

Sistemi Operativi 1

Il sistema operativo è un software, posto al livello 3 dello standard IEEE, che permette la comprensione delle istruzioni del livello precedente (aggiungendo altre) e offrendo un punto di vista più astratto della macchina. Le nuove istruzioni aggiunte permettono l'esecuzione di più programmi in contemporanea, gestire utenti diversi, gestire file e raccoglierli in cartelle e gestire facilmente dispositivi di comunicazione.

VANTAGGI DEL PROGRAMMATORE VANTAGGI DEL SISTEMA

Le nuove istruzioni astratte permettono al SO di ridurre il ruolo di:

una facilità d'uso, consentendo, ad esempio, • **GESTORE**: viene coordinato l'utilizzo delle risorse da parte di l'interazione coi dispositivi I/O e più programmi; tutti i programmi, minimizzando i tempi morti e distribuendo le attese in esecuzione contemporaneamente.

• **CONTROLLORE**: a controllare che i programmi non accedano o compromettano le risorse del SO o d'altrettanti programmi;

RISORSE DEL SISTEMA

Le risorse permettono la condivisione:

della CPU e dei dispositivi I/O (time multiplexing) La multiprogrammazione permette di evitare i tempi morti della memoria e del disco (space multiplexing) ponendo la CPU a un altro programma quando il programma in esecuzione coi dispositivi I/O o più programmi ha finito i tempi morti e distribuendo le attese in esecuzione contemporaneamente. Per farlo, il SO utilizza il **CONTEXT SWITCH**, un meccanismo che permette il passaggio della CPU a bios o con il SO. Per farlo, si ha bisogno salvando lo stato del programma in esecuzione e caricandogli i due modelli:

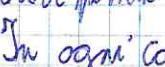
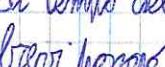
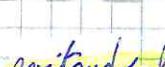
MULTIPROGRAMMAZIONE

MODO UTENTE: esegue il codice dei vari programmi;

MODO KERNEL: esegue il Kernel e altri istruzioni non eseguibili in modo utente, è possibile accedere a indirizzi di memoria riservati al SO.

Passaggio a modo Kernel avviene tramite system call/trap oppure quando si fissa una routine di gestione tramite interrupt.

TIME P1 P2 P3



CPU attende CPU

I/O attende I/O

Il SO, gestendo le risorse, permette di condividere CPU e memoria.

All'aumentare dei livelli di multiprogrammazione,

il carico di CPU-bound cresce più

rapidamente rispetto all'I/O bound.

TIME SHARING

Il time sharing consente a qualsiasi

CPU di lavorare programmi grazie a un timer che invia interrupt alla CPU,

evitando che un programma monopolizzi la CPU. Il quanto

di tempo del timer è solitamente piccolo, tuttavia tempi troppo

lunghi possono portare all'overhead (costi aggiuntivi intorno e dopo).

In ogni caso, il quanto di tempo utilizzato è quello del context switch.

Il Kernel vengono aperti file, memoria virtuale, ecc.
I kernel degli SO microkernel è più robusto
perché comprende "le cose minime" e tutto il resto
è in modo utente, permettendo l'esecuzione parallela
simultanea di più programmi.

TIME P1 P2 TIME P1 P2 SO

tempo del context switch

ARCHITETTURE MODERNE

Grazie all'avvento delle nuove tecnologie, è possibile eseguire più programmi con una risorsa d'elaborazione in contemporanea.

In più si possono combinare con le soluzioni dette in precedenza. Tuttavia rimane il fatto che, con un certo numero di programmi presenti:

- De un programma che da monopoli la CPU deve attendere I/O, lascia il posto a un altro programma;

- È meglio evitare monopolizzazione della CPU.

ASTRAZIONI DEL SISTEMA OPERATIVO: I PROCESSI

Un processo è un'attività d'elaborazione guidata da un programma ed è caratterizzato dai:

- Uno stato d'esecuzione (stato d'indirizzi e valori rispetto al sistema operativo).
- Le informazioni della tabella dei processi del SO.

Un processo è in grado di creare altri processi e può comprendere più thread, avversi programmi sequenziali, eseguiti in parallelo, facente parte di un unico applicativo. Altri metodi sono l'invio di messaggi in mailbox.

MEMORIA VIRTUALE

La memoria virtuale permette ai processi d'utilizzare parte del disco nel caso ne richiedano più di quella disponibile. Attraverso i file system, è possibile utilizzarla gestita dal SO attraverso opportune system call. I file e le cartelle sono ragionati in modo gerarchico.

MECCANISMI DI PROTEZIONE

Grazie alla protezione dei dati dei file degli utenti. La protezione è garantita dall'autentizzazione e dai permessi. È inoltre possibile impostare un bit, detto setuid, che modifica il proprietario del processo che lo esegue, permettendo l'accesso ai file di un altro utente.

INDIRIZZI VIRTUALI E FISICI

La convenienza di più programmi e la loro allocazione è gestita in modo flessibile da indirizzi di memoria virtuali, che vengono traslotti in fisici appena non dovrà essere allocato il programma, o può fare a tempo di compilazione (bigallo vincolante), coricamento (non si può ridurre durante l'esecuzione).

VANTAGGI DELLA MULTI PROGRAMMAZIA

La CPU viene sfruttata meglio, dato che la passa a un altro programma quando il primo è in attesa.

PROBLEMI

Eseguire più programmi in contemporanea può generare interferenze, garantendo l'esecuzione protetta.

SCOLZIONI

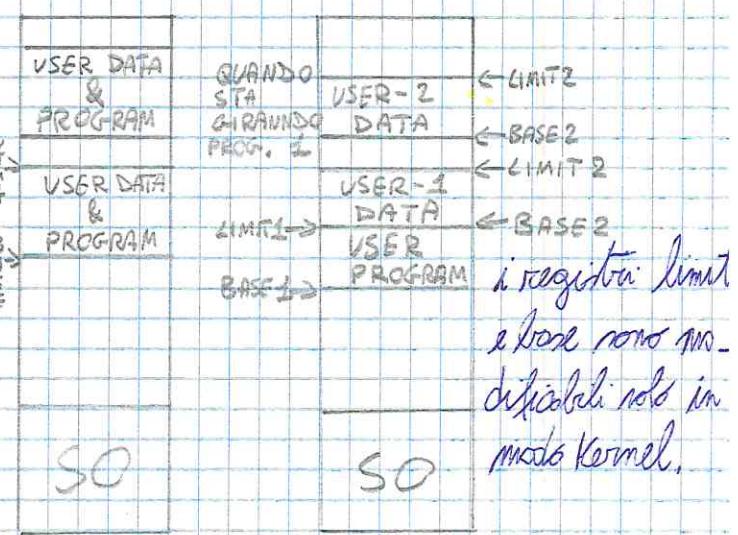
- 1) modo Kernel e modo utente;
- 2) distinguere utenzioni privilegiate (nel kernel) e normali;
- 3) gestione della memoria: maneggiare accesso a carica del SO o di altri programmi;

COMUNICAZIONI TRA PROCESSI

pipe: un "file" avente due estremità, una scrive e l'altra leggibile attraverso le system call, la sincronizzazione tra lettori e scrittori è gestita dal SO eseguito in parallelo, facente parte di un unico applicativo. Altri metodi sono l'invio di messaggi in mailbox.

FILE E DIRECTORY

I file e le cartelle sono ragionati in modo gerarchico. Attraverso i file system, è possibile utilizzarla gestita dal SO attraverso opportune system call. I registri limit e base sono modificabili solo in modo kernel.



La traduzione virtuale-fisica viene eseguita dall'hardware e dal MMU, in più viene anche eseguito il controllo di non superamento dei limiti.

GERARCHIA DELLA MEMORIA

REGISTRI

CACHE

RAM

DISCO

la gerarchia della memoria consiste nel portare le informazioni più utilizzate da un programma in area in modo da ridurre i tempi d'accesso, per questo motivo sono state inventate le cache.

Il disco utilizza la RAM come una cache: troverai la memoria virtuale, il disco scrive in RAM parti del programma, cominciando altre quando serve la gestione avviene via software dal SO).

La gerarchia porta anche buon frutto alla qualità di licetà dei programmi.

INTERRUPT

I/O può essere gestito anche da interrupt, questo consente a linea libera la CPU, ovvero non ha interrupt nell'opportuno bus quando necessario.

Ma ricezione di un interrupt, la CPU:

- completa l'istruzione macchina in corso;
- Salva lo stato della computazione interrotta;
- Salva alla posizione di memoria dove c'è l'interrupt handl.
- Gestisce l'interrupt eseguita lo stato precedente.

Per la gestione degli interrupt, la CPU deve sapere rilevare la presenza, leggere l'id del device, disabilitare gli interrupt (in modo Kernel) mettendo un bit in un opportuno registro e, alla fine, interruppt ad assegnare la CPU ad un altro processo.

risabilitarli tornando al proceso interrotto.

PROCESSI

Il processo è un'astrazione del SO, guidata dal programma; la velocità dipende dal numero di processi e dalla CPU/memoria condivisa.

NOTA: un processo non coincide col programma, infatti quest'ultimo può contenere molti di essi.

dato di un processo viene mantenuto dal SO che contiene i valori dei registri e i dati in memoria.

MEMORIE DI MASSA

Le memorie di massa hanno in genere il compito di memorizzare i dati utilizzati dal SO e dai programmi, si interfacciano all'hardware attraverso SATA. La completezza di questa interfaccia viene fornita attraverso i driver: dei software in grado di interrogare col controller, offrendo funzioni semplici agli strati superiori. La memoria di massa è anche utilizzata come supporto alla memoria virtuale e al file system, gestita in modo trasparente dal SO.

TRAP

Le trap sono interruzioni software provocate da determinate istruzioni, infatti vengono trattate allo stesso modo degli interrupt. Le system call sono realizzate tramite trap. A differenza degli interrupt, le trap sono rintracciabili al programma in esecuzione perché causate proprio da quest'ultimo.

* Più processi in uno stesso programma danno l'idea di uno pseudo-parallelismo (ogni processo sembra parallelo agli altri ma in realtà non lo è). La velocità di un processo non si può garantire borondosi solo sul tempo d'esecuzione della CPU, infatti non si può garantire che:

- un processo arriverà a un certo punto del codice in un dato tempo (velocità assoluta);
- un processo raggiungerà un certo punto prima che un altro processo ne raggiunga un altro (velocità relativa).

Questo perché, durante l'esecuzione, potrebbero arrivare interrupt ad assegnare la CPU ad un altro processo.

Esistono comunque metodi per garantire la velocità dei processi:

- sincroni hard/soft (velocità assoluta);
- sincronizzazione (velocità relativa);

STATI DI UN PROCESSO

Un processo è definito da diversi stati:

- Attivo: processo in esecuzione;

- Waiting: processo in attesa di un evento;

Lo stato Attivo è a sua volta diviso in due sottostati: le interruzioni.

- Running: esecuzione effettiva;

- Ready: processo pronto ma la CPU è impegnata; considerando risorse software, il SO dovrà quindi

SCHEDULER, DISPATCHER

ATTIVO
READY 2 5 RUNNING

1 4 HANDELER 3 6

NEW 5 EVENTO

WAITING

TERMINATED

Lo scheduler è un componente del SO che si

occupa di scegliere un processo tra quelli ready

da mettere in running, il dispatcher invece

applica il context switch. Come è possibile notare,

vi è una relazione tra politica e meccanismo

permesso una maggiore flessibilità e potabilità del SO.

In questo modo, è possibile utilizzare un meccanismo

su diverse politiche e vicende. Ritornando a parlare

degli stati di un processo, occorre aggiungere altri due:

- New: creazione di un processo;

- Terminated: il processo è terminato ma il SO ne tiene ancora

traccia per eventuali informazioni sulla terminazione.

GESTIONE DEI PROCESSI

Il SO mantiene un Process Control Block per ogni

processo nel sistema, che contiene:

- valori dei registri della CPU;

- Stato del processo;

- Dati per lo scheduler;

- Dati per accounting;

- Altre informazioni per la protezione;

La tabella contiene tutti i PCB presenti nel sistema.

I PCB possono essere considerati i nodi

di una lista di processi ready e le code

d'attesa di eventi.

Data che i processi ponono molte funzioni differenti in contemporanea, si può dire che sono in competizione tra loro per l'accesso alle risorse condivise ed è compito del SO evitare le interferenze. Più processi sono però coinvolti per raggiungere un determinato obiettivo e quindi dovranno garantire la comunicazione e la sincronizzazione fra di essi.

VANTAGGIO: la programmazione diventa modulare.

PROBLEMA: Nelle attività cooperanti, il contest non è diviso più così, anche se le risorse sono condivise.

SOLUZIONE: introduzione dei thread.

THREAD

I thread sono flussi di controllo che condividono la stessa memoria ed eseguono parti diverse del codice.

I thread sono utili per le seguenti ragioni:

- parallelismo all'interno d'un'applicazione;

- raccolta di I/O e CPU più semplice rispetto alla sincronizzazione di risorse fra attività cooperative;

- creazione/mutual di thread meno costoso rispetto ai processi.

Ogni thread ha il suo stato, i suoi registri e il suo spazio d'indirizzi e condiviso e non vi è alcuna interazione fra thread, perché progettato per la cooperazione e la condivisione di risorse.

I thread possono essere implementati in modo utente (il SO non ne è a conoscenza) e in modo Kernel.

Il sistema di gestione dei thread è costituito da una libreria di funzioni, per ogni processo, ovvero più thread, associato alla coda, si monta una tabella

dei thread con il loro id e assegna al processo il tempo del processo ai relativi thread.

THREAD IN MODO KERNEL

Vengono gestiti direttamente dal SO tramite le system call, inoltre si occupa anche della schedulazione.

THREAD IN MODO UTENTE

Vengono gestiti attraverso funzioni di libreria ad-hoc;

1. THREAD IN MODO UTENTE

VANTAGGI:

- tempi di context switch bassi;
- implementabile sopra il SO;
- si possono applicare politiche di scheduling ad-hoc.

SVANTAGGI:

- completa gestione di system call bloccanti senza bloccare tutti i thread di un processo; se ne deve occupare la libreria di system call;
- non possono esercitare più thread di un processo in running, anche con più CPU;
- hanno bisogno di chiamate esplicite a delle funzioni per permettere il time-sharing.

CONSEGUENZE DELLA CONDIVISIONE DELLA MEMORIA

Ogni processo/thread che modifica una variabile condivisa, la renderà disponibile anche agli altri. Il recensore dell'ordine delle istruzioni appartiene a ogni esecuzione corrispondente output diversi.

PROBLEMA: se due thread accedono in contemporanea alla stessa variabile per modificarla, il risultato finale potrebbe non essere la "concatenazione" delle due modifiche.

SOLUZIONE: sincronizzazione di processi/thread

SINCRONIZZAZIONE

La sincronizzazione consiste nel far attendere in contemporanea, solo una delle modifiche sarà quella a un processo/thread che un altro processo/thread sia effettiva mentre l'altro andrà avanti. Questa condizione avviene a un certo punto del codice per poter proseguire è detta **seziona critica**, come risolverla? Si inserisca, è molto utile quando bisogna accedere a dati condivisi, all'inizio è alla fine del codice "critico" delle istruzioni, anche se ci sono altre utilità. I meccanismi riportati di sincronizzazione.

trovare in linguaggi concorrenti (attraverso funzioni ad-hoc) system call e librerie di funzioni. L'interazione fra i processi/thread è principalmente data dai seguenti modelli:

- Modello a memoria condivisa: i processi/thread condividono alcune variabili in memoria;
- Modello a scambi di meneggi: i processi/thread si scommettono messaggi in modo esplicito.

1. THREAD IN MODO KERNEL

VANTAGGI:

- è possibile avere più thread in running per brano quando si ha più CPU;
- la gestione delle system call bloccanti non pose alcun problema, anche con thread in attesa.

SVANTAGGI:

- context switch costoso perché richiede il passaggio in modo Kernel;
- vi sono limitazioni sul numero dei thread dato che il SO deve predisporre strutture dati per la gestione.

Per ridurre gli svantaggi, è possibile combinarne entrambe le soluzioni, utilizzando:

- un numero N di threads utente;
 - un numero M di threads Kernel;
- Il sistema di gestione dei thread si occuperà della corrispondenza tra i meccanismi d'intervento devono permettere lo scambio d'informazioni, il corretto ordinamento delle azioni d'apertura e chiusura, il accesso controllato alle risorse condivise, onde evitare eventuali interferenze.

CORSE CRITICHE

Il problema delle variabili condivise riguarda soprattutto le modifiche: se due processi che lavorano con una variabile condivisa effettuano un'operazione su essa

in contemporanea, solo una delle modifiche sarà quella a un processo/thread che un altro processo/thread sia effettiva mentre l'altro andrà avanti. Questa condizione avviene a un certo punto del codice per poter proseguire è detta **seziona critica**, come risolverla? Si inserisca, è molto utile quando bisogna accedere a dati condivisi, all'inizio è alla fine del codice "critico" delle istruzioni, anche se ci sono altre utilità. I meccanismi riportati di sincronizzazione.

REQUISITI PER UNA BUONA SOLUZIONE:

- **Mutua esclusione:** nessun processo deve eseguire la sezione critica quando un altro è in esecuzione lì dentro.
- **Progresso:** se nessun processo è in sezione critica ed è di essi intendono farlo, la ceduta di chi prende ricade su quest'ultimo.
- **Altra limitata:** se un processo fa per entrare nella sezione critica, si accorda l'accesso se non supera il limite massimo degli accessi.

altri requisiti sono:

- garantire la mutua esclusione nelle sezioni critiche;
- non doverci sulla velocità dei processi e sul numero di interruzioni di processi durante la sezione critica.
- un processo non in sezione critica deve impedire l'accesso a PROBLEMI: per motivi di protezione, è opportuno evitare allo stesso da parte di altri;
- un processo, in attesa del suo turno per la sezione critica, non deve attendere un tempo indeterminato;

"SOLUZIONE" CON BUSY WAITING: ALTERNANZA STRETTA

Il funzionamento di questa soluzione è simile a quella precedente, tuttavia non ci soddisfa il requisito di progresso: se un processo può entrare nella sezione critica, ma non vuole farlo, l'altro non lo può fare perché non è il suo turno.

"SOLUZIONE" CON BUSY WAITING: ALGORITMO DI PETERSON

In questo algoritmo, il progetto si dichiara di voler entrare in sezione critica modificando alcune

variabili. Se un processo è l'unico a voler entrare in sezione critica, esce dal ciclo while. Se invece di conseguenze non è garantita il funzionamento nei due casi: i processi non interventi alla sezione critica, entra prima chi detta prima le variabili, l'altro possono entrare quando verranno blaccate.

"SOLUZIONI" CON BUSY WAITING: ISTRUZIONI TSL

Le istruzioni TSL (Test and Set Lock) permettono di rendere atomico una sequenza d'azioni effettuando prima del test su zero e poi impostandola a 1. Nei sistemi multiprocessore, TSL deve riservare il bus fino a quando lettura e scrittura della variabile non sono concluse. Ese, se usate in un certo modo, possono garantire la mutua esclusione, tuttavia non soddisfa il requisito d'attesa limitata (anche se è utilizzata nelle sezioni brevi in tempo).

PROBLEMA DEL PRODOTTORE-CONSUMATORE

In questo problema vi sono più tipi di sincronizzazione: bisogna ordinare l'attività dei processi e poi ottenerne quanto mancano le risorse. Il produttore inserisce dati nel buffer

SOLUZIONE: DISABILITAZIONE INTERRUPT

Con la disabilitazione degli interrupt, si evita ogni interruzione di processi durante la sezione critica. Un processo in modo intatto disabilita gli interrupt (potrebbe moltiplicare la CPU con la sezione critica), in più la funziona nei sistemi con una CPU: le altre fanno girare i processi in parallelo (la disabilitazione degli interrupt è lasciata alla CPU).

"SOLUZIONE" CON BUSY WAITING: VARIABILI LOCK

Si utilizza una variabile booleana per controllare l'accesso alla sezione critica, bloccandola all'entrata e sbloccandola. PROBLEMA: la procedura per entrare nella sezione critica d'un ulteriore sezione critica: i processi potrebbero tenere la variabile bloccata in contemporanea e quindi viene meno la mutua esclusione.

ARCHITETTURE MODERNE

Le soluzioni proposte sono pensate per i sistemi con una CPU: se un processo è l'unico a voler entrare in sezione critica, esce dal ciclo while. Se invece di conseguenze non è garantita il funzionamento nei due casi: i processi non interventi alla sezione critica, entra prima chi detta prima le variabili, l'altro può avvenire in ritardo se la CPU dovrà riservare un bloccaggio nella cache di un altro. Altri miglioramenti sono forniti con istruzioni memory fence / barrier (costose in tempo). Nell'algoritmo di Peterson è possibile inserire un fence nel while per evitare anticipazioni sulla crittura.

La mutua esclusione non è l'unico problema di risanamento: gli processi, infatti, devono anche:

- gestire la condivisione di risorse tra processi (thread);
- garantire che una parte di cache di un processo sia eseguita prima di un'altra parte di cache di un altro processo;
- garantire che tutti i thread completino una fase prima di passare alla successiva.



ASTRAZIONI PER LA SINCRONIZZAZIONE

Le soluzioni finite in precedenza sono esclusivamente come system call e, per l'esecuzione atomica, a basso livello per risolvere problemi più complessi. - nei sistemi con una CPU si possono disabilitare gli interrupt.

Le soluzioni ad alto livello sono i semafori, i monitori, i mutex e le variabili condizionate.

Inoltre è opportuno ridurre al minimo il busy waiting perché fa sprecare tempo alla CPU e, nei sistemi con una CPU, potrebbe invertirsi la priorità: due processi, uno con priorità maggiore dell'altro, lo scheduler sceglierà sempre il primo.

POSSIBILE SOLUZIONE: system call sleep e wakeup, è possibile utilizzare un semplice condizionale, la prima sospende il processo che l'ha chiamata mentre lo secondo lo risveglia. Nel modello produttore/consumatore non:

- stare sleep sul produttore quando il buffer è pieno o sul consumatore quando è vuoto;
- stare wakeup sul produttore/consumatore quando il buffer è non vuoto.

È possibile applicare queste system call nel produttore/consumitore attraverso una contatore che memorizza la posizione finale del buffer.

PROBLEMA: Con una certa combinazione d'eventi, maggiore o uguale a zero ed è utilizzata per una dei due processi potrebbe rimanere rispettivo i problemi di sincronizzazione con un certo numero remoto. **SOLUZIONE:** i semafori.

SEMAFORI

Semafori sono tipi di dati cariati, inizializzabili a un valore intero, modificabile con determinate funzioni, i cui valori, quando diventa minore di 0, permette di sospendere il processo chiamante. Fondamentalmente, semafori utilizzano 3 operazioni:

- init → inizializza con un valore intero;
- up → incrementa il valore di un processo in attesa per down;

- down → decrementa il valore se maggiore di 0, altrimenti mette il processo in attesa. Le 2 down sono operazioni atomiche ed escludono le sezioni critiche.

Le funzioni up e down ci potrebbero implementare le funzioni multithreaded, per l'esecuzione atomica: nei sistemi con una CPU si possono disabilitare gli interrupt.

- nei sistemi multicore si possono utilizzare le istruzioni TSC. Il busy waiting è accettabile dato che la sezione critica è breve.

MUTUA ESCLUSIONE COI SEMAFORI

Per garantire la mutua esclusione nelle sezioni critiche, è possibile utilizzare un semplice condizionale, inizializzato a 1, in questo modo solamente un processo/thread alla volta può eseguire la sezione critica.

SEMAFORI BINARI

I semplici binari sono semafori che possono assumere solo i valori 0 o 1, il loro compito è quello di garantire la mutua esclusione (infatti vengono anche chiamati mutex).

SEMAFORI CONTATORE

Un semaforo contatore può assumere qualsiasi valore maggiore o uguale a zero ed è utilizzato per una dei due processi rispettivo i problemi di sincronizzazione con un certo numero remoto. Possibili utilizzi: produzione- consumo da cinghiale. Possibili utilizzi: produzione-

SPINLOCK

Gli spinlock sono semplici binari, implementati con altrettanto elettoro per garantire la mutua esclusione, sono efficaci se la sezione critica è breve (sotto cento switch) oppure quando i thread girano su CPU differenti: Una condanna i ciclo in attesa attiva su una CPU mentre l'altro continua finché non ha finito.

FUTEX

I futex sono una variente della spinlock in cui si decrementa una variabile leggendo il valore precedente: se era 1, va avanti, altrimenti viene notificato il thread tramite signal. All'arrivo si effettua un inserimento e si sveglia un thread a memoria.

DEADLOCK

Il deadlock è un problema di sincronizzazione in cui due o più processi attendono un evento controllato solo da uno degli altri.

SOLUZIONE DEI 5 FILOSOFI

- Per evitare la starvation, si può imprimere a un filo di mangiare K volte di fila quando ha un vicino affamato - ogni volta che un filosofo con un piatto affamato prele forchette, incrementa un contatore;
- quando il contatore arriva a K, il filosofo mangia.

PROBLEMI DEI LETTORI E SCRITTORI

Supponiamo di avere una struttura dati in cui ogni utente può leggere / scrivere in parallelo (per ragioni d'efficienza di uso). Per mantenere la consistenza della struttura occorre che lo scrittore avvenga in mutua esclusione con le altre operazioni:

Si utilizzano una variabile condizionata che indica il numero di lettori, e due semafori: uno per accedere al database, l'altro per accedere alla variabile.

Se è presente uno scrittore, il primo lettore viene bloccato dal primo semaforo, tutti gli altri dal secondo;

Se il primo lettore supera il primo semaforo, anche gli altri lo superano; quando un lettore finisce, aggiorna la variabile (se è l'ultimo e non blocca più nessuno, rilascia la mutua esclusione).

POSSIBILI SOLUZIONI:

- Combinare l'ordine d'acquisizione delle bacchette a simboli, utilizzare semafori privati per lettore / scrittore per mettere in attesa quando non hanno accedere alla struttura.
- Utilizzare un semaforo contattore, inizializzato a 4, e applicare down e up prima e dopo l'acquisizione delle bacchette.
- Acquisire più risorse contemporaneamente e utilizzare un array di semafori: quando il processo i non può prendere bacchette, viene notificato dal semaforo i.

Il semaforo usato in questo modo è detto **semaphore privato**.

SEMAFORI PRIVATI

I semafori privati sono tali perché, anche se il meccanismo è lo stesso dei normali semafori, non vi è alcun controllo su chi (processo / thread) e come vengono utilizzati.

Un semaforo privato è inizializzato a 0, solo un determinato processo deve seguire la down (per favorire le condizioni di sincronizzazione) e tutti possono seguire

POSSIBILI MIGLIORAMENTI: gestire la starvation

In insieme di processi deve poter acquisire risorse da un pool grande K. Dopo l'utilizzo, lo ritorna vuoto.

Si utilizza un semaforo inizializzato a K, se due processi sono in attesa e un terzo processa una UP, la ad-

di chi viene rassegnato è in UP. È anche possibile utilizzare i semafori privati: il processo che mette in attesa viene notificato su un semaforo privato e ricondotto all'uso di una variabile, al rilascio della risorsa si segnala un processo (grazie alla variabile) e lo si riavvia.

1 MONITOR

I monitor sono costrutti linguistici per creare nuovi tipi di dato e permettere la sincronizzazione delle funzioni definite per esso con un meccanismo automatico e uno non automatico. Se più processi eseguono delle funzioni sulla stessa struttura dati, esse mostrano in mutua esclusione in automatico, sarà il compilatore a indicare le opportune synch call a inizio/fine della funzione.

a segnalazione di una condizione da parte di un processore implica il risveglio di uno dei processi in attesa su quella condizione, tuttavia entrambi non possono sperare sulle variabili condizionali, infatti:

- la segnalazione comporta l'uscita dalla funzione;
- il processo risvegliato attendrà la mutua esclusione insieme ad altri, in attesa del suo rilascio;
- il processo segnalante continuerà l'esecuzione.

2 DIFFERENZE TRA SEMAFORI E MONITOR

un problema risolto in un modo può essere risolto anche con l'altro; i monitor permettono di realizzare programmi strutturati.

3 SCHEDULING

ci sono diversi code:

coda dei Job: processi di cui è richiesta l'esecuzione;
coda dei processi ready: processi pronti a usare la CPU;
coda dei dispositivi: richieste I/O pendenti per ogni dispositivo e processore.

processi migrano da una coda all'altra nel corso della loro vita.
esistono diversi tipi di scheduler:

Job scheduler: decide quali job mettere nella coda dei Job, in un ordine prefissato.

CPU scheduler: seleziona il prossimo processo da eseguire.

- Medium-term scheduler: regola la multiprogrammazione in base alle risorse (memoria in particolare) disponibili.

traverso swap-in e swap-out, si possono porchiaggiare immagini dei processi in disco e viceversa.

VARIABILI CONDIZIONE

Le variabili condizione sono meccanismi di sincronizzazione in cui è possibile far attendere un processo/thread su essa per ottenere il cambio di valore di una condizione booleana, segnalandole quanto avvenne. Quando la variabile viene segnalata, viene rilasciata la mutua esclusione (anche se il processo che l'ha segnalata "mette" quello risvegliato già in mutua esclusione). Non vi è potenza di segnalazione con effetto perché (per la mutua esclusione) se un processo esegue del codice con una wait su una condizione, è improbabile che un altro processo lo segnali prima che il primo effettui una wait.

4 PREVENIRE LE DEADLOCK

Le deadlock avvengono in particolare durante l'allocazione delle risorse, i processi attendono infatti che un altro processore rilasci una risorsa. Per prevenire un deadlock di questo tipo, devono fare le seguenti condizioni:

- le risorse sono allocate in mutua esclusione;
- non bisogna portarle via a chi le sta utilizzando;
- un processo può chiedere altre risorse;
- un processo attende una risorsa di un altro processo.

Gli approcci alla deadlock sono i seguenti:

- ignorare il problema;
- rilevare le deadlock e fare recovery senza reboot;
- gestire le risorse per preventivo;
- si utilizzano algoritmi d'allocazione che evitano di bloccare quando si prevede una possibile deadlock.

Per evitare hold & wait si chiude tutto le risorse mentre prima dell'utilizzo oppure, se si hanno già delle risorse, liberarle e prendere le altre.

Per evitare l'attesa circolare, si impone la richiesta di risorse

OBBIETTIVI DELLO SCHEDULING

- Assegnare in modo equo la CPU ai processi; generali
- mantenere utilizzate le risorse
- minimizzare i tempi di risposta; (interattivi)
- soddisfare richieste introducendo altre priorizzazioni al tempo d'esecuzione; (interattivi)
- rispettare le scadenze, sempre e con poche eccezioni (real time)

CPU SCHEDULING

Il CPU scheduling avviene quando:

- il processo in running termina / si stoppa;
- viene creato un nuovo processo;
- interrupt da timer / dispositivo I/O

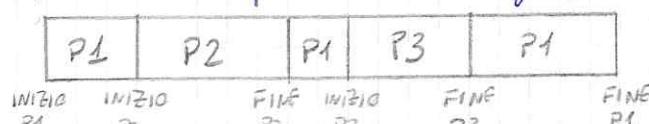
Il CPU scheduling deve avvenire nel primo caso, negli altri non è necessario il suo intervento

JOB SCHEDULING

L'utilizzo efficiente delle risorse è possibile delegando il CPU-burst e l'I/O bound dei processi.

SCELTE DELLO SCHEDULER

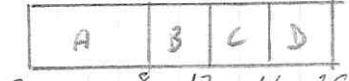
Le scelte dello scheduler sono rappresentabili nei diagrammi di Gantt, in cui si può vedere il tempo d'attesa, di permanenza e quello medio d'ogni processo.



Diversi Job vengono eseguiti fino al completamento

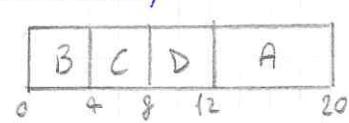
POSSIBILI ORDINI:

- First-Come First-Served (turnaround: 14)



- Shortest Job List (richiesta durata dei Job)

turnaround : 11



SJL risulta ottimale quando i Job arrivano in contemporanea, è anche possibile applicarlo quando i Job non arrivano insieme ma non è molto efficace. SOLUZIONE: Shortest Remaining Time Next

Questo algoritmo può trasferire la CPU a chi la sta utilizzando più a lungo.

Per ridurre la durata di un Job, i due algoritmi fanno una stima e la utilizzano come limite: se il Job lo supera non viene completato.

Un problema di SJF e SRTN è la possibilità di starvation quando un processo viene rifiutato da quelli più co

SJF e SRTN possono essere applicati anche nel CPU-schedul

sulla base della stima del prossimo CPU-burst.

In più bisogna avere un tempo medio di risposta piccolo (dato che il processo CPU-burst è il più costoso) per anticipare le rich

dei dispositivi, al termine infatti:

- l'utente ottiene una risposta e il processo attende la ricezione;
- il processo sottopone un'operazione al dispositivo, ricevuta, richiesta, la più modesta avanti a ottimizzare la gestione di

STIMA DEL CPU-BURST

$$T_{m+1} = \alpha t_m + (1 - \alpha) \bar{t}_m$$

↑ ↑ ↑
 STIMA DURATA $0 \leq \alpha \leq 1$
 CPU BURST CPU-BURST

SCHEDULING NEI SISTEMI INTERATTIVI

Il criterio più semplice per questi sistemi è il Round Robin

se un processo è rimasto in running per un quanto di tempo

va in fondo alla coda dei processi e viene fatto il primo

La durata del quanto di tempo deve minima 1 e volte

il tempo del context switch ma non troppo per evitare tempi

risposta alti, riduttamente tempi a migliorare il turnaround quando

è lungo.

Dato che i CPU burst sono piuttosto brevi, è possibile,

dato un quanto di tempo grande abbastanza, completarne al-

una buona parte all'interno del quanto.

SCHEDULING CON PRIORITÀ

È possibile assegnare dei livelli di priorità ai processi per garantire una scelta ad alta priorità, più essere delle attivazioni parallele esterne oppure interne al SC.

PROBLEMA: un processo più andrà in lavorazione perché non viene chiamato.

SOLUZIONE: incrementare la priorità col passare del tempo

Lo scheduling può essere organizzato

LA CACHE

a cloni, ciascuna con un tipo di scheduling. L'aggiornamento della cache è gestito dall'hardware. I processi con la classe più alta girano per e può essere modulata a ogni contest switch. Se brimi. L'appartenenza a una classe può essere l'hit rate è alto il tempo d'accesso medio è informata e dipende da diversi fattori.

APPARTENENZA DINAMICA ALLE CLASSI

In ogni classe viene eseguito il Round Robin, in una quantità più piccola per le classi più alte. In un processo inizia dalla classe più alta e, se utilizza tutto il quonto, viene declarato. Per eseguire la starvation, un processo viene promosso ogni volta che non viene scelto.

RICLOCAZIONE E PROTEZIONE NEI SISTEMI MULTIPROGRAMMA

Per allocare liberamente i processi, si separano gli indirizzi logici da quelli fisici, l'interazione tra essi può avvenire durante la compilazione, il caricamento oppure real-time. La traduzione avviene effettuata dal MMU al momento dell'accesso memoria, inoltre garantisce la protezione del SO e dei processi confinando un processo nello spazio assegnato.

SCHEDULING A QUOTE

Lo scheduling a quote funziona assegnando una o più quote a seconda della priorità. La quota assegnata a chi è in difetto rispetto all'altra user bottiglia. È possibile usare nelle code multiple assegnando una quota a ogni coda.

ALLOCAZIONE IN PARTIZIONI FISSE

Le allocazioni vengono configurate dal sistema all'avvio e le code possono essere separate per partizione oppure all'avvio. PROBLEMA: vi è spreco di memoria e frammentazione interna (parte della memoria assegnata a un processo non viene utilizzata).

SOLUZIONE: riorganizzare le partizioni.

ALLOCAZIONE IN PARTIZIONI VARIABILI

Si cerca un spazio abbastanza grande e si ricava un processo, fatto con la creazione/termine d'altro processo si creano spazi non contigui.

PROBLEMA: frammentazione esterna (spazi non assegnati e troppo piccoli per essere utilizzati).

SOLUZIONE: compattare periodicamente la memoria e di utilizzarla il disc come "deposto" facendo lo swapping (swap-out per fare spazio, swap-in quando libera memoria).

Con i processi di dimensione variabile, conviene allocare più spazio già dall'inizio, nel caso ci ecceda, giorni spostare il processo in uno spazio più grande e fare la swap-out.

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

ALGORITMI D'ALLOCAZIONE

- **First fit:** il processo viene caricato nella prima allocation grande abbastanza disponibile;
- **Next fit:** Simile al First fit ma parte dall'allocazione in questo modo sono eseguire più processi le cui dimensioni insieme superano la capacità della RAM.
- **Best fit:** Il processo viene caricato nell'allocazione la memoria virtuale ha meno perché può contenere opportunamente grande disponibile;
- **Worst fit:** Il processo viene caricato nell'allocazione piccola perché, quando serve, carica molte pagine (ogni utilizzate). Un altro motivo riguarda la facilità: vengono tolte le hoanine in cui non vi sono stati accessi nel lungo tempo.

ALLOCAZIONE NON CONTIGUA

È possibile suddividere l'immagine di un processo in pagine:

- **Segmentazione:** il processo viene diviso in spazi lineari detti segmenti;
- **Paginazione:** il processo viene diviso in pagine di dimensione prefissata.

L'indirizzo è dato dal numero segmento/pagina più l'offset. **NOTA:** Le pagine hanno un unico spazio d'indirizzamento anche se può essere diverso.

Le pagine e i segmenti vengono caricati in memoria in posti non necessariamente contigui, migliorando l'utilizzo della memoria:

- Vi è meno frammentazione interna con la segmentazione;
- Vi è meno frammentazione esterna con la paginazione perché limitata a metà pagina.

La traduzione degli indirizzi, però, si complica.

La paginazione è trasparente al programmatore perché utilizza un unico spazio d'indirizzi.

La segmentazione invece è visibile, ma permette la protezione dei vari segmenti e, in più facilita la condivisione di segmenti di codice. Inoltre è possibile combinare le due modalità creando la segmentazione paginata.

MEMORIA VIRTUALE

È possibile caricare l'immagine di un processo nel disco e caricare in RAM solo la parte utile al momento corrente.

In questa memoria virtuale la memoria ha meno perché può contenere posti di processo strettamente chiamata o utilizzate in questo momento.

La memoria virtuale è stata inventata infatti per non di eseguire programmi che richiedono più spazio in RAM. Questo permise ai programmatori, nei sistemi primativi, spaccare il codice del programma in overlay, però la gestione è a carico del programmatore. Il demonio pagina modifica automaticamente questa funzione in modo trasparente. Queste tecniche, nei sistemi multiprogrammati, permettono di tenere in memoria più processi anche quando occuperebbero più spazio in RAM.

MEMORIA VIRTUALE PAGINATA

L'immagine del processo in memoria viene divisa in blocki uguali chiamati pagine, la memoria viene invece divisa in base alla stessa dimensione delle pagine. Per ogni processo viene mantenuta una tabella delle pagine, la quale indica dove le pagine sono in RAM e in quale frame. Per mantenere interamente l'immagine del processo, occorre tenere traccia delle pagine su disco. MMU, dato l'indirizzo logico, dovrà individuare la pagina e controllare se è in RAM o è così, traduce l'indirizzo in binario, altrimenti segnala page fault. Per gestire il page fault, a più:

- 1- ricarica la pagina in RAM e di riempierla;
- 2- il frame viene etichettato come busy;
- 3- se nel memory, si salva la vittima su disco;
- 4- si carica la pagina del page fault in RAM (altrimenti passa la CPU a un altro processo);

5- si rimanda la tabella delle pagine e si rimette il processore.

SPOSTAMENTO RAM-DISCO

Le operazioni swap-in permettono l'inserimento in RAM di una pagina, solitamente avvenuta su richiesta, durante un page fault o durante il prepaging del SO.

Le operazioni swap-out avvengono quando si deve liberare dello spazio per nuove pagine, tenendo traccia del dirty bit d'ogni pagina, alle pagine modificate dell'ultimo scambio e copiandone quindi quelle con dirty bit uguali a 1.

RIMPIAZZAMENTO DELLE PAGINE

- algoritmo di rimpiazzeramento dove diminuire il numero di page fault ogni volta che ne incontra una, di seguito delle possibili alternative: Optimal, Least Recently Used, Not Recently Used, FIFO e orologio

In un sistema multrogrammato può esistere il rimpiazzeramento:

- locale: ad ogni processo viene assegnato un certo numero di frame e, quando avviene page fault, il rimpiazzeramento viene fatto solo in questo processo.

- globale: il rimpiazzeramento viene fatto su tutte le pagine in memoria.

Dato che bisognerebbe avere in memoria l'intero WS del processo, è possibile ottenerlo limitando un certo numero equivalente all'insieme delle pagine nell'ultimo riferimento; altrettanto finisce un intervallo di tempo δ , il WS diventa quindi l'insieme delle pagine in un determinato intervallo.

ALGORITMO OTTIMALE

L'algoritmo ottimale richiede i riferimenti futuri. Le pagine vanno sostituite finché non si riempie la RAM e il frame del processo. Al primo riferimento dove è necessario un rimpiazzeramento viene scelta la pagina che verrà usata nel più lontano futuro.

ALGORITMO LRU

Viene scelta la pagina utilizzata nell'ultima volta nel più lontano passato. Questo perché una pagina "recente" ha più probabilità di una pagina più vecchia.

ALGORITMO NRU

L'algoritmo NRU sceglie come ultima la pagina meno utilizzata, scegliendola in ordine crescente a partire dalla fine. Si memorizzano un certo numero di bit per pagina, ponendosi progressivamente a destra. Infine si sceglie il valore minore.

ALGORITMO FIFO

Le pagine vengono mantenute in memoria e ordinate in base all'istante di caricamento. Al rimpiazzeramento viene scelta la testa della lista.

ALGORITMO SECONDA POSSIBILITÀ

È una variante del FIFO che utilizza un ulteriore bit, se è uguale a 1, la testa viene postata in fondo alla lista e si azzerà il bit, altrimenti diventa la vittima.

ALGORITMO OROLOGIO

È una semplificazione della seconda possibilità in cui vi è una lista circolare, invece quando il bit è uguale a 1, si azzerà e si farà al modo successivo.

THRASHING

Il thrashing succede quando la presenza contemporanea di più processi supera la capacità della RAM, i risultati sono un alto tasso di page fault, basso utilizzo della CPU, alto utilizzo del disco. Quando vi è thrashing, tutti i processi rallentano perché devono gestire i page fault. Per evitarlo, occorre monitorare i page fault e le dimensioni del WS e, se necessario, fare swap-out di uno o più processi.

ALGORITMI BASATI SU WORKING SET

Utilizzando il working set dei processi, è possibile cercare di evitare il thrashing e non fare swap out.

Le pagine del WS sono definite da un intervallo di tempo, se il tempo dell'ultimo riferimento della pagina è minore dell'intervallo, essa è nel WS. In caso di page fault, si cerca di non rimpiazzare le pagine del working set con l'algoritmo dell'orologio (mobilista).

Se tutte le pagine sono nel WS allora se ne sceglie una, possibilmente con dirty bit a 0.

Se il page fault è elevato è spesso vero di trovarsi pagine del WS allora conviene farci swap out per prevenire il thrashing. La grandezza dell'intervallo è direttamente proporzionale alla grandezza del WS.

ALGORITMO DELL'OROLOGIO MODIFICATO

Funziona allo stesso modo di quello normale, solo che, quando il bit è 0, rimpiazza la pagina perché non appartiene al WS.

POLITICHE GLOBALI E LOCALI

Una politica è globale quando si può assegnare un numero di frame variabile nel tempo, tenendo conto del page fault. Se supera il numero di frame, occorre aggiungere altri; altrimenti glieli puoi togliere.

PAGE DAEMON

Il paging daemon è un processo in background che periodicamente controlla la situazione dei frame in memoria. Quando non ci sono pagine in memoria, ne seleziona una come forse il page fault handler, se è vuoto invece la prende dal disco.

Se la pagina è idonea per il rimpiazzamento, può essere recuperata se è riferita prima di essere rimpiazzata.

PROBLEMI D'INTERAZIONE GESTIONE MEMORIA E I/O

La gestione della memoria interagisce con quella dell'I/O quando:

- dato un processo che richiede la lettura dal disco, se è in waiting per l'I/O, un altro processo diventa pmm quest'ultimo cerca page fault e potrebbe venire rimpiazzata la pagina del primo processo.

Per evitare ciò, occorre bloccare alcune pagine in RAM oppure fare l'I/O in buffer del SO e copiare tra memoria di sistema e utente.

PAGINE CONDIVISE

Le pagine condivise sono solitamente utilizzate per la creazione di un nuovo processo in cui viene fatta una copia della sua immagine solare (copia modificare pagina read-only) (quando quindi una tra due). Durante la gestione, la pagina viene copiata in due pagine read-write (una per processo) e viene fatta riferire l'indirizzo.

OVERHEAD

L'overhead: calcolato dalla page table e dalla frammapinta, si misura in questo modo:

$$\text{overhead} = \frac{\text{dim Proc} \cdot \text{dim PageTable}}{\text{dim Page}} + \frac{\text{dim Page}}{2}$$

page table frammapinta

L'overhead è minimo quando $\text{dim Page} = \sqrt{2 \cdot \text{dim Proc} \cdot \text{dim PageTable}}$

MMU

il MMU si appoggia al Translation Lookaside Buffer, una memoria associativa che riuscirà a ricercare un numero di pagine in parallelo, tutti gli elementi. Può essere definita come una cache della page table, ogni elemento non trovato viene messo al posto di altro. Dov'è messo invalidato dal contest switch.

PAGE TABLE A DUE LIVELLI

La page table a sua volta suddivide in pagine. In un grande spazio d'indirizzi logici, programma e dati stanno da una parte mentre lo stack dall'altra. Al centro vi è molto spazio inutilizzato.

INTERVENTO DEL SO NELLA GESTIONE DELLA MEMORIA

Il SO interviene quando:

- Viene creato un nuovo processo, determinando e creando la sua memoria e creando la page table.

- Un processo va in running, impostando MMU per il nuovo processo e invalidando TLB.

c'è page fault, facendo il riempimento, se necessario, fare mapping della pagina scelta con quella richiesta.

Un processo termina, rilasciando la page table e i frame occupati.

	PAGINAZIONE	SEGMENTAZIONE
TRASPARENTE	SI	NO
# SPAZIO D'INDIRIZZI LINEARI	1	#SEGMENTI
POSSO SEPARARE I DATI DAL CODICE	SI, NON BISOGNA ALLEGARE CODICE E DATI IN UNA PAGINA	SI
CONDIVIDERE PROCEDURE TRA PROCESSI	SI, GESTENDO TUTTE LE PAGINE	SI, IN MODO "NATURALE"
LO SPAZIO TOTALE PUÒ SUPERARE LA RAM	SI	SI
MOTIVAZIONE	RENDERE DISPONIBILE UN AMPIO SPAZIO DI SUDDIVIDERE CODICE E DATI D'INDIRIZZI AI PROCESSI IN UNITÀ LOGICHE FACILITANDO PROTEZIONE E CONDIVISIONE.	

SEGMENTAZIONE

Segmentazione è una tecnica, ortogonale alla paginazione, attribuita al programma il cui spazio d'indirizzamento è bidimensionale. Cogni programma può avere uno o più segmenti di codice e dati, in ciascuno di essi è possibile definire quali operazioni ammettere sulle informazioni contenute.

La gestione della memoria dovrà quindi occuparsi di allocare i segmenti d'ogni processo in memoria. Tutti i segmenti in memoria hanno informazioni nella tabella dei segmenti gestita dal rispettivo processo.

TRADUZIONE DEGLI INDIRIZZI

MO traduce l'indirizzo utilizzando le informazioni della tabella dei segmenti (ci possono essere diverse validità per accedere ai segmenti).

SEGMENTAZIONE PAGINATA

Questa tecnica è una combinazione delle prime due, suddividendo ogni segmento in pagine di dimensione prefissata, gestendo una page table per ogni segmento.

