1°SEMESTRE

SISTEMI OPERATIVI

Creato da : Andrea Possemato
Amanuense : Tommaso Furlan

TERMINAZIONE DI UN PROCESSO

Exit/Kill : -Un processo termina con una richiesta al sistema operativo (exit)
-Un processo può terminare per effetto di un altro processo (Kill) , in modo controllato rispetto a privilegi e protezioni
-Un processo può terminare per un problema (Errore)

Quando un processo termina , causa la conclusione delle operazioni di I/O bufferizzate , il rilascio delle risorse impegnate (memoria , dispositivi i/o dedicati) ee distruzione del descrittore.

TERMINAZIONE PADRE-FIGLIO

1. La terminazione del processo FIGLIO risveglia il processo PADRE in attesa
2. La terminazione del processo Padre invece può causare : -terminazione processi figli
   -processi orfani (adottati da init)
3. Per i processi DETACHED , non ha influenza la terminazione del processo che li ha creati

CONTEXT SWITCH

La commutazione di contesto è un particolare stato del SO durante il quale avviene il cambiamento del processo correntemente in esecuzione su una delle CPU. Questo permette a più processi di condividere una stessa CPU , ed è utile sia nei sistemi che adoperano un solo processore poiché permette di eseguire più programmi contemporaneamente , sia nell'ambito del calcolo parallelo poiché permette un migliore bilanciamento del carico

Fasi Context Switch : Prima di tutto è necessario salvare lo stato della computazione del processo correntemente in esecuzione su una delle CPU , tra cui il PROGRAM COUNTER , il Contenuto dei Registri , in modo da poter riprendere l'esecuzione anche in seguito. In pratica viene salvato il PCB (Process Control Block).
In seconda luogo, lo scheduler sceglierà un processo dalla coda dei processi pronti in base alla priorità politica di scheduling , e accederà al suo pcb per ripristinare il suo stato nel processore in maniera inversa rispetto alla fase precedente.
Il context Switch avviene in 4 fasi:

1. Inizialmente il processo P1 è in esecuzione , quindi il processore opera nel contesto del processo P1 : lo stack contiene dati locali di P1 , mentre il descrittore di P1 non è significativo. Il descrittore di P2 contiene il contesto di P2 salvato quando P2 ha interrotto l'esecuzione
2. P1 esegue una SVC per richiedere un operazione di I/O. Il suo contesto viene esposto in cima allo stack e il processore opera nel contesto del nucleo
3. Il nucleo porta P1 in stallo di attesa , e P2 in stallo di esecuzione , salvando il contesto di P1 nel descrittore di P1 , e recuperando dal decrittore di P2 il contesto di P2
4. Il nucleo termina la SVC eseguendo un ritorno da interruzione che ripristina il contesto del processore con il contenuto dello stack . Il processore opera nel contesto di P2

STRUTTURE DATI SO

Le strutture dati del sistema operativo mantengono informazioni sullo stato corrente del sistema in termini di processi e risorse

1. **Tabella dei Processi** = Pid allocazione di memoria , file aperti e utilizzati , informazioni di stato , informazioni contabili , programma eseguito

2. **Tabella di Allocazione di Memoria** = Allocazione della memoria centrale e secondaria ai processi , informazoni necessarie per la gestione della memoria virtuale , attributi di protezione per l'accesso ad aree di memoria condivisa

3. T**abella dispositivi I/O=** Stati dei dispositivi di i/o (occupato , assegnato , occupato esclusivamente ad un processo) stato delle operazioni di i/o , informazioni sui buffer utilizzati per il trasferimento dei dati da/a la periferica

4. **Tabella File Aperti** = identificazione dei file , locazione della memoria secondaria , stato corrente di accesso/condivisione/posizione di lettura/scrittura. Informazioni che possono essere gestite attraverso il FILE SYSTEM

GESTIONE PROCESSI IN UNIX

La creazione di un processo in Unix può avvenire in due modi

- **Fork** = si duplica un processo già esistente che dà origine ad un processo figlio , copia del processo creatore detto processo Padre
- **Exec** = viene sostituita l'immagine eseguita , e questo permette ad uno dei due processi di evolvere sepratamente dall'altro

FORK

Una fork avrà un area dati duplicata , un area codice condivisa , con tutte le risorse utilizzate dal processo creante accessibili dal processo creato

- Processo Padre : esito= Fork() riceve esito>0 e uguale al Pid del figlio
- Processo Figlio : esito = Fork() riceve esito=0
- Fallimento/errore : esito= Fork () riceve esito<0

RELAZIONE TRA PROCESSI

Un processo figlio non condivide memoria con un processo padre (condivide solo il codice) , infatti dalla creazione in poi i due processi evolvono separatamente eseguendo la stessa immagine in modo indipendente.
La creazione come detto prima avviene per duplicazione completa (Logica) del processo padre , con il processo figlio che eredita :

- ambiente di lavoro (legato al processo e non all'immagine , quindi sopravvive alla funzione Exec , e viene trasferita al nuovo programma)
- file aperti
- privilegi
- directory di lavoro

TERMINAZIONE PROCESSI UNIX

La terminazione consiste in una serie di processi che lasciano il sistema in stato corrente , cioè vengono chiusi i file aperti , rimossa l'immagine dalla memoria ed eventualmente viene segnalata la chiusura del processo figlio al padre

- **Exit** = questa funzione termina l'esecuzione di un processo , segnalando al processo che l'ha creato un valore numerico che rappresenta l'esito (sintetico) dell'esecuzione [exit(stato)]

- **Wait** = mette un processo in attesa della terminazione di un processo figlio. Il simbolo ! restituisce l'identificativo del processo terminato e il suo stato d'esecuzione [Pid=wait(&status)]

SISTEMI A PROCESSI

Il sistema unix introduce un buon livello di flessibilità attraverso il concetto di FILE , che rappresenta la virtualizzazione di un generico dispositivo in grado di produrre e ricevere dati in sequenza .
Ci sono certe operazioni "limitate" che si possono attuare solo su alcuni file : non ha senso leggere da una stampante o scrivere su una tastiera , infatti è permesso :

- SOLO LETTURA da tastiera
- SOLO SCRITTURA su una stampante

FILE

Un file UNIX è una sequenza di Byte identificabili come una unità contenete informazioni.

- In unix non c'è distinzione tra file di testo e file binari , e la struttura logica di un file è impostata dai programmi applicativi
- Ogni file è identificato da un nome simbolico che lo distingue in modo univoco nel sistema : questo nome simbolico è il PATH NAME
- il suffisso del file ne caratterizza il contenuto permettendo all'utente di derivare alcune proprietà da nome

ELABORAZIONE FILE

I file sono elaborati attraverso 5 classi di operazioni corrispondenti a chiamate al SO

1. Open , Read , Write , Close = utilizzo file esistente
2. Creat = creazione nuovo file
3. Dup = manipolazione dell'associazione tra un file e la sua identificazione in un programma
4. Chmod , Chown , Stat , … = definizione delle proprietà del file
5. Mv , Ln , Mount , Amount = elaborazione della struttura organizzativa

CREAZIONE FILE **[Fd= Creat(Pathname , mode)]**

- Pathname = è il nome simbolico del file
- Mode = è una composizione simbolica di flag (Var con solo due stati : vero/falso – On/off – Yes/no) che identificano i permessi attribuiti agli utenti (read , write , execution)
- Fd = è un numero che viene associato al file per la sua identificazione nel processo (file descriptor)

APERTURA FILE **[Fd=Open(Patchname , flags)]**

- Flags = è una composizione di indicatori binari che definiscono le operazioni permesse e le modaità della loro esecuzione (lettura e/o scrittura)

Prima di poter accedere ad un file esistente , un processo deve aprirlo , cioè renderlo disponibile all'uso verificandone le proprietà e associandolo al processo

CHIUSURA FILE **[Close(Fd)]**

- La chiusura di un file determinano una serie di operazioni che garantiscono la correttezza del sistema rispetto al file ,e viceversa.
- Vengono eseguite le operazioni di I/O bufferizzate e non completate ;
- vengono liberate le risorse occupate dal processo per la gestione del file (buffer I/O) ;
- viene liberato il corrispondente descrittore nella tabella dei file aperti del processo ;

Attenzione = Dal momento che un file può essere condiviso , occorre verificare nella tabella globale dei file , se altri processi stiano utilizzando quel file

LETTURA/SCRITTURA FILE **[readcount=read(Fd , Buffer , Nbyte) o write(Fd , Buffer , Nbyte)]**

- Fd = numero descrittore file
- Buffer = è un inditizzo dell'area dati del processo dove verranno depositati i dati letti/scritti
- Nbyte = è il numero dei byte da leggere/scrivere
- Readcount = numero di byte EFFETTIVAMENTE letti (<=nbyte , < se Eof o Errore)
  o
- Writecount = numero di byte EFFETTIVAMENTE scritti

ERRORI CON I FILE

- creazione (permessi)
- apertura (permessi , file non esistente)
- lettura (permessi , errori dispositivo , eof)
- scrittura (permessi , dispositivo saturo , errori dispositivo)

TIPOLOGIE FILE STANDARD

- Input = associato al descrittore 0 , normalmente è il terminale (tastiera) da cui viene eseguito il processo
- Output = associato al descrittore 1 , normalmente il terminale (video) da cui viene eseguito il processo
- Errore = associato al descrittore e , normalmente il terminale (video) da cui viene eseguito il processore

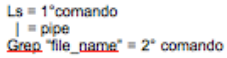
Tutti i processi in Unix hanno accesso a questi tre file predefiniti , infatti la shell in Unix definisce una convenzione per dirottare questi 3 file standard in ingresso/uscita verso altri dispositivi o file

- Stdin = è dirottato sul file che esegue il carattere ' < '
- Stdout = è dirottato sul file che esegue il carattere ' > '
- Stderr = è dirottato sul file che esegue il carattere ' >2 '

PIPE & PIPELINE

Le Pipe e Pipeline sono strumenti sintattici di una shell che permettono di comporre più programmi in sequenza collegando l'output di ciascuno all'input del programma successivo.
I programmi realizzano un elaborazione a più stadi (pipeline) in cui ogni programma rappresenta uno stadio , e i dati fluiscono da uno stadio all'altro in modo sequenziale e sincronizzato

Ls | grep “file_name” => Pipe



PROCESSI e ATTIVITA' CONCORRENTI

Un insieme di processi concorrenti può condividere una parte dei dati e può sincronizzarsi in momenti selezionati dell'esecuzione. Il sistema operativo attraverso lo SCHEDULER e ai meccanismi CONTEXT SWITCH , gestisce l'assegnazione della CPU ai processi.
Il processo può essere visto come un unità di allocazione di risorse (memoria virtuale per immagine del processo , controllo su altre risorse esterne (I/O , file,...)) oppure come un unità di esecuzione (dispatching) , quest'unità identifica un flusso di esecuzione attraverso uno o più programmi. L'esecuzione può essere intervallata/sincronizzata con quella di altri processi , infatti un processo ha uno stato di esecuzione e alcuni attributi che ne determinano le modalità di esecuzione (proprietà , …).
Queste due unità possono essere gestite in modo indipendente , tant'è che questa separazione porta a due concetti distinti :

- l'unità d'allocazione delle risorse è identificata dal concetto di processo
- l'unità d'esecuzione è identificata dal concetto di THREAD (o Lightweight Process)

Il concetto di thread introduce nel progetto di SO una struttura di esecuzione più articolata basata su :

- Condivisione ragionata di risorse
- differenziazione di più flussi d'esecuzione all'interno di un unico processo

THREAD

è un' unità di impiego di CPU all'interno di un processo , il quale può contenere più thread , ciascuno dei quali evolve in modo logicamente separato dagli altri thread

DOVE VENGONO USATI I THREAD

- programmi interattivi che aprono finestre multiple
- server di attività concorrenti e indipendenti (web server)
- sistemi che elaborano eventi e segnali provenienti da fonti diverse , secondo tempistiche non prevedibili

STATI DI ESECUZIONE

1. attivo
2. in attesa
3. pronto
4. terminato

Non esiste lo stato “suspendend” , poiché sia la sospensione che la terminazione di un processo , terminano e sospendono tutti i suoi thread

Ogni Thread è caratterizzato da uno stato di esecuzione , cioè è presente un Program Counter (PC) ,un insieme di Registri , ed uno Stack per i dati locali

CONDIVISIONE
Un thread condivide con altri Thread dello stesso processo il codice , i dati globali e le risorse dell'ambiente esterno. Ogni thread viene eseguito in maniera indipendente (logicamente) dagli altri thread

THREAD vs PROCESSI

La gestione del Thread è più veloce rispetto a quello dei processi , infatti sono più veloci le operazioni di creazione , terminazione e context switch. Per quest'ultimo , viene eliminato (solo per i thread utente) in quanto non è richiesto l'ausilio del So.

Thread Utente VS Thread Kernel

Utente :

- Gestione a carico dell'applicazione
- kernel non ha cognizione dei thread e non li gestisce
- Realizzato attraverso chiamate a funzioni di libreria

Kernel :

- il kernel gestisce le informazioni sul contesto dei processi e dei thread
- lo scheduling delle attività si basa sui thread e non li gestisce

THREAD UTENTE

Vantaggi :

- la commutazione tra le thread non richiede l'intervento del kernel
- lo scheduling dei processi è indispendente da quello dei thread
- lo scheduling può essere ottimizzato per la specifica applicazione
- possono essere implementati su qualsiasi SO attraverso una libreria

Problemi :

- la maggior parte dei system call sono bloccanti , il blocco del processo causa il blocco di tutti i suoi thread
- nei sistemi multiprocessor , il kernel non può assegnare due thread a due processi diversi

THREAD SISTEMA

Vantaggi :

- in un sistema multiprocessor , il kernel può assegnare più thread dello stesso processo a processori diversi
- la sospensione e l'esecuzione delle attività sono eseguite a livello thread

Svantaggi :

- la commutazione di thread all'interno di un processo costa quanto la commutazione di processo
- cade uno dei vantaggi dell'uso dei thread rispetto ai processi

ARCHITETTURE A MICROKERNEL

Il sistema operativo è formato da un piccolo nucleo che contiene le funzioni come sottosistemi esterni :

- device driver
- file system
- gestore della memoria virtuale
- sistema di windowing e interfaccia utente
- sistemi per la sicurezza

Vantaggi :

- interfaccia uniforme delle richieste di servizio da parte dei processi (tutti i servizi sono forniti attraverso lo scambio di messaggi)
- estendibilità , flessibilità , portabilità (è possibile aggiungere , eliminare , riconfigurare nuovi servizi senza "toccare" il kernel)
- Affidabilità (object oriented=progetto modulare ; Verificabilità = kernel limitato nelle dimensioni e nelle funzioni)
- Supporto per i sistemi distribuiti (lo scambio di messaggi è indipendente dall'organizzazione reciproca di mittente e destinatario)

GESTIONE UNITA' CENTRALE

ALGORITMI DI SCHEDULING (Obiettivi generali)

- Fairness = assicurare ad ogni processo una equa ripartizione di risorse
- policy enfocement (rispetto delle politiche) = assicurare che l'algoritmo rispetti le politiche che si vogliono perseguire
- Bilanciamento = assicurare l'utilizzo di tutte le risorse di sistema

Per **sistemi Batch** (insieme di comandi o programmi, tipicamente non interattivi, aggregati per l'esecuzione) :

- Throughput = massimizzare il numero di job eseguiti
- Tempo Turnaround = minimizzare il tempo fra la sottomissione di un lavoro e la sua esecuzione

Per **sistemi Interattivi** o TimeSharing (risposta del terminale dopo poco) :

- tempo di Risposta = risponde prontamente alle richieste di servizio
- Proporzionalità = soddisfare le attese dell'utente

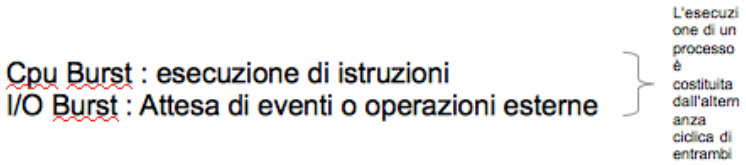
Per **sistemi Real-Time** (risposta del terminale immediata) :

- rispetto delle scadenze = evitare di perdere dati e/o eventi
- predicibilità = prevedere il comportamento del sistema sotto le condizioni di carico previste per evitare il degrado delle prestazioni

SCHEDULING DELL'UNITA' CENTRALE

In un sistema multiprogrammato l'esecuzione passa da un processo all'altro seguendo opportune strategie di gestione delle risorse
La scelta di quale processo selezionare per l'esecuzione , determina le politiche di scheduling che tengono conto di fattori come l'utilizzo del processore , il tempo di risposta e il throughput

CPU – I/O



LIVELLI DI SCHEDULING

A lungo termine (ad alto livello)
- quale processo è ammesso a competere per le risorse
- quale processo inizia l'esecuzione tra quelli che la richiedono

A medio termine (a Livello Intermedio) :
- quale processo è ammesso a competere per l'uso del processore
- quale processo resta in memoria o viene portato fuori memoria durante l'attesa o lo stato di pronto

A breve Termine (a Basso Livello) :
- come vengono assegnate le priorità
- quale processo pronto viene scelto per l'esecuzione

ANALISI LIVELLI

Lungo Termine :
- Determinare quali programmi vengono ammessi all'esecuzione
- Controlla il livello della multiprogrammazione
- Si occupa dell'ammissione di molti processi :
    - Riduce probabilità che tutti siano in stato d'attesa
    - Utilizza complessivamente meglio l'unità centrale
- Difetto = ogni processo ha a disposizione una minore percentuale di tempo di cpu (può bilanciare la presenza di cpu e I/O Bound)

Medio Termine :
- Gestisce la permanenza in memoria dei processi correntemente non in esecuzione
- Soddisfa le esigenze della multiprogrammazione
- Considera i tempi presumibili dei attesa (Attesa su evento Lento)
- gestisce le priorità e le esigenze real-time
- si appoggia al gestore della memoria centrale per l'esecuzione delle operazioni di Swap

Basso Termine :
- Decide quale processo pronto và in esecuzione :
    - scheduler di cpu
    - dispatcher
- Viene invocato su eventi che possono causare il passaggio ad un altro processo (politiche di scheduling) :
    - Interrupt di clock
    - Interrupt di I/O

GESTIONE DELLE PRIORITA'

La gestione delle priorità è gestita/implementata da un sistema di code multiple che ne rappresentano i diversi livelli

- lo scheduler sceglie i processi da eseguire esaminando le code nell'ordine
- i processi a priorità più bassa possono attendere tempo indefinito(starvation)
- le priorità possono essere modificate esaminando la storia passata di un processo per evitare fenomeni di starvation

Gli algoritmi di scheduling si differiscono anche per come gestiscono le priorità

PRE-EMPTION

La sospensione forzata di un processo per eseguirne un altro è detto pre-emption

INTERVENTI SCHEDULER (breve termine)

1. Processo in stato di ESECUZIONE ad ATTESA = quale altro processo pronto andrà in esecuzione?
2. Processo in stato di ATTESA a PRONTO = quale altro processo pronto andrà in esecuzione?
3. Processo in stato di ESECUZIONE a PRONTO = può accedere per un interruzione . Può tornare in esecuzione subito o no?
4. Processo TERMINA = quale processo verrà eseguito?

Casi 1 e 4 : si deve scegliere un altro processso pronto. Queste due politiche di scheduling si dicono Non-PreEmptive

Casi 2 e 3 : si può scegliere un altro processo pronto , oppure si può lasciare o mandare. Queste politiche di scheduling si dicono Pre-Emptive

POLITICHE GESTIONE DELL'UNITA' CENTRALE

Senza Priorità :

- Considerano i processi equivalenti sul piano d'urgenza di esecuzione
- si basano su strategie di ordinamento first come – first served (fifo)

Con priorità :

- dividono i processi in classi secondo l'importanza o la criticità rispetto al tempo di esecuzione
- necessarie nei sistemi con priorità real-time e nei sistemi interattivi

Statiche :

- Un processo conserva nel tempo i suoi diritti di accesso all'unità centrale
- Penalizza i processi a bassa priorità (Starvation)

Dinamiche :

- i processi modificano i propri diritti nel tempo basandosi su comportamento passato o estrapolando quello futuro
- Bilancia le esigenze tra processi di Cpu Bound e I/O Bound

INDICATORI PRESTAZIONI

La misura di prestazione di un sistema di calcolo può essere effettuato tramite diversi indicatori e ha come scopo l'ottimizzazione della reputazione delle risorse , in particolare l'utilizzo di unità di elaborazione

- **Attività di Cpu** : Rapporto fra tempo trascorso nella esecuzione dei processi utente e il tempo trascorso nelle funzioni di SO in attesa

- **Livello di Multiprogrammazione** : Numero di programmi contemporaneamente presenti in memoria centrale

- **Tempo di Attesa** : tempo trascorso tra la richiesta di esecuzione e l'effettivo inizio dell'esecuzione (Tempo trascorso da un processo nello stato di pronto)

- **Tempo di TurnAround(Batch)** : tempo trascorso tra l'ingresso di un programma nel sistema e la fine dell'elaborazione ; il tempo medio è la media dei tempi dei singoli programmi

- **Tempo di Risposta(Interattivo)** : tempo trascorso tra l'immissione di un comando e l'emissione della risposta

- **Throughput(Produttività)** : lavoro svolto dal sistema operativo nell'unità di tempo (numero di processi completati)

- **Fairness** : misura dell'omogeneità di trattamento tra processi diversi

FIRST COME FIRST SERVED (FIFO)

Esegue i processi nell'ordine in cui si trovano nella coda dei processi pronti , il prossimo processo servito è quello più vecchio
Non pre-emptive , con/senza priorità , statico : è algoritmo di base dei sistemi batch
Un problema potrebbe essere che un processo che non esegue operazioni di I/O ha il monopolio della CPU.
Ecco perchè il FCFS è un algoritmo che privilegia i processi cpu bound , cioè un processo i7o bound deve attendere che un processo cpu bound finisca l'esecuzione di un lungo cpu burst. Un processo i/o bound può attendere (stato pronto) anche quando l'operazione di i7o è terminato. Il risultato è inefficiente per l'utilizzo dei dispositivo di i/o (pause tra due richieste di operazioni di i/o)

ROUND ROBIN

Assegna ad ogni processo una quantità di tempo , scaduto il quale il processo è interrotto , rimesso in coda e l'unità viene assegnata ad un altro processo

- garantisce che tutti i processi avanzino in modo equilibrato
- pre-emptive , con o senza priorità , statico o dinamico
- algoritmo di base dei sistemi time-sharing

Criticità : la scelta del quanto tempo è molto complessa , infatti deve essere maggiore del tempo necessario a fare lo scheduling. Inoltre deve essere maggiore del tempo medio necessario per eseguire un CPU burst nei processi I/O Bound

ROUND ROBIN e PROCESSI CPU BOUND

Il round robin privilegia i processi con cpu bound

- un processo i/o bound utilizza spesso l'unità centrale per un tempo minore di quanto assegnato , quindi non va in stato d'attesa
- un processo cpu bound utilizza l'intero tempo assegnato , quindi va in stato di pronto passando così davanti ai processi in attesa

Le prestazioni dei processi I/O , possono migliorare assegnando priorità più alte (RR Virtuale) : un processo che completa un operazione di I/O viene posto in una coda diversa , prioritaria rispetto alla coda dei processi pronti

PREDIZIONE DEL COMPORTAMENTO

La predizione del comportamento futuro si basa sulla valutazione media dei comportamenti passati

$$\bar{S}_{n+1} = (1/n) \sum_{i=1}^{n} T_i = (1/n)Tn + ((n-1)/n)\, Sn$$

- Ti = tempo misurato dall'i-esimo CPU burst
- Si = tempo predetto dell'insieme i-esimo CPU burst

Questa soluzione calcola una media uniforme in cui la storia passata ha lo stesso peso di quella recente.

Problemi :

- non approssima bene la realtà
- il comportamento recente è più influente di quello passato

[Soluzione] MEDIA ESPONENZIALE

$S_{n+1} = \partial Tn + (1-\partial)Sn$ Se $\partial < 1/n$ i cpu burst recenti hanno maggior peso

La media esponenziale è più adeguata per calcolare una stima attendibile.

- Questo Algoritmo implementa implicitamente una politica a priorità , cioè i processi cpu bound hanno un servizio minore rispetto ai processi i/o bound

- I processi con cpu burst più lunghi sono sogetti a starvation , infatti la presenza di molti processi interattivi mantiene basso il tempo d'esecuzione predetto

- La mancanza di pre-emption non è adeguato per sistemi time-sharing , poiché i processi cpu bound hanno priorità più bassa ma possono monopolizzare il sistema dopo un assenza di processi interattivi

SHORTEST PROCESS NEXT (SPN)

Esegue ad ogni turno il processo che prevedibilmente utilizzerà l'unità centrale per il tempo minore , prima di sospendersi o terminare

- la valutazione del comportamento futuro è proiettato dal comportamento passato
- non pre-emptive , con o senza priorità , dinamica
- privilegia processi I/O Bound

SHORTEST REMAINING TIME (SRT)

E' la versione pre-emptive dello shortest process next

- Un processo pronto con un CPU burst stimato più breve del processo in esecuzione , può interromperlo
- compensa il tempo medio di attesa con uno con maggiore complessità di scheduling dovuta alla pre-emption

SPN e **SRT** possono applicarsi all'interno di job anziché che al prossimo cpu burst , solo per lavori batch e poco interattivi

HIGHEST RESPONSE RATIO NEXT (HRRN)

E' una correzione della politica SPN che considera non solo la stima del prossimo CPU burst , ma anche il tempo d'attesa in stato pronto

- un processo con un cpu burst stimato breve ha ALTA priorità
- Un processo con un tempo d'attesa lungo ha ALTA priorità

$$\text{PRIORITA'} = \frac{(\text{Tempo Attesa} + \text{Tempo Esecuzione})}{\text{Tempo Esecuzione}}$$

REAL TIME CON PRIORITA'

- I processi vengono divisi in CLASSI di PRIORITA' , cioè un processo in esecuzione usa l'unità centrale fino a che non si sospende da sola , a meno che non esista un processo pronto più prioritario che assume la precedenza
- Pre-emptive su priorità , statico o dinamico
- E' alla base dei sistemi Hard Real-Time
- Solitamente è combinato con algoritmi
- Le priorità sono divise in classi disgiunte per evitare il blocco dei processi critici nel So

A PRIORITA' VARIABILE

Viene combinato l'algoritmo round robin con una gestione dinamica delle priorità dei processi , questo è noto come Multilevel Feedback.
I processi pronti sono posti in una coda (Ready Queue) con priorità differente , lo scheduler infatti sceglie i processi pronti in ordine di priorità. I nuovi processi sono posti nella coda a priorità massima. Si otterrà così uno scheduling pre-emptive con priorità dinamiche

Le priorità sono modificate in funzione del comportamento dei processi :

- se un processo viene interrotto perchè scade il suo quanto di tempo , passa su una coda con priorità minore
- Se un processo si sospende prima dello scadere di un quanto assegnato , passa allora ad una coda con priorità maggiore
- Favorisce i processi I/O Bound

I processi con cpu burst più lunghi sono soggetti a starvation , infatti la modifica di priorità avviene anche se un processo resta a lungo nello stato di pronto
Se il quanto di tempo è fisso , i processi con cpu burst più lunghi avanzano lentamente , poiché il quanto di tempo può essere modificato in funzione del livello di priorità

$$T(rq_i) = 2^{i-1} \quad \} \text{ modifica del quanto in tempo}$$

FAIR SHARE SCHEDULING

- In un sistema multiutente , ogni utente può eseguire più processi , infatti gli utenti avranno diritti diversi nell'utilizzo della risorsa
- Utenti più attivi ,anche se meno importanti , possono monopolizzare la macchina anche a scapito di utenti (momentaneamente) meno esigenti
- Le risorse libere possono essere distribuite tenendo conto delle risorse già allocate ai diversi processi di un utente

- Gli utenti possono essere a loro volta riorganizzati in gruppi , infatti l'allocazione viene decisa in base alle esigenze del gruppo e non del singolo processo
- Ogni gruppo di processi (utente , gruppo d'utenti) ha diritto ad un utilizzo equo della cpu (fair share) :
    - La cpu è allocata ai processi considerando la loro appartenenza ai gruppi utenti
    - I gruppi più numerosi ricevono meno risorse per ogni processo

Sistemi Unix
In un sistema classico Unix , le risorse sono allocate globalmente e suddivise tra i processi in base alle loro caratteristiche individuali
Sistemi F.S.S
In un sistema Fair Share Scheduling , le risorse sono suddivise a priori tra gruppi di processi e successivamente allocate su base individuale

GESTIONE DELLA MEMORIA

Virtualizzazione = Permette a più processi di comportarsi come se ciascuno avesse una propria unità di elaborazione dedicata , senza la necessità di gestire problemi di competizione e notazione

GESTIONE MEMORIA CENTRALE

- **Rilocazione** = La posizione di un processo in memoria è nota solo al momento dell'esecuzione , e può cambiare nel tempo. Più processi possono alternarsi in memoria in funzione del loro stato di esecuzione

- **Protezione** = un processo non può riferirsi a locazioni di memoria non sue , cioè la protezione può essere esercitata solo al momento dell'esecuzione

- **Condivisione** = deve essere possibile condividere risorse comuni , e deve essere consentito a più processi di utilizzare dati comuni in modo controllato

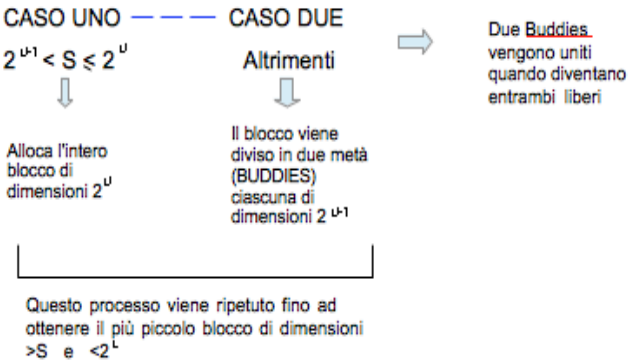
BUDDY SYSTEM

Viene usato in forma modificata da Unix Svr4 per l'allocazione della memoria kernel. La memoria verrà allocata in blocchi di dimensione 2^K

$$L \leq K \leq U \quad \text{Dove} \quad 2^L = \text{Blocco di dimensioni più piccola allocabile}, \quad 2^U = \text{Blocco di dimensione più grande allocabile}$$

Il buddy system è un sistema di allocazione di quantità variabile di dati all'interno di uno spazio di memoria prefissato

Funzionamento : Blocco iniziale (unico) di dimensione 2^U , devo allocare nel'area di dimensioni S



GERARCHIE DI MEMORIA

Le memorie fisiche vengono organizzate in una gerarchia dove variano in modo inverso capacità e velocità di accesso

- **Swapping** : Un solo processo alla volta risiede in memoria , gli altri risiedono sul disco e sono caricati per l'esecuzione

- **Overlay** : Parti diverse della memoria di un processo che non sono utilizzate insieme sono residenti su disco e “sovrapposte” nello stesso spazio di indirizzi quando vengono caricate in memoria centrale

MEMORIA VIRTUALE

La presenza di regole e dispositivi di corrispondenza tra spazi di indirizzamento logico e fisico porta al concetto di memoria virtuale. Un processo vede una memoria logica indipendente dalle reali caratteristiche fisiche
La corrispondenza tra gli indirizzi logici e fisici è attuata da meccanismi Hw/Sw durante l'esecuzione di un processo.
Tabelle di corrispondenza : la contiguità dello spazio di indirizzi è virtuale e realizzata attraverso funzioni e tabelle di corrispondenza .
Un processo può vedere una memoria contigua anche se l'allocazione fisica è sparsa , inoltre scompaiono e diventano meno importanti problemi di frammentazione derivati dalla dinamicità dei processi .

PAGINAZIONE vs SEGMENTAZIONE

Paginazione = meccanismo di gestione efficiente della memoria , indipendente dalla struttura logica del programma/processo ed è trasparente al programmatore e all'ambiente di sviluppo

- Dimensione pagine = influenza la complessità costruttiva e l'efficacia , infatti utilizzando pagine di grandi dimensioni si incorre in uno spreco di spazio , viceversa utilizzando pagne di dimensioni piccole si incorre in una tabella delle pagine molto grande

Segmentazione = può essere visibile e controllabile a livello di codice sorgente ; La segmentazione permette di strutturare i programmi per l'esecuzione. Ci può essere una limitazione per la dimensione dei segmenti , ai segmenti però può essere associato un attributo di protezione e condivisione

PRINCIPIO LOCALITA'

- I riferimenti alla memoria tendono a restare confinati in aree localizzate
- Nel tempo la localizzazione cambia ma il fenomeno persiste

Questo principio , è un principio osservato sperimentalmente che ha basi teoriche nelle caratteristiche degli algortimi e nelle strutture dati aggregate

PAGINAZIONE ON DEMAND

La gestione della memoria a pagine maschera al processo l'effettiva posizione in memoria , a patto di accettare un rallentamento dell'esecuzione , può anche mascherare la presenza continua in memoria di dati e istruzioni

Esempio :

1. L'istruzione utilizza un dato alla locazione logica "X"

2. L'Hw di paginazione mappa l'indirizzo logico "X" nell'indirizzo fisico "Y"
   nb: se "Y" è un indirizzo non valido perchè la pagina che lo contiene
   non è presente in memoria , si genera un page Fault

3. La pagina viene portata in memoria e il processo continua l'esecuzione

4. Un processo per iniziare l'esecuzione ha bisogno che sia presente solo la pagina contenente la sua prima istruzione nella memoria fisica

TRASHING

Fenomeno in cui il processore passa più tempo a caricare/scaricare memoria che ad eseguire istruzioni utili. Questo fenomeno è dovuto al fatto che per contenere più processi solo una parte dello spazio d'indirizzamento di ognuno (processo) è contenuto/conservato in memoria centrale.
Quando la memoria si riempie il sistema deve portare alcune aree fuori memoria per far spazio ad altre aree (Pagine o Process) , effettuando un fenomeno simile al Swap-in /Swap-out.
Questo comporta dei problemi poiché la scelte delle aree da scaricare è critica , infatti un area appena scaricata non dovrebbe essere richiesta subito dopo , onde evitare il trasching.

DIMENSIONE DELLE PAGINE

La dimensione delle pagine è definita dall'architettura della macchina (2^n) , ci si porrà molti interrogativi quando si deve decidere la dimensione specifiche

- Pagine GRANDI = meno pagine per processo ; la Page Table richiede meno memoria virtuale ; trasferimento da/a disco più efficiente ; meno pagine in memoria quindi maggior Hit Ratio nelle cache della TLB

- Pagine PICCOLE = meno frammentazione interna

Dimensioni delle pagine comprese tra 1Kb e 4Kb (alcuni processori supportano altre dimensioni)
Pagine Grandi---> Per i segmenti di codice Pagine Piccole ---> per i dati dei Thread

- Pagine Grandi = possono far aumentare il page fault perchè le pagine sono di meno e ogni pagina contiene anche indirizzi non utilizzati
- Pagine Piccole = pochi page fault poiché le pagine sono molte e ogni pagina contiene solo le informazioni che servono (Principio di Località)

Dimensioni Ottimali = se la dimensione della pagina si avvicina a quella del processo , i page fault diminuiscono

Page Fault = Il numero di page fault dipende anche dal numero di frame allocati ad ogni processo. Diventa 0 se il numero di frame è sufficiente per contenere l'intero processo e deve essere dimensionato in modo da ottimizzare il rapporto tra Page Fault e Frame allocati

STRATEGIA ALLOCAZIONE DELLA MEMORIA

Decidono quando portare in memoria fisica pagine dello spazio logico di indirizzamento

- **paginazione On Demand** = carica una pagina solo quando è richiesta , causando però molti page fault all'inizio che poi decrescono

- **Preparing** = anticipa il caricamento in memoria rispetto al reale utilizzo

- **Principio Località** = pagine contigue sono utilizzate insieme , ma non sempre è efficace come metodo

Working Set = quantità di memoria fisica allocata ad ogni processo che può variare nel tempo

STRATEGI DI SOSTITUZIONE DELLE PAGINE

Non è possibile scegliere una pagina “vittima” in modo completamente libero , alcune frame devono restare allocati permanentemente o per periodo determinati (locked frame)
Non possono sostituire ad esempio codice nel kernel e strutture dati del sistema operativo .
L'insieme di pagine da esaminare per la scelta della pagina vittima può essere .

- **Locale =** nell'ambito dello stesso processo
- **Globale =** la memoria è un unica risorsa comune

STRATEGIA OTTIMA

La pagina vittima è quella che non verrà utilizzata per più tempo , si generano così un minor numero di page fault. Questa strategia non è prevedibile se non su base statistica , ma può essere usata come strategia di riferimento per valutare le altre strategie

Page adress Stream = sequenza di riferimenti alle pagine nel corso dell'esecuzione di un processo

LEAST RECENTLY USEED (LRU)

La pagina vittima è quella che da più tempo non viene utilizzata , per il principio di località infatti non dovrebbe servire nell'immediato futuro e inoltre si avvicina all'efficienza della strategia ottima.
Lru è una strategia che richiede la gestione di informazioni temporali associate ad ogni pagina. Ad ogni pagina viene associato un **TimeStamp** indicante l'istante in cui s'è verficato l'utlimo accesso/modifica.
Il timestamp minore individua la pagina vittima.

VARIANTI LRU

sono algoritmi simili , meno efficaci , ma più semplici

- **LEAST FREQUENTLY USED (LFU) =** sostituisce al tempo d'accesso un informazione sulla frequenza di utilizzo , tant'è che le pagine vittima sono quelle utilizzate meno frequentemente. Ad ogni pagina è associato un contatore che si incrementa ad ogni accesso , le pagine usate meno frequentemente potrebbero quindi essere quelle più recenti.

- **NOT USED RECENTLY (NUR) =** approssima LRU campionando il tempo ad intervalli molto ampi , ad ogni pagina infatti è associata una coppia di Bit utilizzata e modificata. I bit di utilizzo si azzerano periodicamente , mentre ad ogni scrittura si imposta il bit di modifica. Si seleziona una pagina non utilizzata dall'ultimo reset , se manca si sceglie una pagina non modificata (per non dover riscrivere su memoria esterna)
  Implementazione del NUR sono la strategia di **Clock** e **Fifo**

STRATEGIA FIFO

- **Pagine Fisiche =** gestite secondo una struttura a coda
- **Pagina vittima =** quella che risiede da più tempo in memoria

L'algoritmo Fifo è molto semplice da implementare (buffer circolare = è una struttura di dati che utilizza un singolo buffer con dimensione fissa) e no coincide con la strategia LRU.
E' un algoritmo indipendente dalla quantità e frequenza dei riferimenti , infatti per il principio della località la pagina potrebbe essere utilizzata spesso lungo un intervallo di tempo non breve

La strategia FIFO però **non** è **efficiente**

- non considera l'uso delle pagine ma solo il momento del primo caricamento
- Non riconosce quindi il maggior utilizzo di alcune pagine rispetto alle altre

STRATEGIA DI CLOCK

Pagine fisiche (organizzate in buffer circolari) = Ogi pagina fisica (frame) ha associato un bit di uso che viene imposto a 1 quando :

- una pagina logica viene caricata per la prima volta nel frame
- una pagina caricata nel frame viene utilizzata

Pagina Vittima = la scelta della pagina vittima avviene scandendo il buffer , infatti la pagina vittima è il prossimo frame che ha il bit di uso a 0 , non sarà quindi stata utilizzata di recente. Durante la scansione del buffer , i bit di uso dei frame esaminati vengono rimessi a 0 ; in questo modo una pagina utilizzata frequentemente rimetterà il bit di uso a 1 e non verrà selezionata al prossimo giro

GESTIONE BUFFERIZZATA DEI FRAME SOSTITUITI

I Frame utilizzati per la sostituzione delle pagine sono conservati in una coda in memoria centrale.

- Un frame da sostituire viene accodato alla lista e il corrispondente elemento della page table (serve per memorizzare la mappatura tra indirizzi virtuali e indirizzi fisici) viene azzerato
- La pagina rimane in memoria e può essere riutilizzata senza ricaricarla da memoria secondaria
- durante un page fault si esamina la lista per verificare se la pagina è ancora presente
- Se non è più presente , viene caricata nel frame in cima alla coda (meno utilizzata)
- per minimizzare i tempi di swap si gesticscono due liste diverse per pagine modificate e Non

ALLOCAZIONE DEI FRAME AD UN PROCESSO

l'allocazione dei frame ad un processo deve bilanciare due esigenze contrastanti

1. pochi frame comportano molti page fault
2. molti frame comportano pochi processi in memoria

Si possono quindi adottare molte strategie d'allocazione : Fissa in ambito locale , variabile in ambito globale , variabile in ambito locale

- Fissa - Variabile = determina se la quantità di memoria allocata ad un processo cambia o no
- Locale – Globale = determina lo spazio di ricerca della pagina vittima

ALLOCAZIONE FISSA IN AMBITO LOCALE

Ad ogni processo è assegnato un numero fisso di frame che dipende dalle esigenze applicative , tant'è che durante la gestione di un page fault , i frame esaminati per la sostituzione sono quelli allocati al processo :

- semplice da implementare
- segue una delle strategie esaminate

E' difficile determinare una dimensione soddisfacente per il processo in modo statico e costante

- Troppo Basso = molti page fault
- Troppo Alto = multiprogrammazione limitata

ALLOCAZIONE VARIABILE IN AMBITO LOCALE

I frame liberi sono organizzati in una lista globale , infatti in caso di page fault al processo viene assegnato un frame libero indipendentemente dalla sua attuale dimensione fisica.
Il numero di frame allocati ad un processo aumenta :

- la richiesta di altri processi che nel tempo possono ribilanciare la distribuzione di frame
- se la lista si esaurisce , la scelta su quale processo penalizzare può essere aribtraria
- la gestione bufferizzata dei frame liberi può migliorare l'efficienza

[usato in unix svr4]

ALLOCAZIONE VARIABILE IN AMBITO LOCALE

Ad ogni processo è assegnato un numro predefinito di frame all'inizio dell'esecuzione , cosicchè l'occorrenza di un page fault viene risolta nell'ambito dei frame allocati al processo , in questo modo non si creano squilibri nei confronti di altri processi.
Periodicamente , la quantità di memoria fisica allocata ad un processo è ridefinita sulla base del comportamento osservato , in questo modo si migliora la prestazione adattandosi aòòa dinamica reale dei processi
[usato da Win2000/Xp]

STRATEGIA WORKING SET

Si basa sul principio di località ed è una strategia di allocazione variabile in ambito locale.
Il working set di un processo al tempo t [W(t,w) --->descrive la località del programma] è l'insieme delle pagine utilizzate nelle ultime w unità di tempo.

ANDAMENTO DEL WORKING SET
Il working set di un programma ha una variazione prevedibile

1. Aumenta all'inizio dell'esecuzione , poi si stabilizza per il principio di località
2. Cresce nuovamente quando il processo sposta il proprio indirizzamento verso un altro ambito di località
3. Raggiunge un massimo quando contiene pagine che appartengono alle due località
4. Decresce nuovamente quando il processo si assesta su sul nuovo ambito di località

La determinazione della dimensione dell'insieme di frame assegnati ad un processo si basa sull'osservazione del comportamento del working set

- Si controlla la dimensione di variazione di WS
- Periodicamente si rimuovono dall'insieme di frame assegnati WS le pagine non utilizzata dal WS
- Quando l'insieme dei frame allocati è minore del WS vengono allocate ulteriori frame
- Se non sono disponibili , l'intero processo viene sospeso fuori memoria

PAGE FAULT FREQUENCY (PFF)

Si definiscono un limite superiore U e un limite inferiore L per il tasso di page fault. Se il tasso di page fault supera il valore di U , si allocheranno più frame , viceversa se il tasso di page fault scende sotto il limite di L, si liberano frame.
In questo modo il numero di frame allocati segue da vicino l'effettivo valore del WS