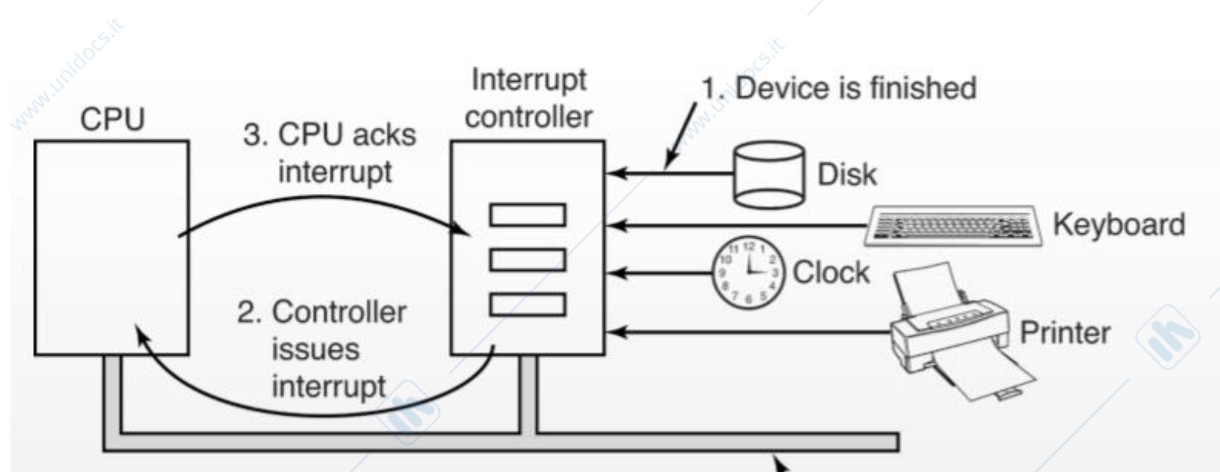
Problemi

È necessario garantire la *mutua esclusione* alla zona di memoria in cui il disco andrà a scrivere.
Il secondo problema è che il disco per andare a scrivere in memoria lo deve fare attraverso il bus dati e indirizzi e se usa il bus potrà andare ad interferire con le altre comunicazioni che ci sono. Due possibili soluzione a quest'ultimo problema sono:

1- Tecnica **BURST**, cioè i dati vanno inseriti in memoria, allora il controller entra in contesa per l'utilizzo del bus e si mette in coda. Appena ha la possibilità si impossessa del bus e trasferisce tutti i dati. In questo modo la CPU potrebbe non lavorare per qualche millisecondo.
2- La modalità chiamata **CYCLE STEALING**, controlla il bus e nel momento in cui nessuno sta utilizzando il bus, solo allora, lo usa. Non si riuscirà a trasferire tutto, ma è necessario attendere il prossimo periodo di inattività. Non sono intrusivo rispetto alle altre operazioni presenti sul bus. Questa soluzione è ottima in quanto non viene interrotta la CPU, ma allo stesso tempo si impiegherò più tempo per trasferire i dati

Non esiste una soluzione migliore, dipende da come è costruito l'hardware.

Interrupt Rivisitato



Come viene gestito un interrupt:

- La CPU fa richiesta di I/O
- Questa richiesta viene mediata dall'interrupt controller (non è il DMA controller)
- La perfireca ha generato o consumato il dato. La periferica comunica che ha terminato la sua operazione (che non è obbligatoriamente un interrapt). Invia un segnale su una linea del bus che gli è stata assegnata
- A sua volta l'interrupt controller capta il segnale e lo converte in un interrupt
- L'interrupt arriva alla CPU e se è disponibile lo gestisce immediatamente, altrimenti viene momentaneamente ignorato. In più segna un flag presente all'interno dell'interrupt controller che indica che l'interupt è stato gestito. Nel caso in cui il flag non viene aggiornato allora l'interrupt controller ha il compito di rinviare il segnale alla CPU cosa che una periferica non può fare.

Perché un interrupt può andare perso

- È possibile che l'interrupt non venga considerato, ad esempio se la CPU disabilita gli interrupt per garantire l'atomicità di alcune operazioni in casi specifici, ad esempio durante lo scheduling.

Differenza tra Interrupt e Trap:

- La **Trap** è una segnalazione su una condizione di errore o di richiesta di servizio. (divisone per zero, segment fault [avviene nella memoria segmentata], page fault) La TRAP accade in momenti ben precisi, ovvero nella vita della CPU può accadere di ricevere, ad esempio, una TRAP di page fault quando la CPU è nella fase di decodifica delle istruzioni. Oppure sarà possibile ricevere una Trap di divisione per zero quando la CPU si trova alla fine della fase di esecuzione di un'operazione.
  Quindi la TRAP arriva in maniera **sincrona**, ovvero è sincronizzata con una serie di eventi che si verificano nella vita della CPU.
- **L'interrupt** arriva in maniera **asincrona**, cioè quando gli pare.

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

Oggi i processori utilizzano più core e quindi è possibile che vengano eseguite più istruzioni non in sequenza e a velocità diverse, questo comporta che quando arriva l'interrupt è necessario consolidare lo stato della CPU, cioè lo stato esecutivo della CPU deve essere il più corretto possibile per effettuare una fotografia.
Ad esempio, non è corretto eseguire una fotografia della CPU mentre uno o più core sono a metà della fase di fetch.
Dato che quando arriva un interrupt le operazioni possono trovarsi in diversi stati di completamento, per questo motivo sarà necessario parlare di **Interrupt Preciso**, ovvero un interrupt nel momento in cui arriva mette la CPU in una condizione determinata/definita cioè il valore del program counter è conosciuto e preciso.

a) Nel caso in cui si ha una CPU con un solo core tutte le istruzioni precedenti al PC sono state eseguite e tutte le istruzioni dopo il PC non sono state eseguite. In questa condizione il valore del PC è corretto.
b) Mentre nel caso in cui si ha una CPU con più core si ha la situazione in cui il PC ha un valore ipotetico e si hanno vari livelli di esecuzione delle varie istruzioni. In questa situazione non si riesce a gestire l'interrupt, perché non è possibile conoscere il valore dei registri. Questo obbliga a creare degli interrupt che abbiano una validità temporanea. Infatti nel momento in cui arriva l'interrupt, la CPU ha un lasso di tempo breve e può o fare (17.38) delle istruzioni, oppure portarle al completamento il più veloce possibile. Se l'interrupt non è preciso, cioè se lo stato della CPU non è determinata il sistema non funzionerà.

Le proprietà dell'interrupt preciso sono:

1- Il Program Counter è salvato in un posto conosciuto
2- Tutte le istruzioni eseguite prima del PC sono state completate.
3- Nessuna istruzione successiva oltre a quella puntata dal PC è stata eseguita
4- Lo stato di esecuzione dell'istruzione puntata dal PC è conosciuto.

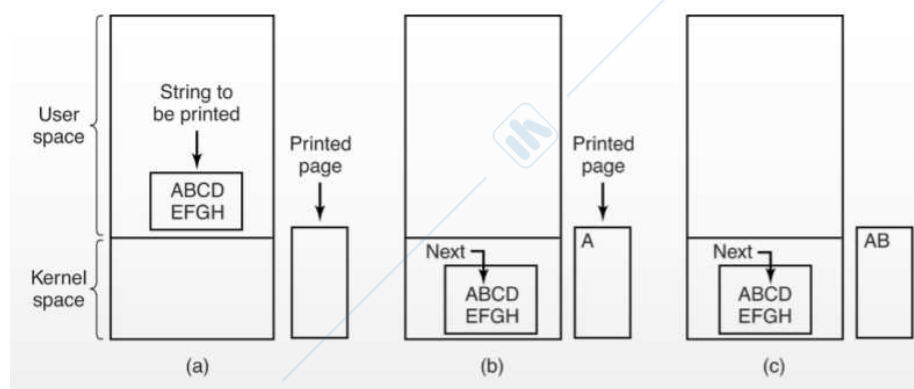
Principi del Software per I/O

Il Software per gestire l'I/O deve avere determinate caratteristiche, che sono:

1- **L'indipendenza dal dispositivo**, cioè progettare programmi che possano accedere a qualsiasi dispositivo di I/O senza dover specificare in anticipo il dispositivo. (lettura di un file da CD-ROM,ecc..)
2- **Nomenclatura uniforme** (Unform Naming), cioè il nome di un file o di un dispositivo dovrebbe esser semplicemente una stringa di caratteri o un numero intero e non dovrebbe dipendere dal dispositivo.
3- **Gestione degli errori**, generalmente gli errori vengono gestiti al livello hardware mentre i livelli più alti devono esser avvisati della presenza dell'errore solo se i livelli più in basso non sono riusciti a risolverlo. Ed è importate gestire tutti gli errori allo stesso modo.
4- È necessario definire se un'operazione di I/O a livello software è gestita in modo **asincrono** o **sincrono**. Ovvero se fare busy waiting oppure gestire le operazioni basate sugli interrupt, questo lo si decide mentre si scrive il kernel.
5- **Buffer**, ovvero non tutti i dispositivi (esempio la tastiera) hanno a disposizione un buffer interno e spesso i dati uscenti da un dispositivo non possono essere memorizzati direttamente nella destinazione finale e viene utilizzato un buffer interno (nel kernel credo).

I/O programmato

L'I/O sincrono con busy waiting il sistema non risponde fin quando l'I/O non termina. Quando si deve stampare un documento molto grande, la stampante è orientata al carattere, cioè viene inviato inviati un carattere per volta dove vengono memorizzati i dati. I dati inizialmente vengono inseriti in un buffer all'interno dell'applicazione nello spazio utente (a). Il processo utente



acquisisce la stampante per la scrittura eseguendo una chiamata di sistema. Quando la stampante sarà disponibile il processo utente esegue una chiamata di sistema richiedendo al sistema operativo di stampare la

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

stringa. Con l'I/O programmato, i dati vengono copiati in un buffer nello spazio del kernel (b). Successivamente vengono spostati nella periferica entrando in un ciclo (c) per copiare un carattere per volta. Generalmente le periferiche orientate al carattere sono più lente delle periferiche orientate al buffer.
L'aspetto essenziale dell'I/O programmato è che la CPU interroga continuamente il dispositivo, questo comportamento è chiamato **busy waiting**.

I/O gestito da interrupt

Mentre la CPU lavora possono giungere degli interrupt che la interrompono, salvando il processo attuale e concentrandosi su un altro processo con priorità più alta. Si ha un I/O asincrono in questo modello è possibile utilizzare anche il DMA.

Stratificazione dell'I/O

Il software per I/O è organizzato in 4 layer

Bisogna definire un software per I/O accessibile all'utente, questo deve utilizzare delle system call offerte dal sistema operativo ed indipendenti dai dispositivi. Le system call grazie ai devices drive vanno a specificare cosa devono fare. Il device drive dà a disposizione dei pezzi di software che collego alle routine di gestione dell'interrupt. Solo quest'ultime andranno sull'hardware.
Questo è un modello, ma ogni sistema lo implementa come vuole.

Gestione degli Interrupt

La procedura della gestione dell'interrupt.
Quando arriva un interrupt preciso alla CPU

1) metto in sicurezza lo stato della CPU, registri e PSW
2) Metto in sicurezza lo stato del processo, costruire un ambiente esecutivo con cui andrò ad eseguire il codice che gestisce l'interrupt.
3) Creo un contesto in cui posso eseguire il codice della interupt service routine, impostare la TLB, la MMU, la page table. Disporre uno stack per la procedura di servizio dell'interrupt.(ISR)
4) Invio un ACK (Acknowledge) all'interrupt controller. Il quale marca il lavoro come eseguito.
5) Copia dei registri salvati nella tabella dei processi.
6) Eseguo l'interrupt service routine (ISR)
- È un pezzo di codice che viene recuperato in base al valore dell'interrupt che è presente sul bus dati e agli eventuali parametri che arrivano insieme all'interrupt
7) Scelta di quale processo eseguire come successivo
- Viene attivato lo scheduler, perché è possibile che l'interrupt ha fatto si che qualche processo che era bloccato sia diventato pronto all'esecuzione ed ha una priorità maggiore del processo in esecuzione in quel momento (non è sempre valido)
8) Impostare il nuovo contesto del processo
9) Caricare i nuovi registri incluso il PSW del nuovo processo oppure del processo che era prima in esecuzione.
10) Avvio dell'esecuzione del nuovo processo.

Device driver

Il device driver è quel pezzo di software del kernel che si interfaccia con il dispositivo. Ciascun dispositivo di I/O necessita di un driver del dispositivo ed è solitamente fornito dal produttore. Il driver è parte del kernel e possono essere classificati in due diverse categorie.
Dispositivi a blocchi, come i dischi. **Dispositivi a caratteri**, come tastiere. Un driver di un dispositivo ha molte funzioni:

- Accetta richieste di lettura e scrittura
- Inizializza il dispositivo

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

- Verifica se il dispositivo è attualmente in uso
- Controllo del dispositivo, cioè vengono scritti i dati nei registri del controller del dispositivo e viene controllato se il controller ha accettato il comando e che sia pronto ad accettare il successivo. Questa sequenza continua fino a quando non sono stati inviati tutti i comandi.

Si possono verificare alcune complicanze, come

1. Un dispositivo di I/O potrebbe terminare mentre è in esecuzione un driver
2. Mentre un driver è in esecuzione il sistema può informarlo che l'utente ha rimosso improvvisamente quel dispositivo dal sistema. Deve essere annullato il trasferimento dell'I/O senza danneggiare alcuna struttura dati dal kernel, e deve esser rimosso qualunque richiesta del dispositivo svanito.
3. Ai driver non sono consentite chiamate di sistema, ma alcune chiamate a determinate procedure kernel sono permesse.

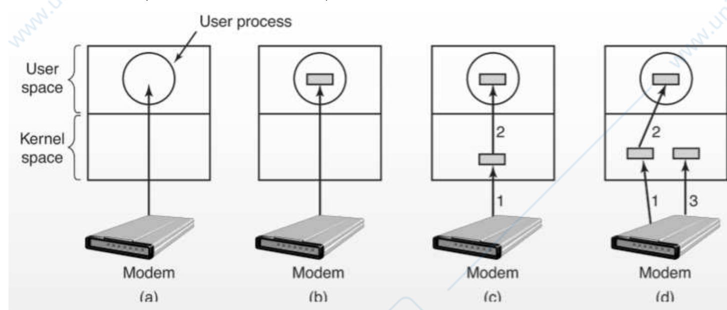
Software per I/O Indipendente dal dispositivo

La funzione principale del software indipendente dal dispositivo è di eseguire le operazioni di I/O che sono comuni a tutti i dispositivi e di fornire un'interfaccia uniforme al software del livello utente. Le funzioni sono:

- Interfacciamento uniforme dei driver dei dispositivi
- Denominazione dei dispositivi di I/O

Buffering

Viene utilizzato il buffering all'interno del kernel quando un dispositivo non ha un buffering al suo interno. (come la tastiera)



Considerando un processo che voglia leggere i dati dal modem:

a) Il processo utente esegue una chiamata di lettura e si blocca in attesa di un carattere. Ogni carattere in arrivo causa un interrupt. La procedura di servizio degli interrupt presenta il carattere al processo utente e si sblocca. Dopo aver messo il carattere da qualche parte, il processo si blocca di nuovo per legge un altro carattere. Il problema è che il processo utente deve essere riavviato ad ogni carattere in ingresso.
b) Il processo fornisce un buffer nello spazio utente. La procedura di servizio degli interrupt mette i caratteri in ingresso nel buffer finché non è pieno. A quel punto il processo utente si sveglia. Questo procedimento è più efficiente del precedente, ma il problema sorge mente si svuota il buffer e arriva un carattere.
c) È possibile utilizzare un buffer nel kernel e avere il gestore degli interrupt che vi mette i caratteri. Quando questo è pieno viene copiato nel buffer utente. Il problema sorge quando arrivano i caratteri mentre la pagina con il buffer utente viene letta dal disco.
d) Possiamo utilizzare due buffer nel kernel e uno nello spazio utente. Quando si riempie il primo buffer si utilizza il secondo, mentre si svuota il primo nel buffer utente. In questo modo i buffer fanno a turno. Uno schema di buffering di questo genere è detto **Buffering doppio**.
e) Un'altra forma di buffering utilizzata è **Buffer circolare**, essa è composta da una zona di memoria e due puntatori. Una punta alla parola libera successiva, dove poter inserire i nuovi dati. l'altro punta alla prima parola dei dati nel buffer che non è stata ancora rimossa.

Un buffer è controproducente quando i dati sono inseriti troppe volte al suo interno.

Error reporting

È necessario fare una classifica degli errori documentata.

Allocazione e rilascio dei dispositivi dedicati

Aggiungere una periferica ed eliminarla, mentre il calcolatore è in esecuzione.

Dimensione dei blocchi indipendente dal dispositivo

Dischi differenti possono avere settori di diverse dimensioni, sta al software indipendente dal dispositivo nascondere questo aspetto fornendo una dimensione di blocco uniforme ai livelli più in alto.

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

Deadlock

Il deadlock è una condizione di stallo in cui due o più processi sono in attesa bloccandosi a vicenda. I deadlock si possono verificare sia su risorse hardware (ad esempio un disco) che su risorse software (ad esempio una base di dati). Una risorsa è qualcosa che nel tempo può essere acquisita, usata e rilasciata. Possiamo avere due tipi di risorse

Risorsa Prelazionabili: può essere portata via dal processo che la possiede senza effetti dannosi.

Risorsa non-prelazionabili: non può essere portata via dal suo attuale proprietario senza causare il fallimento.

Solitamente i deadlock coinvolgono risorse non-preemptable, quelle preemptable possono essere risolte riallocando le risorse da un processo ad un altro.

I 5 filosofi

Ogni filosofo ha bisogno sia della forchetta di sinistra che quella di destra per poter mangiare. Se tutti prendono la forchette destra si viene a creare un deadlock.

!!!Chiesto all'esame dall'assistente!!!

```
1.       // il tavolo contiene tutto l'ambiente
2. public class Tavolo {
3.
4.     // il numero dei filosifi e delle forchette che vanno da 0 a 4
5.     static private int N = 5;
6.
7.     public static void main(String args[]) {
8.
9.         // creazione di un array di forchette
10.        Forchetta[] fList = new Forchetta [N];
11.        for (int i = 0; i < N; i += 1) fList[i] = new Forchetta ();
12.
13.        // creazione di un array di filosofi con forchette assegnate
14.        Filosofo[] pList = new Filosofo [N];
15.        for (int i = 0; i < N; i += 1) {
16.            pList[i] = new Filosofo (fList[i], fList[(i + 1) % N]);
17.            pList[i].start();
18.        }
19.
20.        // creazione e inizio dell'osservatore
21.        new Osservatore(pList).start();
22.    }
23. }
```

Al filosofo viene assegnato prima la forchetta di sinistra poi quella di destra. Il %n lo posso sostituire con un *if(i=n)* allora la forchetta di destra è uguale a 0.

```
1. // il filosofo farà il lavoro e quindi è l'oggetto attivo
2. public class Filosofo extends Thread {
3.
4.     private Forchetta left, right;
5.
6.     public char status; // how to show the philosopher (status)
7.     public char manoSinistra, manoDestra; // how to show each hand (with fork or not)
8.
9.     public Filosofo (Forchetta f1, Forchetta f2) {
10.        left = f1;
11.        right = f2;
12.        status = 'T'; //stato pensa
13.        manoSinistra = manoDestra = '_';
14.    };
15.
```


Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

```
16.
17.     public void run() {
18.         // Ogni filosofo pensarà e mangerà all'infinito
19.         while (true) {
20.             // think
21.             try { sleep (5); } catch (Exception e) {;}  //pensa
22.             left.raccogli(); manoSinistra = '^'; //prende la forchetta sx
23.             right.raccogli(); manoDestra = '^'; //prende la forchetta dx
24.             status = 'E'; // stato mangia
25.             // eat
26.             try { sleep (10); } catch (Exception e) {;} //mangia
27.             right.lascia(); manoDestra = '_';
28.             left.lascia(); manoSinistra = '_';
29.             status = 'T'; //stato pensa
30.         }
31.     }
32. }
```

Per metter giù le forchette è necessario invertire l'ordine rispetto a come sono state prese. Se la pausa sleep è molto minima (oppure viene eliminata) il sistema non funziona perché si verifica il deadlock in quanto una delle regole della sincronizzazione è quella di non fare mai assunzioni sulla velocità del processore o sul tempo in cui i processi non richiedono le risorse.

```
1.  public class Osservatore extends Thread {
2.
3.      // lista dei filosofi da stampare
4.      private Filosofo [] pList;
5.      //l'osservatore deve conoscere la lista dei filosofi
6.      public Osservatore  (Filosofo [] pl) {
7.          pList = pl;
8.      }
9.
10.     // l'osservatore stamperà periodicamente lo stato del tavolo
11.     public void run() {
12.         while (true) {
13.             for (int i = 0; i < pList.length; i += 1) {
14.                 System.out.print(pList[i].manoSinistra);
15.                 System.out.print(pList[i].status);
16.                 System.out.print(pList[i].manoDestra);
17.                 System.out.print(" ");
18.             }
19.             System.out.println();
20.
21.             try { sleep(500); } catch (Exception e) { ; } //pausa mezzo secondo
22.         }
23.     }
24. }
```

```
1.  // la mutua esclusione è sulla forchetta
2.  // la forchetta è un mutex
3.
4.  public class Forchetta {
5.
6.      private boolean busy;    // variabile da proteggere
7.
8.      public Forchetta  () { busy = false; }
9.
10.     // raccogli la forchetta
11.     public synchronized void raccogli() {
12.         while (busy) { // fin quando busy è true il thread rimane nel ciclo e viene messo in attesa
13.             try { this.wait(); }
14.             catch (Exception e) {;}
15.         }
16.         busy = true;
17.     }
18.
19.     // lascia la forchetta
20.     public synchronized void lascia() {
21.         busy = false;
22.         notifyAll(); //sveglia i thread che erano in wait
23.     }
24. }
```


Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

Problemi

1- È possibile acquisire la forchetta sinistra o destra e aspettare di acquisire l'altra. Questo errore si chiama under wait
2- È possibile che i filosofi entrino in un'attesa circolare, cioè tutti quanti con la forchetta destra ed uno con la forchetta sinistra

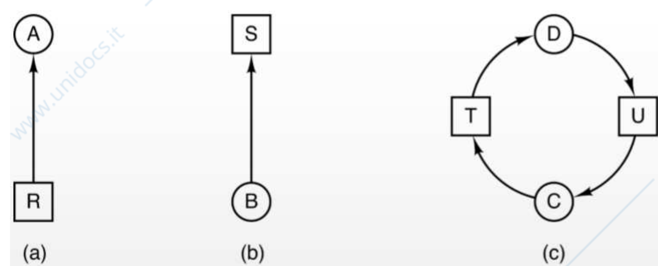
"Un deadlock è una situazione in cui un insieme di processi che sono in attesa di un evento che solo un altro processo dello stesso insieme può causare, ma anche questo processo è in attesa di altro."

Questo accade se si verificano una di queste quattro condizioni:

1. Mutua esclusione; alcune risorse vengono assegnate ad uso esclusivo
2. Hold and Wait; un processo può acquisire una risorsa e aspettare per acquisirne un'altra.
3. Impossibilità di prelazione; non è possibile sottrarre risorse già assegnate in precedenza. Devono esser esplicitamente rilasciate dal processo che le possiede.
4. Attesa circolare; si viene a creare una situazione di attesa circolare di due o più processi, ciascuno dei quali è in attesa di una risorsa trattenuta dal membro successivo della catena.

Qualora una di queste condizioni non si verifica, non è possibile parlare di deadlock delle risorse.

Un modello per deadlock



I cerchi sono i processi, mentre le risorse sono i quadrati. La freccia indica la direzione da un nodo risorsa a un nodo processo, il processo detiene la risorsa. Mentre se la freccia ha una direzione dal processo ad una risorsa, allora il processo aspetta la risorsa.

a) La risorsa R è assegnata al processo A
b) Il processo B aspetta la risorsa S
c) È un ciclo dove si verifica un deadlock

Infatti i grafi delle risorse sono un valido strumento che ci permette di capire se una data sequenza di richieste porti a una condizione di deadlock. Se nel grafo sono presenti dei cicli, allora sicuramente si verificherà un dedlock.

Alcuni metodi per affrontare i deadlock sono:

1- Ignorarli, ma dipende dalla loro incidenza.
2- Lasciar che avvenga il deadlock, rilevarlo e cercare di correggerlo.
3- Esclusione dinamica, cioè evitare il deadlock tramite lo scheduling (cioè un'allocazione attenta delle risorse). Infatti è possibile che lo scheduling trovi una combinazione che non mi provochi il deadlock, mentre un'altra volta riesce a trovare una combinazione che provochi il deadlock.
4- Prevenire tramite la negazione strutturale di una delle quattro condizioni richieste.

Algoritmo per identificazione dei cicli

Viene utilizzato algoritmo di dijkstra che permette di calcolare il percorso minimo da un nodo agli altri nodi connessi. Per ogni nodo del grafo è necessario eseguire l'algoritmo e controllare che nella lista creata dall'algoritmo non siano presenti doppioni, quindi che non si verificano cicli.

Algoritmo

1. Inizializzare una lista L vuota
2. Aggiungere il nodo corrente alla fine della lista L e verificare se il nodo appare due volte nella lista L. Se appare due volte allora il grafo presenta un ciclo ed è in deadlock e l'algoritmo termina.
3. Dal nodo corrente verificare se ci sono altre uscite non segnati. In tal caso passare al punto 4, altrimenti al punto 5.
4. Selezionare a caso un arco uscente e segnarlo come marcato ed eseguire dal passo 2
5. Se questo nodo è quello iniziale, il grafo non contiene cicli e l'algoritmo termina. Altrimenti si è raggiunto un vicolo cieco e bisogna tornare al nodo precedente e andare al passaggio 2.

Come trovare un deadlock con risorse multiple

Fino ad adesso abbiamo ipotizzato che esiste una sola copia di ogni risorsa, ma non è così nella realtà in quanto è possibile che siano presenti più risorse dello stesso tipo. Con la presenza di risorse multiple per rilevare i deadlock dobbiamo considerare un vettore di risorse esistenti e un vettore delle risorse disponibili. Dove le risorse disponibili saranno sempre minori o uguali alle risorse esistenti. È possibile scrivere in una

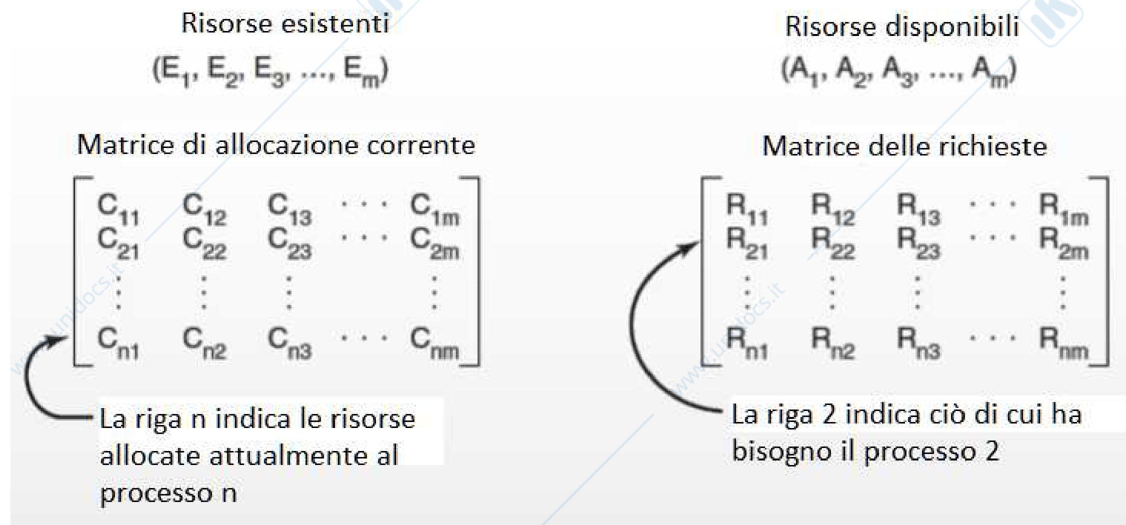
Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

matrice *C* quali sono le risorse che sono attualmente allocate ad ogni processo. Dove i processi sono le righe e le risorse le colonne, quindi C_{ij} è il numero di risorse j possedute dal processo i. Insieme a questa matrice è presente un'altra matrice *R* di richieste, cioè quante risorse di ogni tipo ogni processo ha bisogno per terminare l'operazione. Dove R_{ij} è il numero di istanze della risorsa j richiesta dal processo i. Possiamo osservare che

$$\sum_{i=1}^{n} C_{ij} + A_j = E_j \ per\ ogni\ j$$

Cioè se sommiamo tutte le istanze della risorsa j allocate e a queste aggiungiamo tutte le istanze disponibili, presenti nel vettore A, il risultato è il numero di istanze esistenti di quella classe di risorse, cioè il numero presente nel vettore E.
È possibile avere più richieste di quante risorse si ha, in questo caso il sistema non è in deadlock.



*la riga della matrice C indica per ogni riga un processo differente e di ogni processo il numero di ogni risorsa presente

Algoritmo

L'algoritmo di rilevamento dei deadlock si basa sul confronto tra i vettori. Avviene:

1. Si cerca di eliminare un processo dopo l'altro, cioè eliminando tutte le righe della matrice. Per fare ciò è necessario trovare un processo tale per cui le risorse che ha allocate più le risorse disponibili sono tali da soddisfare le sue richieste per arrivare al completamento. Quindi la somma di una riga della matrice C più il vettore A darà un risultato maggiore o uguale alla riga della matrice delle richieste R.
2. Trovato il processo si assegnano le risorse così al suo completamento esso rilascerà tutte le risorse a lui allocate aumentando così il numero di risorse disponibiliri rispetto a prima.
3. Ripeto l'operazione, cioè controllo se esiste un processo tale per cui con le nuove risorse disponibili potrebbe arrivare al compimento. Se esiste allora gli vengono assegnate le risorse.
4. Se al completamento tutti i processi vengono eliminati allora il sistema non è in deadlock, mentre se non trovo un processo tale per cui non è possibile fare queste operazioni allora il sistema è in deadlock e le righe di riferimento sono i processi che stanno generando il deadlock. Perché sono processi che non arriveranno mai a compimento anche se gli si assegnano tutte le risorse disponibili.

Risoluzione di un deadlock

Abbiamo a disposizione diversi modi

- **Prelazione delle risorse**; consiste nel prelevare una risorsa dal suo attuale proprietario e assegnarla a un altro processo. I problemi sono
  - nel scegliere nel modo corretto la risorsa da prelazionare il rischio è quello di mettere le strutture dati della risorsa in uno stato inconsistente.
  - Non tutte le risorse possono essere prelazionate.
- **Uso di Rollback**; questo metodo consiste nel generare dei **CHECKPOINT** dei processi, cioè scrivere il suo stato e lo stato delle risorse assegnate. Quando si verifica un deadlock, è possibile ripristinare l'ultimo checkpoint che è stato registrato. Questo funziona in casi specifici come in quei casi in cui la computazione dipende strettamente dall'I/O.
- **Eliminazione di processi**; se ne occupa il SO, il processo rilascerà tutte le risorse acquisite per renderle disponibili agli altri processi. Il problema è scegliere quale processo eliminare. Il processo vittima dovrebbe essere quello che impatta di meno sul sistema, ma questo il calcolatore non potrà saperlo e solitamente viene chiesto l'intervento dell'utente.

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

Evitare un deadlock

I principali algoritmi per evitare i deadlock si basano sul concetto degli stati sicuri. All'interno del sistema tutte le richieste di allocazione delle risorse vengono controllate, questa viene concessa se e solo se non porta il sistema in uno stato non sicuro. Uno stato del sistema si dice **sicuro** quando esiste almeno una sequenza di schedulazione delle risorse tale per cui tutti i processi arrivano a compimento.

Algoritmo del banchiere (singola risorsa)

L'algoritmo del banchiere considera ogni richiesta nel momento in cui si verifica e decide, eseguendo una simulazione utilizzando l'algoritmo delle matrici, se accettando la richiesta essa porti ad uno stato sicuro (cioè se tutti i processi possono essere soddisfatti) allora la richiesta viene accettata, altrimenti viene annullata.

Algoritmo del banchiere (risorse multiple)

Process	Tape drives	Plotters	Printers	Blu-rays
A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

Resources assigned

Process	Tape drives	Plotters	Printers	Blu-rays
A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

Resources still needed

E = (6342)
P = (5322)
A = (1020)

Si utilizza una matrice delle risorse assegnate, una matrice delle risorse necessarie per arrivare al compimento (risorse che ancora deve richiedere, mentre nell'algoritmo delle matrici si fa riferimento alle risorse totali richieste). P è il vettore delle risorse attualmente allocate, A il vettore delle risorse ancora disponibili, mentre E è il vettore delle risorse totali.

Algoritmo

1. Nella matrice delle risorse necessaria, viene individuata una riga, in cui i valori sono tutti minori o uguali al vettore delle risorse ancora disponibili. Se la riga non viene individuata, allora il sistema andrà in deadlock.
2. Assumendo che venga trovata la riga che soddisfi le condizioni precedenti, gli vengono assegnate le risorse necessarie per terminare. Al termine, la riga viene marcata come completa e vengono rilasciate le risorse utilizzate. Questi vengono aggiunti al vettore delle risorse disponibili.
3. Si ripete il passaggio 1 e 2 fino a quando tutti i processi non sono marcati come conclusi, in questo caso non si verifica nessun deadlock. Altrimenti se è presente una riga non marcata questa porterà al deadlock.

L'algoritmo del banchiere ha il difetto che vuole sapere in anticipo quante risorse sono necessarie e non è possibile prevedere il futuro. In più il numero dei processi non è fisso, ma variabile e le risorse possono anche scomparire. Per questi motivi principali l'algoritmo del banchiere non può essere utilizzato per prevenire i deadlock.

Prevenire un deadlock

Per evitare i deadlock è impossibile perché dovremmo predire il futuro. Allora torniamo alle 4 condizioni di **Coffman**.

1- **Mutua esclusione**: per evitare il problema della mutua esclusione, cioè che il software non abbia bisogno di semafori e/o monitor. Per realizzarla è possibile che le risorse non fossero mai assegnate esclusivamente a un singolo processo. Questo accade nei sistemi con **spooler** dove la risorsa su cui si effettua la mutua esclusione viene tolte dalla disponibilità generale e viene assegnata permanentemente ad un processo che la gestisce, altri processi che vogliono utilizzare le risorse lo chiedono al processo precedente.

2- **Hold and wait**: prendo la risorsa la tengo in mano e intento cerco di possedere l'altra risorsa. È possibile risolverla obbligando un processo a prendere tutte le risorse che gli servono in un colpo solo. Non è possibile dichiarare in anticipo tutte le risorse che sono necessarie, in questo casa sarà necessario rilasciare le risorse in possesso ed effettuare una nuova richiesta con le nuove risorse. Nel caso dei 5 filosi la forchetta è in condivisione con il vicino. È necessario aggiungere un elemento al sistema, cioè dichiarare un elemento (cameriere) in più nel modello il quale ha il compito di assegnare al filosofo entrambe le forchette se sono disponibili.

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

```
3-  PUBLIC CLASS TABLE {
4-
5-      // THE NUMBER OF PHILOSOPHERS AND FORKS
6-      STATIC PRIVATE INT N = 5;
7-
8-      PUBLIC STATIC VOID MAIN(STRING ARGS[]) {
9-
10-         // CREO UN CAMERIERE PER GESTIRE LE FORCHETTE
11-         MAITRE M = NEW MAITRE(N);
12-
13-         // CREATE AN ARRAY OF PHILOSOPHERS WITH ASSIGNED FORKS
14-         PHILOSOPHER[] PLIST = NEW PHILOSOPHER[N];
15-         FOR (INT I = 0; I < N; I += 1) {
16-             PLIST[I] = NEW PHILOSOPHER(M, I);
17-             PLIST[I].START();
18-         }
19-
20-         // CREATE AND START AN OBSERVER
21-         NEW OBSERVER(PLIST, M).START();
22-     }
23- }
```

```
1.  PUBLIC CLASS PHILOSOPHER EXTENDS THREAD {
2.
3.      PRIVATE MAITRE M;
4.      PRIVATE INT ID;
5.
6.      PUBLIC CHAR STATUS; // HOW TO SHOW THE PHILOSOPHER (STATUS)
7.      PUBLIC CHAR LEFTHAND, RIGHTHAND; // HOW TO SHOW EACH HAND (WITH FORK OR NOT)
8.
9.      PUBLIC PHILOSOPHER(MAITRE MT, INT I) {
10.         M = MT;
11.         ID = I;
12.         STATUS = 'T';
13.         LEFTHAND = RIGHTHAND = '_';
14.     };
15.
16.     PUBLIC VOID RUN() {
17.         // OGNI FILOSOFO PENSERÀ E MANGERÀ ALL'INFINITO
18.         WHILE (TRUE) {
19.             // THINK
20.             // TRY { SLEEP (5); } CATCH (EXCEPTION E) {;}
21.             // TO IMPROVED CONSISTENCY, HANDS AND STATUS GET INTO THE CRITICAL REGION
22.             // AND WILL BE MANAGED BY THE MAITRE
23.             M.GET(THIS, ID);
24.             // EAT
25.             // TRY { SLEEP (10); } CATCH (EXCEPTION E) {;}
26.             M.GIVEBACK(THIS, ID);
27.         }
28.     }
29.
30. }
31. }
```

```
1.  PUBLIC CLASS MAITRE {
2.
3.      PRIVATE BOOLEAN[] BUSY;
4.
5.      PUBLIC MAITRE(INT N) {// IDENTIFICATIVO DEL CAMERIERE N
6.          BUSY = NEW BOOLEAN[N];//ARRAY CHE INDICA SE LE FORCHETTE SONO DISPONIBILI
7.          FOR (INT I = 0; I < N; I += 1) BUSY[I] = FALSE; //INIZIALIZZAZIONE DELL'ARRAY A FALSE
8.      }
9.
10.     PUBLIC SYNCHRONIZED VOID GET(PHILOSOPHER P, INT I) {
11.         WHILE (BUSY[I] || BUSY[(I + 1) % BUSY.LENGTH]) { //FIN QUANDO ENTRAMBE LE FORCHETTE NON SONO LIB
ERE VIENE MESSO IN ATTESA
12.             TRY { WAIT(); }
13.             CATCH (EXCEPTION E) {;}
14.         }//ESCO DA WHILE QUANDO TUTTE E DUE LE FOCHETTE SONO LIBERE E LE POSSO BLOCCARE
15.         P.LEFTHAND = P.RIGHTHAND = '^';
```


Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

```
16.             p.status = 'E';
17.             busy[i] = busy[(i + 1) % busy.length] = true;
18.         }
19.
20.         public synchronized void giveBack(Philosopher p, int i) {//lascio entrambe le forchette
21.             busy[i] = busy[(i + 1) % busy.length] = false;
22.             p.leftHand = p.rightHand = '_';
23.             p.status = 'T';
24.             notifyAll(); // this is sub-
    optimal,risveglia i processi che sono in attesa sulle forchette precedentemente rilasciate
25.         }
26.
27.         // per avere un output affidabile è necessario sincronizzare la stampa
28.         public synchronized void printStatus(Philosopher[] pList) {
29.             for (int i = 0; i < pList.length; i += 1) {
30.                 System.out.print(pList[i].leftHand);
31.                 System.out.print(pList[i].status);
32.                 System.out.print(pList[i].rightHand);
33.                 System.out.print(" ");
34.             }
35.             System.out.println();
36.             // notifyAll is NOT required
37.         }
38. }
39.
40. }
```

```
1.  public class Observer extends Thread {
2.
3.      // list of philosophers to print out
4.      private Philosopher[] pList;
5.
6.      // we must now ask the maitre to print out the system status
7.      private Maitre m;
8.
9.      public Observer(Philosopher[] pl, Maitre mt) {
10.         pList = pl;
11.         m = mt;
12.     }
13.
14.     // l'osservatore stamperà periodicamente lo stato del tavolo
15.     public void run() {
16.         while (true) {
17.             m.printStatus(pList);
18.             try { sleep(500); } catch (Exception e) { ; }
19.         }
20.     }
21. }
```

3- **Non prelazionabilità**: Togliere una risorsa improvvisamente ad un processo a cui era stata assegnata, non è sempre la soluzione migliore soprattutto perché non sempre è facile identificare quale risorsa è conveniente scegliere. Una possibile soluzione è quella di virtualizzare alcune risorse, ma non tutte le risorse possono essere virtualizzate, come i database.
4- **Circular wait**: alle risorse richieste viene assegnato un numero globale e durante l'esecuzione si è tenuti a rispettare l'ordine di esecuzione. Se è necessaria una risorsa precedente è obbligatorio rilasciare tutte le risorse successive alla risorsa richiesta.

```
5-  public class Table {
6-
7-      // the number of philosophers and forks
8-      static private int N = 5;
9-
10-     public static void main(String args[]) {
11-
12-         // create an array of forks
13-         Fork[] fList = new Fork[N];
14-         for (int i = 0; i < N; i += 1) fList[i] = new Fork();
15-
16-         // create an array of philosophers with assigned forks
17-         Philosopher[] pList = new Philosopher[N];
```


Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

```
18-         for (int i = 0; i < N; i += 1) {
19-             // we are imposing a global order in resource allocation
20-             int firstForkIndex = i;
21-             int secondForkIndex = (i + 1) % N;
22-             if (firstForkIndex < secondForkIndex)
23-                 pList[i] = new Philosopher(fList[firstForkIndex], fList[secondForkIndex]);
24-             else
25-                 pList[i] = new Philosopher(fList[secondForkIndex], fList[firstForkIndex]);
26-             // In alternative, we could keep this part and have the philosopher allocate
27-             // resources in order. In such case we must add an index field to the Fork class
28-             pList[i].start();
29-         }
30-
31-         // create and start an observer
32-         new Observer(pList).start();
33-     }
34- }
```

```
1.  // a fork is essentially a mutex
2.
3.  public class Fork {
4.
5.      private boolean busy;    // variable to protect
6.
7.      public Fork() { busy = false; }
8.
9.      // pick up the fork
10.     public synchronized void pick() {
11.         while (busy) {
12.             try { this.wait(); }
13.             catch (Exception e) {;}
14.         }
15.         busy = true;
16.     }
17.
18.     // put down the fork
19.     public synchronized void drop() {
20.         busy = false;
21.         notifyAll();
22.     }
23. }
24. }
```

```
1.  public class Observer extends Thread {
2.
3.      // list of philosophers to print out
4.      private Philosopher[] pList;
5.
6.      public Observer(Philosopher[] pl) {
7.          pList = pl;
8.      }
9.
10.     // the observer will periodically print the status of the table
11.     public void run() {
12.         while (true) {
13.             for (int i = 0; i < pList.length; i += 1) {
14.                 System.out.print(pList[i].leftHand);
15.                 System.out.print(pList[i].status);
16.                 System.out.print(pList[i].rightHand);
17.                 System.out.print(" ");
18.             }
19.             System.out.println();
20.
21.             try { sleep(500); } catch (Exception e) { ; }
22.         }
23.     }
24. }
25.
26. }
```


Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

```
1.  public class Philosopher extends Thread {
2.
3.      private Fork left, right;
4.
5.      public char status; // how to show the philosopher (status)
6.      public char leftHand, rightHand; // how to show each hand (with fork or not)
7.
8.      public Philosopher(Fork f1, Fork f2) {
9.          left = f1;
10.         right = f2;
11.         status = 'T';
12.         leftHand = rightHand = '_';
13.     };
14.
15.     public void run() {
16.         // Each philosopher will endlessly think and eat
17.         while (true) {
18.             // think
19.             // comment out the following line and you will see a deadlock
20.             // try { sleep (5); } catch (Exception e) {;}
21.             left.pick(); leftHand = '^';
22.             right.pick(); rightHand = '^';
23.             status = 'E';
24.             // eat
25.             // comment out the following line and you will see a deadlock
26.             // try { sleep (10); } catch (Exception e) {;}
27.             right.drop(); rightHand = '_';
28.             left.drop(); leftHand = '_';
29.             status = 'T';
30.         }
31.     }
32.
33. } }
34.
```

Altre questioni

- Two-phase locking: consiste in due fasi, la prima è quella di provare a bloccare tutte le risorse (nel caso dei filosofi blocca l'intero tavolo) accaparrarsi le risorse necessarie e rilasciare tutte le altre. Questa soluzione aumenta la complessità del codice.
- Communication deadlock: è un tipo di deadlock che si verifica principalmente nei sistemi distribuiti, cioè un insieme di calcolatori uniti attraverso una rete. A volte essi richiedono delle risorse non presenti nello stesso sistema, per questo il deadlock è difficile da rintracciare ed in più si potrebbero verificare degli errori a livello di protocollo in modo tale da inviare un messaggio sbagliato e causare un deadlock.
- Livelock: si verifica quando due processi utilizzano il proprio quanto di tempo per acquisire la stessa risorsa senza fare progressi. (perché il quanto di tempo scade) Il livelock è una situazione simile al deadlock(ma non lo è), in cui le entità non sono effettivamente bloccate, ma di fatto non fanno alcun progresso.
- Starvation: viene obbligato un processo ad aspettare un tempo enormemente alto per poter accedere ad una risorsa. La starvation non implica il deadlock in quanto le altre entità possono progredire, mentre il deadlock implica la starvation in quanto nessuna entità farà alcun progresso per questo il deadlock è una forma "terminale" di starvation.

Come affrontare un deadlock

Ignorare: algoritmo dello struzzo.

Trovare:

- Trovare una risorsa attraverso il grafo delle risorse.
- Trovare più risorse attraverso l'algoritmo delle matrici.

Risolvere:

- Prelazione di una risorsa, se è possibile viene prelevata temporaneamente una risorsa
- Rollback
- Eliminazione dei processi

Evitare:

- Traiettoria delle risorse; la traiettoria non deve finire nello spazio di deadlock
- Stato sicuro/non sicuro
- Algoritmo del banchiere

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

CLOUD COMPUTING

Il cloud è sempre esistito. L'obiettivo è quello di ottimizzare l'uso delle risorse unendo più server virtualizzati all'interno di una singola macchina fisica. Questo comporta un notevole risparmio perché bisogna comprare meno hardware e meno risorse sprecate.

Caratteristiche e problemi della virtualizzazione

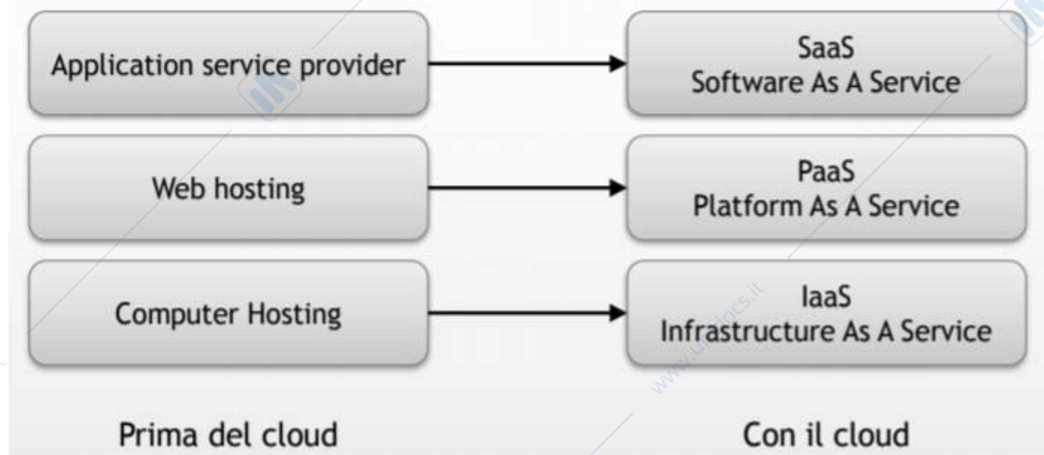
- Comunque devo comprare un calcolatore su dove virtualizzare.
  - Il calcolatore consuma energia e deve essere raffreddato
  - Dopo un tot di anni è necessario cambiarlo perché non c'è più un supporto o pezzi di ricambio
- È necessario fare manutenzione
- Il sistema operativo che ospita diventa critico, cioè se si ferma la macchina fisica anche le macchine virtuali si bloccano. Questo si chiama Single point of failure.

Soluzioni

- **Housing**: si compra il calcolatore e si chiede di ospitarlo a qualche azienda, pagando un affitto. Elettricità e raffreddamento sono un problema dell'azienda. Questa soluzione non viene più utilizzata ad oggi.
- **Hosting**: si prende in affitto il calcolatore di qualche azienda e dove sia l'hardware e la manutenzione sono un problema dell'azienda. Oggi l'azienda invece di affittare l'intero calcolatore fisico, affitta una macchina virtuale all'interno di un sistema di virtualizzazione. Questo diminuisce i costi ed in più è possibile sposare la macchina a piacere, quando si effettua un aggiornamento tecnologico.
- **Cloud**: Il cloud è un modello per abilitare facilmente, su richiesta di accesso alla rete ad un pool condiviso di risorse di calcolo configurabili. Che possono essere forniti rapidamente con uno sforzo di gestione minimo senza bisogno di una iterazione costante.

Livelli dei cloud (As a service)

Architettura SPI



SAAS

È la virtualizzazione di una applicazione, serve all'utente finale.

- **Vantaggi**: il più grande vantaggio è la scalabilità. Infatti le applicazioni possono essere spostate rapidamente su hardware diversi in risposta all'aumento della domanda. Altri vantaggi sono backup, prezzi flessibili, portabilità.
- **Svantaggi**: estrema fiducia nel provider, in quanto la responsabilità sulla sicurezza ricada su di lui. In caso di instabilità del server, il provider potrebbe facilmente trasferire l'applicazione su un altro server.

PAAS

È un tipo di servizio venduto tipicamente agli sviluppatori. Consiste nell'offrire una macchina virtuale dove poter progettare e fornire un proprio servizio.

- **Vantaggi**: riduce i tempi di manutenzione e fornisce una elevata quantità di personalizzazione del servizio.
- **Svantaggi**: è necessario del tempo per portare un'applicazione da una piattaforma ad un'altra.

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

IAAS

È un servizio che consiste nell'accedere a risorse fisicamente separate. Si ottiene un calcolatore virtuale.

- **Vantaggi**: possibilità di salvare lo stato del server virtuale in qualsiasi momento, avendo così una comoda procedura di backup. Anche se questo comporta un aumento dei costi.
- **Svantaggi**: la sicurezza non è elevata infatti non è raccomandabile far gestire dati sensibili, come dati di carte di credito. In più il provider potrebbe decidere di chiudere il server senza preavviso se individua comportamenti sospetti (attività non etiche) oppure se il server è stato compromesso dagli hacker.

Utility computing

Il Cloud viene definito "Utility computing", cioè la capacità di calcolo non viene vista come uno strumento che serve per poter lavorare, ma diventa anche una risorsa. I servizi di calcolo vengono erogati da qualcuno con cui ho un contratto e il mondo in cui ho di accedervi è quello di mandarmi una fattura basandosi su quanto uso ho fatto delle risorse. se usiamo il cloud come utility computing questo dovrebbe essere

1) Efficiente nell'uso delle risorse,
2) Scalabile, nel momento in cui ho necessità di tante risorse, queste saranno disponibili. Mentre nel caso in cui non ho tanta necessità le risorse saranno impiegate in altro.
3) Elastico, possibilità di aggiungere e rimuovere risorse a secondo del bisogno.
4) Senza manutenzione, cioè a carico del provider
5) Sempre disponibile, o meglio quasi sempre
6) Interoperabile e portabile, posso combinare servizi di cloud diversi e trasportare i miei dati da un cloud all'altro

Caratteristiche del cloud

1) Efficienza nell'uso delle risorse
   - Le risorse vengono aggregate e uniformate in maniera tale da fornire un servizio omogeneo a più utenti.
   - L'allocazione delle risorse viene adattata dinamicamente in base alle richieste degli utenti
2) Elasticità
   - Le risorse di calcolo possono essere aggiunte e rimosse rapidamente in maniera tale da ottenere la scalabilità necessaria alle richieste degli utenti.
3) Scalabilità, cioè la capacità di un sistema di mantenere inalterata la sua efficienza anche di fronte ad un carico di lavoro variabile. Si misura in job per unità di tempo.
4) Senza manutenzione esterna
   - I servizi del cloud vengono erogati senza richiedere l'intervento di un operatore umano.
5) Accessibile e sempre disponibile
   - Le risorse devono essere sempre disponibili eventualmente prevedendo meccanismi di ridondanza a carico del provider.
   - L'accesso alle risorse deve avvenire tramite meccanismi standard compatibili con piattaforme diverse.

Tutte queste caratteristiche interessano il provider e no l'utente finale.

Tre tipi di cloud

- Pubblico
  - L'infrastruttura è accessibile attraverso Internet, l'uso può essere libero o a pagamento.
- Community
  - L'infrastruttura è condivisa da varie entità con richieste infrastrutturali omogenee (sarebbe il peer to peer)
- Privato
  - L'infrastruttura è strettamente interna alla mia azienda, e la uso per i miei servizi
- Ibrido
  - L'infrastruttura ha dei segmenti di cloud privato (mio) che sfruttano servizi di un cloud pubblico (di miei fornitori)

Il lato oscuro del cloud

1. Il cloud viene utilizzato soprattutto per tagliare i costi
2. Le banche, in un momento qualche del giorno le banche devo fornire i dati di un determinato giorno. Questo tipo di richieste vengono fatte soprattutto per cause in tribunale. Le banche operano su due indici, OPEX i costi che si hanno mensilmente per i servizi, CAPEX i costi per i beni. (comprare un

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

calcolatore CAPEX, comprare una macchina virtuale è OPEX). Alla banca interessa che l'utente abbia un OPEX più alto possibile perché questi possono essere eliminati, mentre rivendere i beni è più difficile.
3. La rete fa differenza, perché ci sono dei costi di connettività che solitamente non vengono inclusi nel preventivo.
4. Non ci sono standard definiti, si ha un meccanismo documentato per arrivare a parlare con il provider http, quale lingua

SISTEMI MULTIMEDIALI

Il dato multimediale

Il dato multimediale viene utilizzato immediatamente. Il dato segue delle dinamiche tutte sue perché è tollerabile alle perdite, ma dipende dal tipo. Infatti il dato video è più tollerabile di quello audio. Il dato multimediale è analogico, mentre un calcolatore è digitale. In fine il dato multimediale occupa molto spazio. La prima occasione in cui si viene ad utilizzare il dato multimediale è la telefonia. Inizialmente questo trattamento era analogico in tempi recenti si è iniziato a traferire queste informazioni in forma digitale. Perché i bit possono essere immagazzinati, mentre la forma d'onda non posso farla.

Caratteristiche delle informazioni multimediali

(all'intenro di un sistema operativo)

1. Sono digitali (stanno in un file); devono essere codificate in qualche modo. Codifica cioè usare un'informazione per trasferire la sua rappresentazione da un domino ad un altro. Esistono varie codifiche, codifica digitale che trasforma una forma d'onda in una sequenza di bit, una codifica mpeg che trasforma una sequenza di matrici in un file con certe caratteristiche.
2. Occupano molto spazio, per questo è necessario fare la compressione. Comprimere è un effetto di una buona codifica.
3. Devono poter garantire la fruizione in real time
4. Devono poter essere usate alla stregua di un media "classico", ovvero di poter mettere pausa, andare avanti,ecc..

Codifica di un dato multimediale

Codifica consiste nel rappresentare un'informazione da un dominio ad un altro, cioè stabilire una corrispondenza tra informazione analogica e sequenze di bit per rappresentarla.

Motivi per codificare

Per dare sicurezza, per evitare che il contenuto sia accessibile a chiunque (cifratura), oppure per proteggere la proprietà intellettuale.

- Watermarking: si riferisce all'inclusione di informazioni all'interno di un file multimediale che può essere successivamente rilevato o estratto per trarre informazioni sulla sua origine e provenienza.
- Steganografia: è una tecnica di occultamente delle informazioni. I dati che si vuole proteggere vengono occultati grazie l'uso di altri dati meno sensibili che risulteranno a prima vista più leggibili scoraggiando l'utente malintenzionato a ricerca i dati sensibili occultati.

Per dare affidabilità, infatti una giusta codifica può aiutare a correggere gli errori di trasmissione.

Codificare NON vuol dire comprimere. Una compressione è una codifica che fa risparmiare spazio. La compressione di un contenuto è un effetto collaterale di una codifica ben strutturata.

Dumb coding

Consiste nel codificare il filmato come una sequenza di frame
Codifico ogni frame come una matrice di pixel
Codifico ogni pixel in formato RGB (3byte)

Ottimizzazione dello spazio

Esistono tecniche di codifica per creare delle associazioni tra un insieme di informazioni da rappresentare e stringhe binarie (bit) tali per cui il numero totale di bit utilizzati è minimo. Queste tecniche non fanno perdere informazioni; riducono solo lo spazio totale occupato. Queste codifiche vengono definite **lossless**.

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

Ridondanza

Un sistema di codifica può tentare di scartare le informazioni inutili (non captate dal nostro cervello) o che potrebbero essere ricostruite in maniera automatica. Questo tipo di codifica vengono chiamate **lossy** perché ci permettono di risparmiare spazio perdendo informazioni.
La codifica lossy lavora sul concetto di ridondanza.

- Ridondanza spaziale; cioè in un contenuto dove sono presenti informazioni simili, queste informazioni vengono aggregate.
- Ridondanza temporale; cioè informazioni che si susseguono nel tempo non variano enormemente in due istanti successivi e li possono approssimare.

Nell'uomo la percezione sull'asse verticale è molto amplificata rispetto all'asse orizzontale. In più il nostro cervello reagisce più facilmente alle alte frequenze rispetto alle basse frequenze.

La compressione dei dati è una cosa difficoltosa da trattare, essa dipende da tre parametri che vengono racchiusi in tre categorie:

- Dal mezzo con cui si fruisce del contenuto.
- Da agenti fisiologici.
- Da agenti psicologici.

Il mezzo di fruizione

Il mezzo da cui fruisce il contenuto impone un limite tecnologico. Infatti è possibile che capiti di codificare/inviare delle informazioni che semplicemente non potranno essere fruite in quanto il dispositivo destinatario non ha la tecnologia adatta per riprodurre il contenuto.

Parametri fisiologici

Non tutte le informazioni hanno la stessa importanza per il nostro cervello.

- Di una figura percepiamo più facilmente la forma.
- Di un suono siamo più sensibili alle variazioni istantanee.
- Il cervello ricostruisce autonomamente parte dell'informazione, ma per ogni senso lo fa con soglie e parametri diversi.

Parametri psicologici

Il cervello riconosce l'informazione mancante, ma COME LA VORREMMO e non come realmente è. Infatti l'essere umano tende a riconoscere gli oggetti istintivamente, basandosi sull'esperienza.

Audio e video

L'audio e il video sono dei dati multimediali diversi e devono essere tratti in maniera disgiunta. I vincoli sono:

- Tecnici, cioè hanno campionamento differente.
- Fisiologici, la sensibilità non è uniforme o distribuita diversamente. Il volume del PC è logaritmico ed è necessario aggiustare la scala, mentre l'occhio ha una sensibilità diversa in base a quanto un colore è più vicino al bianco e/o nero.
- Psicologici, la diversa tolleranza alla perdita dei dati.

MPEG

L'MPEG è una organizzazione fondata dalla ISO e ha il compito di definire gli standard per la compressione audio e video. Esso comprende un insieme di regole per l'eliminazione della ridondanza spaziale e temporale in maniera tale da non disturbare troppo la percezione dell'utente medio.

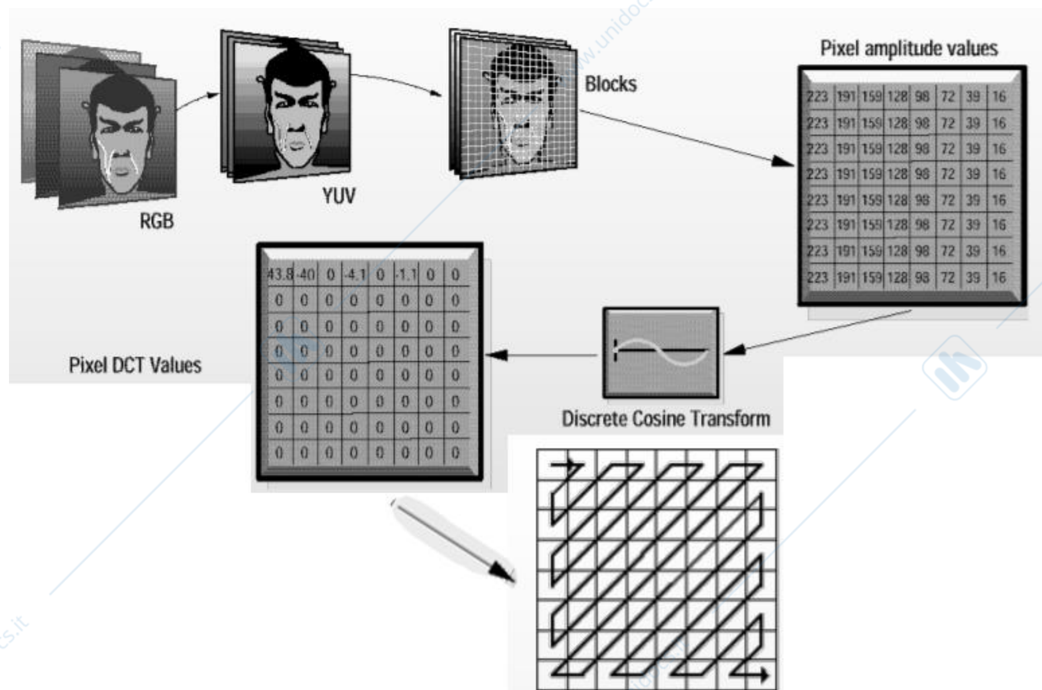
La codifica MPEG

- È una codifica asimmetrica, il codificatore fa tutto il lavoro pesante e deve creare uno stream corretto. Il decodificatore fa operazioni semplici e deve essere poco oneroso per il calcolatore.
- Tenendo fisso lo standard, è possibile implementare codificatori più efficienti senza modificare il software di visualizzazione. Oppure è possibile creare nuovi player (e nuovi dispositivi) senza codificare nuovamente i contenuti.
- MPEG costruisce un video codificando, secondo alcuni parametri per ridurre le ridondanze, una sequenza di fotogrammi (non è una sequenza di immagini jpeg; quella si chiama MJPEG)

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

Ridondanza spaziale

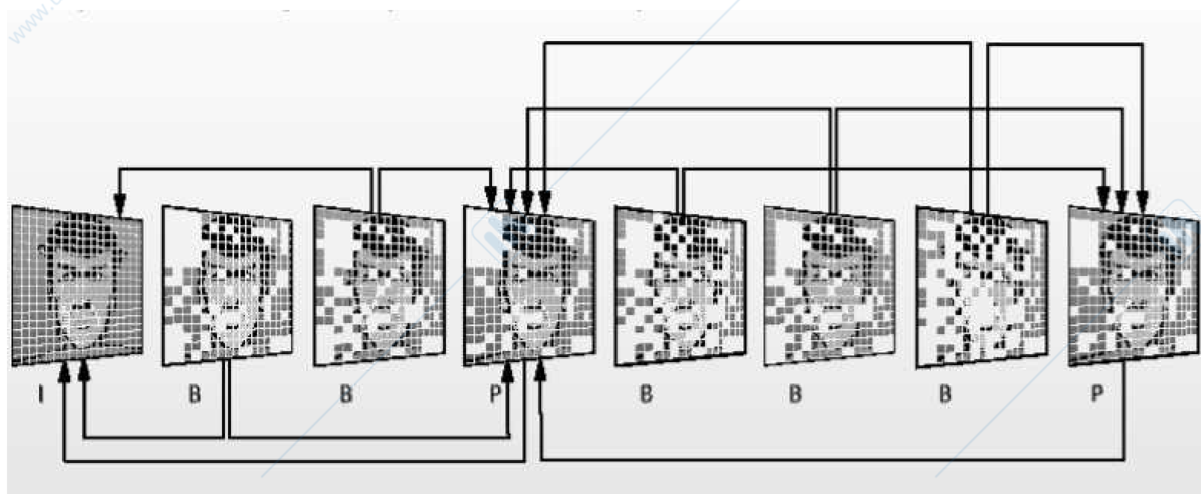


Prendendo in considerazione un frame, esso è costituito da tre livelli, rosso verde e blu (RGB). Vengono effettuate codifiche separate per i tre colori nel formato YUV. Questo cambio di colore ci permette di rappresentare meglio le alte frequenze.
Per ogni componente di colore viene suddivisa in blocchi solitamente 8x8 in modo da esser "ripulita" dalle basse frequenze ed ottenere le alte (cioè i bordi). Di ogni blocco considero il primo blocco che si trova a sinistra e rappresento i valori come differenza rispetto al primo. Viene applicato un filtro, una funzione della trasformata dicerta del coseno. A questo punto ottengo una matrice a cui applico una **Linearizzazione a ZigZag** partendo dal primo blocco ed effettuando un percorso a zig zag verso l'ultimo blocco più a destra. Ricordando che in alto a sinistra si trova il valore più grande, mentre tutti gli altri saranno molto vicino allo 0. Con queste condizioni di partenza la costruzione di un codice a lunghezza minima è molto efficiente.

Ridondanza temporale

Per quanto riguarda la ridondanza temporale vengono definiti diversi tipi di frame, abbiamo i frame *indipendenti I*, frame che *predicano P* che indicano la differenza rispetto al frame precedente e frame *bidirezionali B* i quali descrivono una scena per differenza con quelle sia precedente che successive.
La variazione nel tempo può essere trasmessa come differenza di immagini o come lo spostamento di un rettangolo all'interno dell'immagine, questi rettangoli prendono il nome di macro-blocchi.

GOP – Group of pictures



Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

È una sequenza di frame I, P e B dove:
- Gli I-frame sono codificati indipendentemente, come immagini jpeg
- I P-frame sono codificati per differenza rispetto all'I-frame che li precede, includendo i vettori di movimento
- I B-frame sono codificati come differenza con gli I/P-frame che li precedono/seguono

Definizioni

- Una *codifica* esprime il modo in cui l'informazione audio o video viene trattata. (MPEG, H264)
- Un *container* (contenitore) dichiara la sintassi con cui i bit codificati vengono scritti in un file. (.avi, .mov)

Streaming

Si parla di streaming tutte le volte che:
- Vi è del contenuto che cominciamo ad utilizzare prima che sia arrivato completamente
- L'invio del contenuto avviene in maniera continua e asincrona fintanto che non è stato inviato tutto.

Si verificano principalmente due problemi nello streaming. Il problema del carico di dati sulla rete e il tempo di trasferimento dall'hard drive (file system). Quest'ultimo non sarà mai regolare in quanto il SO sta svolgendo anche altre operazioni, perché il disco e il video non sono sincronizzati, e per la presenza di eventuali interrupt imprevedibili.
In più anche i frame non sono tutti uguali, infatti i frame I, P e B hanno **tempi di trasferimento** diversi, in quanto hanno dimensioni differenti, e hanno **tempi di elaborazione** diversi. Infatti solo il frame I è indipendente, mentre per elaborare un frame è necessario conoscere tutti i frame B che precedono un frame P.

Buffering

L'unico modo che un'applicazione ha di rendere nuovamente regolare un contenuto multimediale è quello di farlo passare per un buffer. I dati entrano come il sistema concede (senza regole). I dati escono secondo il profilo richiesto (in modo regolare).

Dimensione del buffer

- Troppo piccolo non va bene;
  - Se non c'è spazio i dati in arrivo verranno scarti e persi
  - Si potrebbe andare in buffer overrun/underrun
- Troppo grande non va bene
  - Si introducono ritardi incettabili

Buffer underrun

È quando si ricevono dei dati troppo lentamente e si arriva ad un punto nello streaming che non si hanno più dati da consumare. Nel grafico questo si nota in quanto i due segmenti si toccano (slide 44)

Buffer overrun

È quando ricevo dati troppo velocemente tanto da riempire il buffer ed è necessario buttare via i dati che arrivano.

E se perdiamo dei dati?

Il dato multimediale è strutturato per tollerare la perdita di dati, infatti vengono forniti dei dati a caso (rumore) o degli zeri in sostituzione alla perdita di dati, anche se a volte il risultato non è bellissimo. La tolleranza del nostro occhio è del 20 e il 25% delle informazioni video perse. La tolleranza del nostro udito è del 3%

Funzionalita' VCR

Queste funzionalità danno la possibilità di selezionare un punto durante la riproduzione, di andare avanti e indietro, mettere pausa, ecc..
Solitamente questo comporta una lenta ripartenza da un fotogramma solitamente successivo a quello che avevamo scelto. Questo perché il buffer viene svuotato e si ricostruisce quel minimo di informazioni che servono per una fruizione garantita. Solitamente il punto di partenza è sempre un I-frame.
L'unico modo per saltare (andare in avanti o indietro nel video) in modo efficiente da un frame all'altro è quello di garantire un modo efficiente tra un I-frame all'altro. Per minimizzare i disturbi alla ricezione dei dati è necessario agire sul sistema operativo.

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

1. Tempo di elaborazione il più costante possibile, è necessario uno scheduler specializzato per il supporto ai sistemi multimediali
2. Tempo di estrazione più uniforme possibile, serve un file system specializzato al contenuto multimediale.
3. Funzionalità VCR senza problemi, saltare da un I-frame all'altro senza problemi.

Processi multimediali

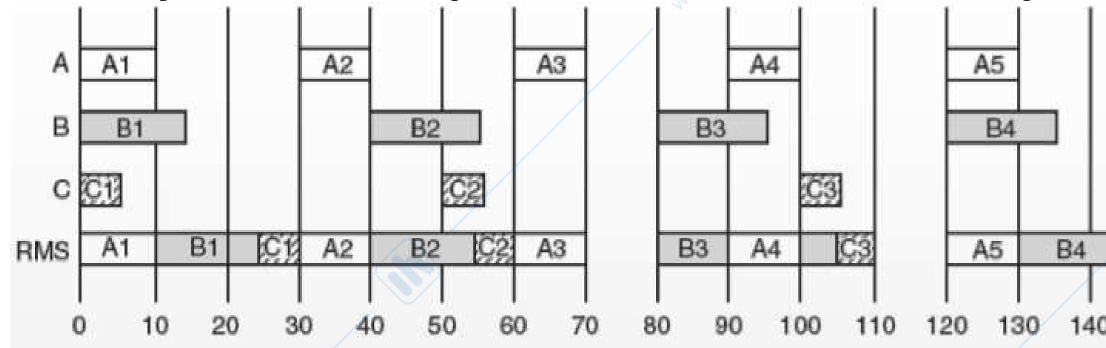
I processi multimediali sono una categoria specifica di processi real-time: sono processi periodici. Come i processi real-time, i processi multimediali ragionano in termini di scadenza temporali (deadline), solo che in questo caso sono ricorrenti e note a priori. Un flusso multimediale è un insieme di più stream, ciopè uno stream audio, video e sottotitoli. Ognuno ha i suoi tempi. Ci sono due algoritmi che cercano di trovare la sequenza giusta

RMS (Rate Monotonic Scheduling)

È una politica di scheduling pensata appositamente per i processi periodici.

1. Il burst di ogni processo deve essere terminato entro lo scadere del suo periodo.
2. Non ci sono dipendenze tra processi
3. I burst richiesti sono sempre gli stesi ad ogni periodo
4. La prelazione della CPU si presuppone abbia un overhead trascurabile
5. I processi non periodici (se ce ne sono) vengono schedulati solo se la CPU non ha altro da fare (non hanno deadline)

RMS è un algoritmo di scheduling a priorità statica. (il periodo è da un blocco all'altro)
Ogni processo riceve una priorità inversamente proporzionale al periodo.
Processi con periodi brevi (ovvero, frequenza di esecuzione alta) verranno schedulati con precedenza.



A= audio, B=video, C=sottotitoli.
Nel momento in cui inizia la riproduzione multimediali con priorità statica, assegno la priorità ai processi.
Ho A,B,C all'istante 0. Inizio con A perché ha priorità più alta. All'istante 10 A ha finito, tra B e C, B ha una priorità più alta perché ha un periodo più breve e in fine si avrà C.

Questo non funziona sempre in quanto RMS non sfrutta la CPU al 100%, RMS funziona se il tasso di utilizzo del sistema è inferiore ad una certa soglia.

$$\sum_{i-1}^{m} \frac{C_i}{P_i} < m(\sqrt[m]{2} - 1)$$

Dove m= numero dei processi e la sommatoria è la condizione di schedulabilità. Inoltre

$$\lim_{m\to\infty} m\left(\sqrt[m]{2} - 1\right) = 0.7\ circa$$

È possibile fare una schedulazione con RMS di processi periodici nel momento in cui l'uso della CPU non superi il 70%. Potrebbe essere o meno una soluzione dipende dall'hardware.

EDS (Earliest Deadline First Scheduling)

EDS è una variante di RMS che prevede priorità dinamiche.
Ogni processo riceve una priorità inversamente proporzionale al tempo rimanente alla sua deadline (cioè la priorità più alta viene assegnata al processo più vicino al dealine)
Con EDS non è più strettamente necessario che i processi siano periodici neppure che abbiamo un burst sempre uguale. (è un'approssimazione del Shortest Job First SJF)
EDS è una politica di schedulazione real-time generica. Anche EDS non riesce a dare una sicurezza massima, ma è migliore del RMS

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

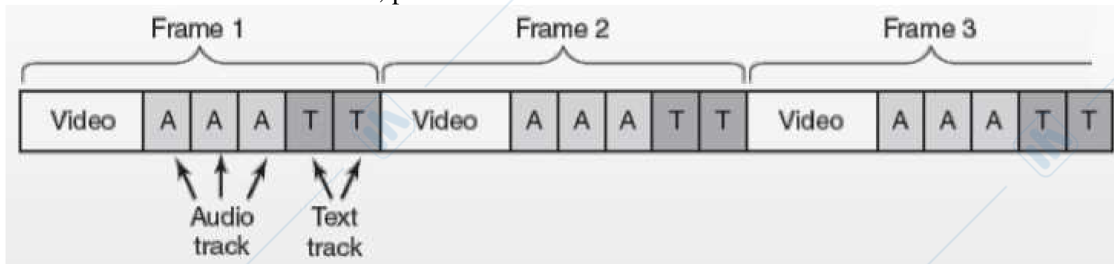
File system con supporto multimediale

I file system con supporto multimediale devono vere principalmente due caratteristiche:

- Tempo di accesso ridotto al minimo.
- Facilità di passare da un punto all'altro della riproduzione

Minimo tempo di accesso

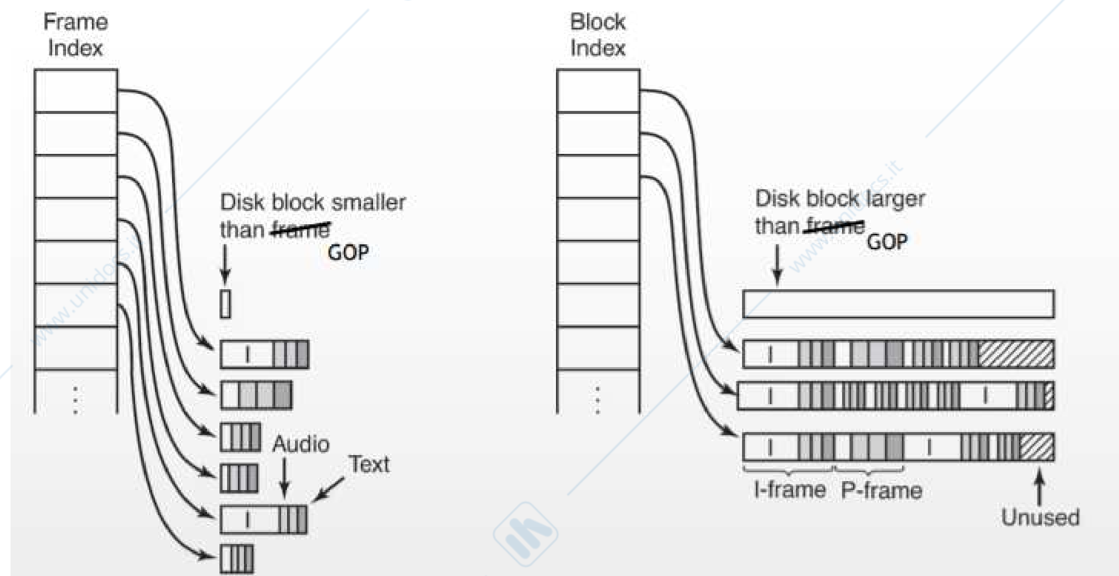
Basta memorizzare tutti i file sul disco in modalità sequenziale, tendo conto della codifica e del container. Ad ogni frame viene associato in modo contiguo la traccia video, audio e in fine i sottotitoli. Sapendo la dimensione massima di un frame, passare da un frame all'altro diventa molto facile.



Manipolazione della riproduzione

Dobbiamo aggiungere al file dei metadati. All'interno dell'i-node vengono aggiunti delle informazioni sul GOP. Allora un datablock potrebbe essere più grande o più piccolo della dimensione di un GOP

- **Dimensione blocco >= dimensione GOP**
  - Uso un "block index", per saltare all'interno del filmato
  - Ho una frammentazione interna
  - Non ho bisongo di gestire allocazioni sequenziali.
- **Dimensione blocco < dimensione GOP**
  - Uso un "frame index", questo è il caso medio. I frame index sono quei frame che fanno rifermento a dei I-Frame. Questi frame dovranno essere sempre i primi frame all'interno di un datablock. Se un datablock non è sufficiente a contenere tutto un GOP si continua nel datablock successivo, ma questo datablock non sarà elencato nel fram index. Tengo traccia di dove sono gli I-frame
  - Devo gestirmi delle allocazioni sequenziali
  - Non ho spreco di spazio



Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

Assistente

Scheduling di Processi

10282 (RR) - nello stesso istante ci sono due processi che possono acquisire la CPU allora scegliamo il primo in ordine alfabetico

39529 (SJFCP) –

Memory Manager

11057 (ottimale) –

Produttore e consumatore

Utilizzando i monitor

I monitor sono costrutti del linguaggio di programmazione, cioè esistono delle keyword di linguaggio che vengono inseriti nel progetto. Vengono utilizzati per sincronizzare diversi thread.

```
1.  /*
2.   * Implementazione con Monitor
3.   */
4.
5.  public class ProducerConsumer {
6.
7.      public static void main(String[] args) {
8.
9.          Buffer buffer = new Buffer();   // Creo buffer condiviso
10.
11.         Producer producer = new Producer(buffer);   // Creo thread produttore, passo al co
    struttore il buffer condiviso
12.         Consumer consumer = new Consumer(buffer);   // Creo thread consumatore, passo al c
    ostruttore il buffer condiviso
13.
14.         producer.start();   // Avvio thread produttore
15.         consumer.start();   // Avvio thread consumatore
16.
17.     }
18.
19. }
```

L'oggetto buffer viene passato come parametro sia al produttore che al consumatore. Questo perché i monitor in java avvengono attraverso un look sull'oggetto, essendo l'oggetto condiviso tra il consumatore e il produttore capiscono chi lo sta utilizzando.

```
1.  public class Producer extends Thread {
2.
3.      private Buffer synchronizedBuffer;
4.
5.      public Producer(Buffer buffer) {
6.          this.synchronizedBuffer = buffer;   // Assegno a variabile privata il buffer condi
    viso
7.      }
8.
9.      @Override
10.     public void run() {
11.         while(true) {   // Thread loop
12.             try {
13.                 synchronizedBuffer.produce();   // Chiamo ripetutamente la funzione produc
    e() del buffer condiviso
14.             } catch (InterruptedException e) {
15.                 e.printStackTrace();    // gestisco l'eccezione lanciata da wait()
16.             }
17.         }
18.     }
19. }
```

Il metodo run() è il codice che in modo concorrente viene eseguito. Il produttore esegue all'infinito il metodo produci del buffer. Il metodo produce() è inserito nel buffer perché su questo metodo viene eseguito un meccanismo di look. Se il metodo produci è sincronizzato, cioè viene usata la keyword synchronized sul

Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

metodo produce() allora se più produttori contemporaneamente tendono ad avviare il metodo produce() si scontreranno con un meccanismo interno di Java che dirà a loro di come accedere alla funzione produci, altrimenti se una funzione di un oggetto non è sincronizzata chiunque voglia può accedere.

```
1.  public class Consumer extends Thread {
2.
3.      private Buffer synchronizedBuffer;
4.
5.      public Consumer(Buffer buffer) {
6.          this.synchronizedBuffer = buffer;   // Assegno a variabile privata il buffer condi
    viso
7.      }
8.
9.      @Override
10.     public void run() {
11.         while(true) {   // Thread loop
12.             try {
13.                 synchronizedBuffer.consume();   // Chiamo ripetutamente la funzione consum
    e() del buffer condiviso
14.             } catch (InterruptedException e) {
15.                 e.printStackTrace();    // gestisco l'eccezione lanciata da wait()
16.             }
17.         }
18.     }
19. }
```

```
1.  import java.util.ArrayList;
2.
3.  public class Buffer {
4.
5.      final static int BUFFER_SIZE = 10;  // grandezza buffer costante
6.
7.      private final ArrayList<Object> list = new ArrayList<Object>(); // Inizializzo ArrayLi
    st per contenere elementi del buffer
8.
9.      // solo un thread alla volta potrà accedere ad una funzione sincronizzata sull'oggetto
     buffer
10.
11.     public synchronized void produce() throws InterruptedException { // funzione del produ
    ttore
12.         if( list.size() < BUFFER_SIZE ) {    // se il buffer non è pieno
13.             list.add(new Object()); // accesso esclusivo al buffer: produco elemento e lo
    aggiungo
14.             this.notify();  // risveglio (porto in stato di pronto) un thread (se disponib
    ile) precedentemente andato in wait su questo oggetto
15.             System.out.println("Object produced. Remaining objects: " + list.size());
16.         } else {
17.             this.wait();    // se il buffer è pieno, rilascio il lock sull'oggetto buffer
    e mando il produttore in wait
18.         }
19.     }
20.
21.     public synchronized void consume() throws InterruptedException {    // funzione del co
    nsumatore
22.         if( list.size() > 0 ) {  // se il buffer non è vuoto
23.             list.remove(0); // accesso esclusivo al buffer: rimuovo elemento
24.             System.out.println("Object consumed. Remaining objects: " + list.size());
25.             this.notify(); // risveglio (porto in stato di pronto) un thread (se disponibi
    le) precedentemente andato in wait su questo oggetto
26.         } else {
27.             this.wait();    // se il buffer è vuoto, rilascio il lock sull'oggetto buffer
    e mando il consumatore in wait
28.         }
29.     }
30.
31. }
```


Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

È presente un solo buffer passato come parametro tra consumatore e produttore, questa classe ha due metodi con la keyword synchronized. Solo un thread alla volta può accedere al corpo della funzione specificata. Se avessi più produttori, sullo stesso buffer, che vogliono accedere contemporaneamente al metodo produci, se ne occuperà la macchina virtuale java, in modo tale che solo un produttore alla volta possa accedere al metodo produci e quindi che riesca ad ottenere accesso esclusivo alla risorsa condivisa. Se un produttore riesce ad accedere al metodo sincronizzato esegue una notify, cioè se il produttore riesce a produrre qualcosa, lancia una notifica ad un altro oggetto che è in attesa della risorsa buffer e lo risveglia dal suo stato di wait. Il metodo consume() rimuove un elemento se la grandezza del buffer è uguale a 0 allora andrà in wait().

Utilizzando i semafori e i mutex

```
1.  /*
2.   * Implementazione con Semafori e Mutex (Semafori e Mutex sono qui implementati utilizzand
    o i monitor)
3.   */
4.
5.  public class ProducerConsumer {
6.
7.      public static void main(String[] args) {
8.
9.          Buffer buffer = new Buffer();   // Creo buffer condiviso
10.
11.         Producer producer = new Producer(buffer);   // Creo thread produttore, passo al co
    struttore il buffer condiviso
12.         Consumer consumer = new Consumer(buffer);   // Creo thread consumatore, passo al c
    ostruttore il buffer condiviso
13.
14.         producer.start();   // Avvio thread produttore
15.         consumer.start();   // Avvio thread consumatore
16.
17.     }
18.
19. }
```

```
1.  public class Producer extends Thread {
2.
3.      private Buffer synchronizedBuffer;
4.
5.      public Producer(Buffer buffer) {
6.          this.synchronizedBuffer = buffer;   // Assegno a variabile privata il buffer condi
    viso
7.      }
8.
9.      @Override
10.     public void run() {
11.         while(true) {   // Thread loop
12.             try {
13.                 synchronizedBuffer.produce();   // Chiamo ripetutamente la funzione produc
    e() del buffer condiviso
14.             } catch (InterruptedException e) {
15.                 e.printStackTrace();    // gestisco l'eccezione lanciata da wait() all'int
    erno della funzione down()
16.             }
17.         }
18.     }
19. }
```


Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

```
1.  public class Consumer extends Thread {
2.  
3.      private Buffer synchronizedBuffer;
4.  
5.      public Consumer(Buffer buffer) {
6.          this.synchronizedBuffer = buffer;   // Assegno a variabile privata il buffer condi
    viso
7.      }
8.  
9.      @Override
10.     public void run() {
11.         while(true) {   // Thread loop
12.             try {
13.                 synchronizedBuffer.consume();   // Chiamo ripetutamente la funzione consum
    e() del buffer condiviso
14.             } catch (InterruptedException e) {
15.                 e.printStackTrace();    // gestisco l'eccezione lanciata da wait() all'int
    erno della funzione down()
16.             }
17.         }
18.     }
19. }
```

```
1.  import java.util.ArrayList;
2.  
3.  public class Buffer {
4.  
5.      final static int BUFFER_SIZE = 10;  // grandezza buffer costante
6.  
7.      private final ArrayList<Object> list = new ArrayList<Object>(); // Inizializzo ArrayLi
    st per contenere elementi del buffer
8.      Semaphore empty = new Semaphore(BUFFER_SIZE);   // Inizializzo semaforo empty con v =
    10
9.      Semaphore full = new Semaphore(0);  // Inizializzo semaforo full con v = 0
10.     Mutex mutex = new Mutex(true);  // Inizializzo mutex con v = 1
11. 
12.     public void produce() throws InterruptedException { // funzione del produttore
13.         empty.down();   // decremento semaforo empty, il thread andrà in wait se il buffer
     è pieno
14.         mutex.down();   // decremento mutex, il thread andrà in wait se un altro thread st
    a già operando sul buffer
15.         list.add(new Object()); // accesso esclusivo al buffer: produco elemento e lo aggi
    ungo
16.         System.out.println("Object produced. Remaining objects: " + list.size());
17.         mutex.up(); // incremento il mutex, verrà risvegliato (stato di pronto) un thread
    (se disponibile) precedentemente mandato in attesa di operare sul buffer
18.         full.up(); // incremento il semaforo full, verrà risvegliato (stato di pronto) un
    thread consumatore (se disponibile) precedentemente mandato in attesa perchè il buffer era
     vuoto
19.     }
20. 
21.     public void consume() throws InterruptedException {
22.         full.down();    // decremento semaforo full, il thread andrà in wait se il buffer
    è vuoto
23.         mutex.down();   // decremento mutex, il thread andrà in wait se un altro thread st
    a già operando sul buffer
24.         list.remove(0); // accesso esclusivo al buffer: rimuovo elemento
25.         System.out.println("Object consumed. Remaining objects: " + list.size());
26.         mutex.up(); // incremento il mutex, verrà risvegliato (stato di pronto) un thread
    (se disponibile) precedentemente mandato in attesa di operare sul buffer
27.         empty.up(); // incremento il semaforo empty, verrà risvegliato (stato di pronto) u
    n thread produttore (se disponibile) precedentemente mandato in attesa perchè il buffer er
    a pieno
28.     }
29. 
30. }
```


Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

In questo caso i metodi consumi e produci, non sono sincronizzati e per questo sarà possibile che più thread contemporaneamente possano accedere all'interno dei metodi. In questo caso all'interno dei metodi vengono utilizzati i semafori e i mutex (i mutex sono dei semafori inizializzati ad 1, possono assumere solamente lo stato 0 o lo stato 1). I mutex essendo un metodo di controllo binario, vengono utilizzati per controllare l'accesso esclusivo della risorsa. I semafori controlla quanti flussi possono accedere contemporaneamente ad esempio, se un semaforo è inizializzato a 5 allora i primi 5 flussi che arrivano possono entrare, dal sesto in poi devono aspettare che qualcuno esca per potervi accedere. Sia i semafori che i mutex hanno principalmente due funzioni UP, cioè incrementa il valore e DOWN, cioè decrementa il valore. Il meccanismo di controllo avviene quando si raggiungono valori critici, cioè quando si esegue un DOWN e il valore è 0, un UP quando il valore è inferiore a 0

Esempi sul codice	
Abbiamo due semafori e un mutex, prendiamo in considerazione un produttore. Il semaforo empty è inizializzato a 10, il semaforo full a 0 e il mutex è impostato ad 1. Il thread produttore esegue empty.down e decrementa il semaforo da 10 a 9. Dato che il valore è maggiore di 0 il thread può continuare la sua esecuzione. L'istruzione mutex.down imposta il mutex a 0 ed essendo non inferiore a 0 può entrare. Riesce ad eseguire le istruzioni aggiungendo l'oggetto al buffer e la println. Mutex.up esce dal mutex riportando il mutex a 1. Full.up incrementa un altro semaforo full ad 1.	Il buffer è vuoto. Prendiamo in considerazione che lo scheduler concede al thread consumatore di accedere alla sua funzione. Ricordiamo di non abbiamo controllo sullo scheduler. Se il consumatore accede al buffer mentre è ancora vuoto. L'istruzione full.down porta un decremento del semaforo che andrà dallo 0 a -1, in questo caso il consumatore si pone in uno stato di wait. Rimarrà addormentato fin quando non verrà rieseguito una up su un semaforo o mutex che possa risvegliare questo flusso. Infatti nel metodo up nella classe mutex è presente un notify().

L'ordine di esecuzione, l'ordine nel produttore e consumatore è differente.

```
1.  /*
2.   * Implementazione Mutex usando i Monitor
3.   *
4.   */
5.
6.
7.  public class Mutex {
8.
9.      private int v;
10.
11.     public Mutex(boolean value) {   // Un mutex si comporta come un semaforo inizializzato
    a 0 o 1
12.         if(value) {     // se mutex è inizializzato con True dal costruttore v è pari a 1,
    altrimenti 0
13.             this.v = 1;
14.         } else {
15.             this.v = 0;
16.         }
17.     }
18.
19.     synchronized void down() throws InterruptedException {
20.         v--;    // decremento il valore di v
21.         if(v < 0) {
22.             this.wait();    // se v è minore di zero dopo il decremento, rilascio il lock
   sull'oggetto semaforo e mando il thread chiamante in wait
23.         }
24.     }
25.
26.     synchronized void up() {
27.         v++;    // incremento il valore di v
28.         this.notify();  // risveglio (porto in stato di pronto) un thread (se disponibile)
    precedentemente andato in wait su questo oggetto
29.     }
30.
31. }
```


Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

```
1.  /*
2.   * Implementazione Semaforo usando i Monitor
3.   */
4.
5.  public class Semaphore {
6.
7.      private int v;
8.
9.      public Semaphore(int value) {
10.         this.v = value; // Inizializzo il semaforo con il valore passato dal costruttore
11.     }
12.
13.     synchronized void down() throws InterruptedException {
14.         v--;    // decremento il valore di v
15.         if(v < 0) {
16.             this.wait();    // se v è minore di zero dopo il decremento, rilascio il lock
    sull'oggetto semaforo e mando il thread chiamante in wait
17.         }
18.     }
19.
20.     synchronized void up() {
21.         v++;    // incremento il valore di v
22.         this.notify();  // risveglio (porto in stato di pronto) un thread (se disponibile)
     precedentemente andato in wait su questo oggetto
23.     }
24.
25. }
```


Document shared on www.docsity.com
Downloaded by: massimiliano-demiri (epiphone12321@gmail.com)
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari