CPU

registri generali memorizzano variabili e risultati temporanei.

registri di controllo servono per controllarne il funzionamento.

CPU – registri di controllo:
• Program Counter (PC): indirizzo istruzione da prelevare.
• Stack Pointer (SP): puntatore al top dello stack contenente i frame delle procedure già iniziate ma non ancora terminate.
• Program Status Word (PSW):
* Privileged Mode (PM): bit di modalità user/kernel
* Condition Code (CC): codici di condizione impostati da istruzioni di confronto
* Interrupt Mask (IM): determinano le classi di interrupt abilitate e non abilitate.
* Interrupt Code (IC): serve per comunicare al S.O. le cause dell'interrupt
* Memory Protection Information (MPI) – informazioni sulla porzione di memoria accessibile

INTERRUPT

Gli interrupt costituiscono un meccanismo per notificare alla CPU un evento o una condizione avvenuti nel sistema che devono essere trattati dal S.O. e interrompe l'esecuzione del programma;

Hardware interrupt.

Si tratta di eventi asincroni rispetto al programma in esecuzione: il programma non ha modo di prevedere se e quando questi eventi si verificano.

-I/O interrupt: usati dai dispositivi I/O per notificare eventi alla CPU che richiedono di essere trattati dal S.O.
-Timer interrupt.

Fasi dell'interrupt hardware :

1. La CPU, mentre sta eseguendo l'istruzione i del programma P, riceve un interrupt request
2. la CPU sospende l'esecuzione di P e salta alla procedura di gestione dell'interrupt, detta interrupt handler;
3. l'interrupt handler, che è parte del S.O., gestisce l'interrupt :
    -memorizza i dati di PC,SP e PSW nella SIRA
    -esegue le istruzioni che deve eseguire
    -ripristina i dati di PC,SP e PSW
4. l'interrupt handler restituisce il controllo a P(dipende dalle politiche di scheduling);
5. P riprende l'esecuzione prelevando l'istruzione i+1, come se nulla fosse accaduto.

Gerarchia tipica degli interrupt hardware:
- machine error
- clock interrupt
- disk interrupt
- fast device interrupt
- slow device interrupt

Software interrupt

detti anche trap,Sono usati dai programmi per chiedere esplicitamente l'intervento del S.O.
Sono causati da un ' istruzione trap.
(gerarchia interrupt uguale a hw interrupt)

system call

Una system call è una richiesta al S.O. effettuata da un programma. Vengono realizzate con la TRAP

system call in unix: -create, -open,-read-write,-close

invocare una system call

1. Se programmiamo in assembly usiamo l'istruzione TRAP.
2. Se programmiamo in C abbiamo una libreria di funzioni associate alle system call.

program interrupt:

(sincroni all'esecuzione del programma):
Eccezioni, dette anche interrupt interni:
Sono eventi causati da situazioni anomale interne al programma, che devono essere trattati dal S.O..

DISPOSITIVI DI I/O E DMA

Dispositivi di I/O. Distinguiamo tra:

-Controller (detto anche adapter): componente elettronica che comunica con la CPU (e le altre unità) via bus
-Dispositivo vero e proprio: componente meccanica che viene gestita dal controller

L'interfaccia tra CPU e controller viene usata dai driver Questa interfaccia prevede:
• registri di controllo, detti anche porte di I/O
• un buffer: serve per memorizzare dati durante le operazioni di I/O.

soluzioni per le comunicazioni tra la CPU e le porte di I/O:
-porte di I/O gestite con istruzioni macchina ad hoc
- Memory Mapped I/O

soluzioni per eseguire l ' I/O per conto di un programma P:

1 – Programmed I/O (abbiamo del busy waiting)
2 - Interrupt Driven I/O (overhead)
3- Direct Memory Access (DMA) capace di accedere direttamente alla memoria e di lavorare in parallelo con la CPU.

PROCESSI

MEMORY LAYOUT DEI PROGRAMMI

Regioni(indirizzi virtuali):

• Area di testo. Contiene il testo del programma
• Area dati. Contiene le variabili globali del programma
• Area di stack. frame delle procedure già chiamate ma non ancora terminate.
• area heap

IL CONCETTO DI PROCESSO

un processo è un'istanza di un programma in esecuzione

Possiamo avere più processi relativi al medesimo programma, Ognuna di queste esecuzioni non deve interferire con le altre, anche se si sovrappongono dal punto di vista temporale,

Il programma è un'entità passiva che non esegue nessuna azione di per sé.

Le entità che vengono schedulate dal S.O. sono pertanto i processi,

Le aree di testo/dati /stack , le risorse e lo stato della CPU sono associate al processo,

PARALLELISMO E CONCORRENZA

Due eventi sono paralleli se occorrono nello stesso momento La concorrenza è l'illusione del parallelismo

Interleaving= assegnamento CPU "a turno" ai vari processi

La velocità di avanzamento di un processo non è uniforme nel tempo e non è riproducibile.

• Multiprocessing: più processi possono eseguire su una macchina dotata di più CPU.
• Distributive Processing: più processi possono eseguire su più macchine distribuite ed indipendenti.

In questi casi il parallelismo può essere reale,

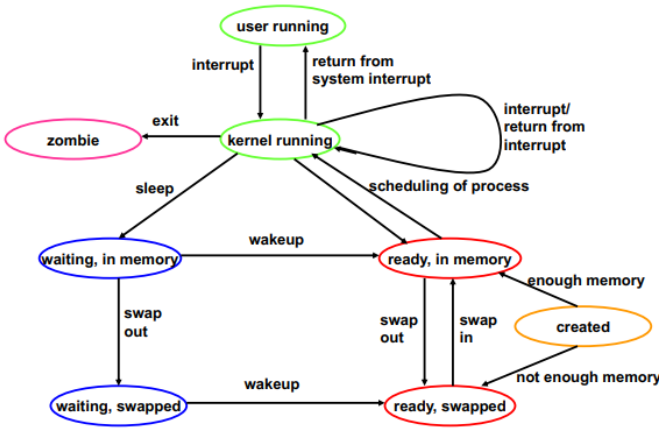
STATI DI UN PROCESSO

Lo stato è un indicatore dell'attività che sta svolgendo il processo.

• running: il processo è in esecuzione,;
• ready: il processo non è in esecuzione, ma sarebbe in grado di eseguire se ottenesse la CPU;
• waiting(blocked o sleeping): il processo non è in esecuzione, non sarebbe in grado di eseguire se ottenesse la CPU,

transizioni possibili:
- ready à running: lo scheduler seleziona il processo per l'esecuzione.
- running à ready: il processo subisce una preemption o scadenza del time slice.
- running à waiting: il processo si trova impossibilitato ad eseguire. Si blocca in attesa di un evento sbloccante
- waiting à ready: si verifica l'evento sbloccante atteso dal processo

Stati Unix system V



IMPLEMENTAZIONE DEI PROCESSI – PROCESS CONTROL BLOCK

Il S.O. tiene traccia di tutti i processi presenti nel sistema, avvalendosi di apposite strutture dati.

Il S.O. mantiene una process table, con una voce, Process Control Block (PCB), per ogni processo.

i PCB sono nella kernel area

Voci certamente presenti in ogni PCB:
• identificativo del processo
• stato del processo
• stato della CPU
• parametri da usare per lo scheduling
• puntatore per accedere all' area di testo/dati/stack
• puntatore per accedere a file/dispositivi aperti
• un PCB pointer

Page fault: è un 'eccezione causata da un accesso ad una pagina di memoria virtuale che non è nella memoria fisica

Swap out: Il S.O. effettua ad intervalli regolari degli swapping out delle pagine meno utilizzate per "liberare memoria".

IMPLEMENTAZIONE DEI PROCESSI – CONTEXT SWITCH

il contesto di un processo (process context), detto anche ambiente, è costituito da:
• register context: il contenuto dei registri della CPU;
• user-level context: aree testo, dati e stack;
• system-level context (inaccessibile dal programma user)

Il S.O. effettua 3 operazioni fondamentali sui processi:
• Context save: quanto il processo running va in ready o waiting, il S.O. salva nel PCB tutti i dati necessari a far ripartire il processo in futuro.
• Scheduling: tenendo conto della politica di scheduling, il S.O. sceglie un processo tra quelli in ready per l'esecuzione.
• Dispatching: dopo che un processo viene schedulato, il S.O. deve ripristinarne il contesto sfruttando i dati salvati nel PCB al momento del context save.

Si parla di context switch quando il S.O. effettua il context save per un processo P, schedula un processo P' diverso da P e ne effettua il dispatching

CREAZIONE E TERMINAZIONE DI PROCESSI

La system call exec serve per "cambiarsi il programma ".

La system call wait è una richiesta al S.O. di mettersi in stato di WAITING

Quando un processo termina, esegue la system call exit e va in stato ZOMBIE.

Tipi di processo:
-user process sono associati ad un utente che lavora ad un terminale
-daemon process non sono associati a nessun utente, non terminano, eseguono funzioni vitali per il sistema, eseguono in modalità user.
-kernel processes sono daemon che eseguono in modalità kernel.

Circostanze che causano la terminazione di un processo:
•system call per terminarsi o terminare altri processi
• interrupt handler delle eccezioni può forzare la terminazione del processo.

SYSTEM CALL IN UNIX

Oltre al PCB, UNIX assegna ad ogni processo una user area,
Il PCB ha un pointer alla user area
I PCB sono allocati in una Process Table

La user area serve anche per memorizzare i parametri, il risultato e l'eventuale codice di errore delle system call.

THREAD

Se il linguaggio offre i costrutti la programmazione concorrente

si può guadagnare in termini di:
• semplicità di programmazione;
• efficienza di esecuzione

programmando l'applicazione con più processi l'attività di programmazione è più semplice, ma si rischia di non guadagnare in efficienza.

Soluzione: thread

Un thread è un'esecuzione di un programma che usa le risorse di un processo.

Un processo può avere più thread. Possibile implementazione:
1. Aree testo e dati e risorse logiche e fisiche sono condivise;
2. Ogni thread ha i propri identificatore (Thread IDentifier, TID), CPU state, stack e stato

Per ogni thread è necessario avere un Thread Control Block che contiene:
- identificatore (Thread IDentifier, TID),
- CPU state,
- stack
- stato

Il thread switching tra due thread dello stesso processo è piu efficiente perché:

• non vanno aggiornati i dati della MMU relativi alle aree testo e dati;
• la cache della memoria e del disco richiedono meno aggiornamenti;

La creazione di un thread è più veloce della creazione di un processo, perché il TCB ha meno dati del PCB

due processi possono scambiarsi informazioni mediante invio e ricezione di messaggi, implementabili con system call.

Thread implementati nello spazio kernel:

1. il S.O. gestisce i thread;
2. i TCB sono strutture dati gestite dal S.O.;
3. il S.O. offre system call per creare/terminare thread,
4. lo scheduler del S.O. deve schedulare thread anziché processi;
5. il programmatore maneggia i thread chiamando le funzioni di libreria wrapper di system call.

Thread implementati nello spazio user:

1. il S.O. non sa nulla dei thread, gestisce solo processi,
2. TCB sono strutture dati gestite dalla libreria di procedure;

3. esistono procedure per creare/terminare thread, per far cambiare loro il programma,
4. lo scheduler del S.O. schedula processi, una delle procedure della libreria è lo scheduler delle thread;
5. il programmatore maneggia i thread usando le procedure della libreria.

Vantaggi:

• funziona anche su S.O. che non implementano i thread.
• il thread switching non richiede l'intervento del S.O., quindi è più veloce;
• l'algoritmo di scheduler può essere personalizzato.

Svantaggi:

• se un thread fa una system call che manda la thread in waiting, per il S.O. va in waiting il processo, quindi si bloccano tutte i thread.

PROCESS SYNCHRONIZATION

INTERAZIONI TRA PROCESSI

Su un sistema uni-processore, programmare un 'applicazione con un insieme di processi/thread concorrenti può dare i seguenti vantaggi:

• semplicità di programmazione;
• efficienza;
• possibilità di assegnare compiti più urgenti a processi/ thread con priorità più elevata

Abbiamo 4 tipi di interazioni tra processi:

• data sharing: i processi condividono i dati della memoria condivisa ed alcuni file.
• control synchronization: un'azione ai di un processo Pi è abilitata solo dopo che un altro processo Pj ha svolto un azione aj .
• message passing: un processo Pi invia un messaggio ad un processo Pj che lo riceve.
• signals: un processo Pi invia un segnale ad un altro processo Pj per segnalare una situazione particolare.

Due processi concorrenti Pi e Pj sono processi interagenti se vale almeno una delle seguenti due proprietà:
• Ri e Wj hanno intersezione non vuota,
• Rj e Wi hanno intersezione non vuota.

Due processi concorrenti Pi e Pj sono processi indipendenti se non sono interagenti.

Se due processi/thread concorrenti Pi e Pj sono indipendenti:

• competono per le risorse rallentandosi a vicenda,
• i loro comportamenti non dipendono dalla loro velocità relativa e sono riproducibili.

Se due processi/thread Pi e Pj sono interagenti:

• competono per le risorse ,rallentandosi a vicenda,
• i loro comportamenti dipendono dalla loro velocità relativa e non sono riproducibili.

RACE CONDITIONS

Le race condition hanno due conseguenze:

• il comportamento dei processi/thread non è corretto;
• i dati su cui ha luogo la race condition diventano inconsistenti, cioè assumono uno stato non previsto.

Prevenire la race condition:
- mutua esclusione= quando un processo accede ad un dato condiviso d, nessun altro processo deve poter accedere concorrentemente a d.

SEZIONI CRITICHE

una sezione critica (critical section - CS) per un dato condiviso d è una porzione di codice che viene certamente eseguita non concorrentemente con se stessa o con altre sezioni critiche per d

Proprietà richieste alle implementazioni delle CS:
• correttezza: le CS per un dato d non possono essere eseguite concorrentemente;
• progress: se nessun processo sta eseguendo una CS per d, e alcuni processi manifestano la volontà di eseguire CS per d, allora uno di essi deve poter eseguire la propria CS per d;
• bounded wait: dopo che un processo Pi manifesta la volontà di accedere ad una CS per d, il numero di accessi alle CS per d da parte di un qualsiasi altro processo Pj che precedono l'accesso di Pi deve essere <= di un dato intero k

Implementare le CS:
-tentativi errati:
1 – disabilitare gli interrupt: corretta solo su sistemi uniprocessore e non garantisce la proprietà progress;
2 – uso di variabili lock condivise: non è corretta, perché la race condition può aver luogo sulla variabile lock
3 – uso di variabili turno condivise: Questa soluzione è corretta ma non soddisfa la proprietà del progresso

un'operazione indivisibile(o atomica) su un dato condiviso d è un'operazione che è con certezza eseguita in modo non concorrente rispetto ad altre operazioni su d.

-tentativi giusti:

1- Istruzione Test and Set (TS instruction o TSL – test and set lock) L'istruzione TS ha uno oppure due parametri
2- Istruzione swap: istruzione indivisibile che scambia il contenuto di due locazioni di memoria.

implementare le CS a alto livello:

1. approccio algoritmico:i processi effettuano controlli prima di eseguire le CS al fine di garantire la mutua esclusione
2. uso di primitive software
3. uso di costrutti ad hoc per la programmazione concorrente.

Le soluzioni 2 e 3 si vengono realizzate sulla base delle istruzioni indivisibili, quali la TS e la SWAP.

APPROCCIO ALGORITMICO ALLE SEZIONI CRITICHE

L'approccio algoritmico prevede di implementare le CS senza usare istruzioni hardware ad hoc, system call o costrutti del linguaggio ad hoc

Approccio algoritmico alle CS idee funzionanti:

- Uso delle variabili turno, algoritmo di Dekker. Soluzione vincente, ma valida solo per 2 processi.
- Uso delle variabili turno, algoritmo di Peterson. Soluzione vincente, ma valida solo per 2 processi.

Se i processi sono N>2:

• l'algoritmo di Eisenberg e McGuire, che estende al caso N l'algoritmo di Dekker;
• il Bakery algorithm .

MONITOR

Il monitor è un costrutto offerto da alcuni linguaggi

Un tipo monitor consiste di:

• dichiarazione di variabili;
• inizializzazione delle variabili;
• procedure che usano le variabili

GESTIONE DELLA MEMORIA

Obiettivi del S.O.:

1. Allocare più processi in memoria, per avere parallelismo .
2. Proteggere la memoria dei processi, impedendo ai processi di accedere alle aree degli altri processi.

Traduzione, linking, loading

• Source program
• Object module: programma in linguaggio macchina con riferimenti a funzioni di librerie.
• Binary program: programma eseguibile.
• Binary program in memoria: programma eseguibile in memoria.

rilocazione statica da parte del loader: il loader modifica gli indirizzi del programma prima che il programma venga eseguito.

il linking dinamico, il linker statico sostituisce un riferimento con la logica per effettuare il calcolo dell'indirizzo durante l'esecuzione del programma.

Modello di allocazione della memoria

Durante l'esecuzione del programma, vengono allocati due tipi di dati:

• le variabili il cui scope è associato a blocchi, funzioni, procedure.
• i dati creati dinamicamente con i costrutti appositi,

heap allocator= Possiamo chiamare heap allocator/deallocator le routine che si occupano di allocare/deallocare memoria nello heap. Queste routine fanno parte del RunTime Support (RTS) del linguaggio,

Modello di allocazione della memoria usato dal S.O.:
-il codice ed i dati statici sono allocati in un'area di dimensione statica,
-l'area di heap e lo stack condividono un'area di dimensione statica, ma le due aree hanno dimensione variabile e "crescono" in direzione opposta.

Gerarchia di memoria

i sistemi di computazione offrono una gerarchia di unità di memoria con velocità diverse: le unità più veloci e costose sono quelle di minor capacità.

• La CPU accede alla gerarchia di memoria tramite la MMU (Memory Management Unit), che traduce gli indirizzi logici in indirizzi fisici.

• Il byte che si trova all'indirizzo fisico della main memory generato dalla MMU viene in realtà cercato nella cache. Se non viene trovato, allora è necessario trasferirlo dalla main memory alla cache.

• Se il sistema offre la memoria virtuale, una parte delle aree di memoria del processo non si trovano nella main memory ma sul disco. Gli indirizzi fisici generati dalla MMU sono virtuali.

RIUSO DELLA MEMORIA

L'allocazione della memoria è un problema che possiamo vedere a due livelli:

- il S.O. deve allocare memoria per i processi;
- il RTS deve allocare memoria per i dati PCD.

In entrambi i casi è necessario riutilizzare la memoria liberata. Per farlo, è possibile tener traccia della memoria libera in una free list.

Tecniche per allocare area k byte con free list:

• first fit: si cerca la prima area libera di dimensione >=k. Se l'area ha dimensione d, si allocano k byte e l'area rimanente di d-k byte rimane nella free list; (c'è frammentazione)
• best fit: si cerca l'area libera più piccola tra quelle di dimensione >=k. Se l'area ha dimensione d, si allocano k byte e l'area rimanente di d-k byte rimane nella free list; (è poco efficiente e dopo un po' c'è frammentazione)

• next fit: si adotta la first fit partendo, però, dal punto in cui era stata effettuata l'ultima allocazione.(compromesso)

Frammentazione= aree di memoria non utilizzabili in un sistema di computazione

Tecniche per prevenire/contrastare la frammentazione:
• boundary tag = all'inizio ed alla fine di ogni area c'è un tag, contenente la coppia le aree libere sono collegate da una catena di pointer quando un'area occupata viene liberata, se è possibile la si unisce ad un'area libera confinante
• memory compaction= le aree libere sono collegate da una catena di pointer a determinati intervalli di tempo tutte le aree libere vengono unite in un'unica area libera.

ALLOCAZIONE DELLA MEMORIA

allocazione contigua della memoria quando ogni processo è allocato in una singola area di memoria contigua.
La rilocazione dei processi è semplice: è sufficiente disporre del registro RR
L'allocazione contigua può essere integrata con lo swapping
Allocare in memoria un processo che richiede k byte è possibile solo se è disponibile un'area di memoria dimensione >=k.

allocazione non contigua della memoria quando ogni processo può essere allocato in più aree di memoria non adiacenti.
Ogni porzione del processo che viene allocata in un'area contigua è detta componente
Allocare in memoria un processo che richiede k byte non necessita di una singola area di memoria di dimensione >=k
Il problema della frammentazione è risolto, oppure limitato

Abbiamo due tecniche principali per l'allocazione non contigua:

• paginazione:

-La memoria è divisa in 2^f page frame, ciascuno di capacità 2^s byte, dove 2^f+s è la dimensione della memoria.
-Ogni processo è logicamente diviso in pagine aventi le medesime dimensione dei page frame.
-Allocare un processo in memoria significa associare un page frame ad ogni sua pagina.
-Non esiste frammentazione

-Per ogni processo il S.O. mantiene una page table, che tiene traccia del page frame in cui è allocata ogni pagina.
-Il S.O. deve tener traccia dei page frame liberi per allocare i processi. Di norma vengono organizzati in una free frame list.

• segmentazione:

-Ogni processo è logicamente diviso in segmenti
-Ogni segmento di ogni processo ha la propria dimensione e, pertanto, NON può avere un corrispondente segment frame in memoria.
-Ognuno dei byte dello spazio del processo è individuato da un indirizzo composto da due dati: il numero del segmento e l'offset nel segmento.
-Allocare un processo in memoria significa associare un'area di memoria libera ad ogni suo segmento
-E' possibile la frammentazione
-La segmentazione è più adatta a facilitare la condivisione di moduli di codice, strutture dati, oggetti
-Per ogni processo il S.O. mantiene una segment table
-Non ha senso avere una segment frame list, non essendoci segment frame

Segmentazione + paginazione:

• I segmenti sono entità logiche divise in pagine che hanno la stessa dimensione dei page frame.
• Ogni entry della segment table contiene un pointer alla page table del segmento.
• Si hanno i vantaggi di entrambe le tecniche.

MEMORIA VIRTUALE

memoria virtuale quando il S.O. crea l'illusione che il sistema abbia più memoria di quella effettiva.

L'obiettivo è di rendere i processi indipendenti dalla capacità di memoria del sistema.
L'implementazione si basa sull'uso del disco.

La memoria virtuale si basa sulla paginazione (demand paging) e/o segmentazione.

Demand paging:

c1:
-la memoria è divisa in page frame ed i processi sono costituiti da pagine
-l'intero spazio logico dei processi è memorizzato su un paging device,
-l'area del paging device allocata per un processo viene detta swap space del processo
-quando un processo inizia l'esecuzione, viene allocato solo un page frame per la pagina che contiene la prima istruzione

c2:
-quando il processo tenta di accedere ad una pagina cui non è assegnato nessun page frame, la pagina viene copiata dallo swap space in un page frame libero;
-quando un processo modifica una pagina, l'immagine della pagina sullo swap space è obsoleta;
-se servono page frame, il S.O. può liberare i page frame associati alle pagine dei processi, in tal caso le immagini di tali pagine sullo swap space vanno aggiornate.

Concetti demand paging:

• page fault: eccezione che viene sollevata dalla MMU quando un processo tenta di accedere ad una pagina che non è associata a nessun page frame;
• page-in: in seguito al page fault, il S.O. (page fault handler) copia la pagina dallo swap space in un page frame libero.
• page-out: quando il S.O. libera un page frame, se la pagina associata è stata modificata dopo l'ultimo page-in, deve essere copiata sullo swap space;

• page replacement: consiste nel liberare un page frame e nel successivo page-in di una pagina diversa.

Traduzione con uso del TLB:
-prevede due cicli di memoria per ogni accesso alla memoria
-Per risparmiare uno dei due cicli, la MMU può usare un Translation Look-aside Buffer (TLB),

la MMU genera l'indirizzo fisico a partire dall'indirizzo logico nel modo seguente:
1. usando la entry del TLB dedicata alla pagina, se esiste;
2. sfruttando la entry relativa alla pagina sulla page table, se tale entry ha validity bit 1 e se il caso 1) non è applicabile;
3. generando un page fault se i casi 1) e 2) non sono applicabili

la MMU accede alla page table del processo running; in memoria sono presenti le page table di tutti i processi; per implementare facilmente questo aspetto l'architettura può offrire il Page Table Address Register (PTAR), contenente l'indirizzo della page table del processo running;

Protezione della memoria:

• se un processo tenta di accedere ad un indirizzo al di fuori del proprio spazio logico, viene generata un memory protection exception. il controllo può essere fatto facilmente se l'architettura offre un Page Table Size Register (PTSR) che memorizza il numero dell'ultima pagina del processo running.

• il campo prot della page table codifica le protezioni in lettura/ scrittura delle pagine se l'architettura fornisce un adeguato supporto, il valore del campo può essere confrontato con il tipo di accesso che viene effettuato se il tipo di accesso non è consentito, la MMU genera una memory protection exception.