3. Si spieghino i concetti di linking statico e loading statico.
3. Si spieghi il ruolo e il funzionamento della IPT.
Che cos'è una PCB e qual è il suo ruolo?
si descriva in <= 30 righe di testo il demand paging
Descrivere gli stati di un processo e le transizioni.
s3.Spiegare la paginazione e anche come la MMU traduce i bit. (file teoria martitna)
Descrivere il funzionamento del TLB nel contesto paging
Cosa è il tlb e quale è il suo ruolo

1. cosa è la multiprogrammazione, come gestisce la memoria, i dispositivi e i requisiti hardware

Più job vengono caricati in memoria che viene partizionata in modo da poterli caricare contemporaneamente. Quando un job inizia un'operazione di I/O, la CPU non rimane inattiva, ma viene assegnata ad un altro job. Perché ciò sia possibile, l'hardware deve consentire a CPU e dispositivi di lavorare in parallelo.

CPU SCHEDULING quando un job in esecuzione avvia un'operazione I/O, la CPU deve essere assegnata ad un job tra quelli in memoria non impegnati in attività di I/O. E' necessaria una politica di scheduling per scegliere il job da eseguire.

GESTIONE DELLA MEMORIA ogni job deve poter accedere ai propri dati, ma non a quelli degli altri. Un esempio per la protezione prevede che la CPU sia dotata di due registri: Lower Bound Register(LBR) e Upper Bound Register(UBR). In primis la CPU controlla che l'indirizzo interessato sia compreso tra gli indirizzi contenuti nei due registri, se non lo è si genera un segnale hardware che ferma il programma.

GESTIONE DEI DISPOSITIVI ogni job deve poter usare i dispositivi, senza interferire sull'uso degli stessi da parte degli altri job. Ci sono due modalità: Partitioning (ad ogni job vengono assegnati i dispositivi staticamente) e Pooling (ad ogni job vengono assegnati dispositivi dinamicamente)

REQUISITI HARDWARE l'hardware deve consentire alla CPU ed ai dispositivi di I/O di lavorare in parallelo, sfruttando il DMA (Direct Memory Access, cioè accesso diretto alla memoria), che permette ai dispositivi di accedere alla RAM senza passare dalla CPU. Quando un dispositivo di I/O ha terminato di eseguire un'operazione per conto di un job, esso deve essere inserito nell'elenco dei job schedulabili, ricorrendo alla tecnica dell'**interrupt**.

1. THROUGHPUT E CLASSIFICAZIONE DEI PROGRAMMI

Il Throughput è il rapporto tra numero di programmi eseguiti e il tempo per eseguirli. Un programma si dice:

- CPU-bound se fa uso intenso di CPU e poco I/O

- I/O-bound se effettua tanto I/O e poco CPU

Per ottimizzare il throughput, lo scheduler deve privilegiare i programmi I/O-bound rispetto a quelli CPU-bound.

Quindi in un sistema multiprogrammato si introducono i concetti di **program priority** (SO assegna ad ogni programma una priorità) e **preemption** (CPU sottratta forzatamente dal programma in esecuzione).

Per migliorare il throughput si assegna ai programmi CPU-bound una priorità minore rispetto a quelli I/O-bound.

3) cosa è lo SPOOLING

È un meccanismo che estrae i singoli job dal batch: i programmi possono essere caricati su disco; appena un job termina e libera una partizione di memoria, il SO la assegna a uno dei job su disco che può essere eseguito; per migliorare il throughput, conviene avere in memoria sia programmi CPU-bound che I/O-bound; anche i dispositivi di output possono essere gestiti in modo analogo. SPOSTAMENTO DI DATI IN UNA ZONA DI MEMORIA DOVE POI VERRANNO SMISTATI.

4) TIMESHARING

È un'estensione della multiprogrammazione in cui ad ogni job viene assegnato un quanto di tempo (time slice) allo scadere del quale la CPU viene assegnata ad un altro job anche in assenza di I/O o preemption. È adatto per sistemi dotati di più terminali interattivi: ogni utente ha l'impressione di interagire direttamente e continuamente con la macchina. Lo scheduler non può essere basato sulle priorità, ma deve assegnare la CPU equamente a tutti i job, eseguendo ciascuno di essi a turno (round robin). Il SO deve garantire anche la protezione dei dati di ciascun utente. Il timesharing penalizza il throughput, ma migliora i tempi di risposta per gli utenti interattivi, privilegiando la user convenience.

5) LSI/VLSI

LSI significa "Integrazione su larga scala". È una tecnologia di progettazione dei circuiti integrati (IC) in cui un chip può contenere centinaia o migliaia di transistor.**Caratteristiche**: Risale agli anni '70. Permette di implementare funzioni relativamente complesse, come memorie RAM, ROM e microprocessori semplici. Usata in applicazioni che richiedono complessità moderata, come calcolatori tascabili, controller e periferiche.ESEMPI circuti per calcolatrici. VLSI significa "Integrazione su scala molto larga". Con questa tecnologia, milioni o addirittura miliardi di transistor possono essere integrati su un singolo chip. **Caratteristiche**: È stata sviluppata negli anni '80 e successivi. Consente di realizzare sistemi complessi su un unico chip, come processori moderni, memorie avanzate e sistemi su chip (**SoC**). È alla base dell'elettronica moderna, inclusi smartphone, computer e dispositivi IoT.**Esempi**: Microprocessori moderni (come quelli Intel, AMD, o ARM). Memorie DRAM e NAND flash.

6) cosa è la cpu

Il SO interagisce direttamente con l'hardware.

La CPU preleva le istruzioni dalla memoria(fetch), le decodifica(decode) e le esegue(execute). La CPU è dotata di registri:

- Generali: costituiscono il primo livello della gerarchia di memoria: essi contengono i dati e i risultati temporanei su cui la CPU lavora
- Di controllo controllano lo stato e funzionamento della CPU stessa:

- Program Counter (PC): l'indirizzo dell'istruzione da prelevare
- Stack Pointer (SP): l'indirizzo della cima dello stack in cui sono contenuti i frame delle procedure
- Program Status Word (PSW): un insieme di uno o più registri che è suddiviso in gruppi di bit con significati particolari, cioè Privileged Mode (PM, bit di modalità user/kernel), Condition Code (CC, codici di condizione impostati dall'ALU), Interrupt Mask (IM) e Interrupt Code (IC, gestire gli interrupt), Memory Protection Information (MPI, info sulla porzione di memoria accessibile).

7) cosa sono gli interupt e come si dividono

costituiscono un meccanismo per segnalare alla CPU un evento o una condizione avvenuti nel sistema che devono essere trattati dal SO. Gli obbiettivi sono due:

1. Interrompere il normale ciclo di esecuzione della CPU, che di norma continuerebbe a eseguire le istruzioni del programma corrente
2. Richiedere l'intervento del SO, cioè avviare l'esecuzione di codice appartenente al SO.

Un interrupt avviene quando la CPU riceve un segnale hardware (interrupt request) su un'apposita linea del bus di sistema. A seconda dell'origine di questo segnale, gli interrupt si suddividono in:

hardware interrupt □ il segnale è inviato dal clock o dal controller

program interrupt □ il segnale è provocato dal programma in esecuzione

8) cosa sono e come si suddividono gli hardware interrupt

Gli hardware interrupt si suddividono in:

timer interrupt□ usati dal clock per notificare il tick

I/O interrupt□ usati dai dispositivi di I/O per notificare eventi trattati dal SO

Questi sono eventi asincroni rispetto al programma in esecuzione, che non ha modo di prevedere se e quando si verificano. Le operazioni che il SO deve eseguire per gestirli varia in base al tipo di evento. Ad esempio, deve controllare se il tick ha causato l'esaurimento del time slice del programma in esecuzione e nel caso eseguire il context switch. Oppure deve rendere nuovamente schedulabile un programma quando termina l'operazione di I/O che esso stava aspettando.

Le fasi sono:

1. La CPU, mentre sta eseguendo l'istruzione i di un programma P, riceve un'interrupt request.
2. Una volta termina l'esecuzione dell'istruzione i, la CPU sospende l'esecuzione di P e salta invece alla procedura di gestione dell'interrupt, detta interrupt handler, che è parte del codice del SO.
3. L'interrupt handler gestisce l'interrupt
4. L'interrupt handler restituisce il controllo a P, oppure a un altro programma P' interrotto in precedenza
5. Il programma schedulato al punto 4 riprende la propria esecuzione dal punto in cui era stato interrotto.

8bis) **Dare la definizione di interrupt e quali tipi di interrupt possono verificarsi**

Gli interrupt sono un meccanismo per **notificare** alla **CPU** un **evento** o una condizione avvenuti nel sistema che devono essere **trattati** dalla **CPU**; quindi, se abbiamo un programma in esecuzione, l'interrupt deve **interrompere** il normale ciclo di esecuzione ed **eseguire codice** del s.o. Esistono principalmente due tipi di interrupt: interrupt program e interrupt hardware.

- **Interrupt hardware**: sono eventi asincroni rispetto al sistema (non posso prevedere quando accadono) e li distinguiamo tra:

  → **i/o interrupt**: usati dai dispositivi per notificare eventi alla CPU che devono essere trattati dal S.o.

  → **timer interrupt**: usato dal clock per notificare il tick (tempo per monopolizzare la CPU scaduto).

- **Program interrupt**: sono eventi sincroni rispetto al sistema (possiamo sapere quando avvengono) e li distinguiamo tra:

  → **Eccezioni:** situazioni anomale in seguito a operazioni aritmetiche o operazioni di reindirizzamento

  → **Software interrupt o trap**: vengono fatte tramite system call offerte al programmatore per richiedere l'intervento del s.o.

Entrambe possiedono una **gerarchia** e un ordine di **priorità**, e durante l'esecuzione di interrupt di una data priorità, gli interrupt di **priorità più bassa** anche se manifestati in seguito, **NON** comportano alcuna modifica fino a che la priorità di esecuzione **non diminuirà fino al loro livello**. (interrupt hardware di priorità notevolmente più ALTA rispetto a quelle program).

9) cosa sono i program interrupt ?

I **Program Interrupt** sono interruzioni causate dall'esecuzione di un programma e si dividono in:
Eccezioni (Interrupt Interni) Origine: Errori o situazioni particolari del programma. **Esempi**: Divisione per zero, page fault, violazione della memoria. **Caratteristica**: Sincrone e non intenzionali.
Software Interrupt (Trap) Origine: Generati intenzionalmente da un'istruzione TRAP per richiedere servizi al sistema operativo. **Caratteristica**: Usati per chiamate di sistema come I/O o gestione risorse. **Parametro**: Passato come operando della TRAP o tramite stack. **Differenza**: Le eccezioni segnalano errori, mentre i trap richiedono servizi al sistema operativo.

10) interrupt request e come vengono gestite ?

Quando si gestisce un interrupt request si ignorano temporaneamente le richieste di interrupt della medesima classe e edelle classi con minor priorità, la cui gestione viene rimandata. Tale richieste rimangono pendenti. Il registro PSW contiene alcuni bit che svolgono il ruolo di interrupt mask (IM). L'interrupt mask `e una sequenza di bit che mi dicono quali interruzioni sono abilitate e quali mascherate. Esistono due implementazioni possibili per l'interrupt mask: l'IM contiene un bit (enabled/masked off) per ogni classe di interrupt. l'IM contiene un valore m per abilitare tutti e soli gli interrupt di priorità

11) cosa è un dispostivo I/O(controller) e DMA e come comunicano le cpu con le porte i/o

Si distingue tra controller (o adapter che comunica con la CPU tramite il bus di sistema) e dispositivo vero e proprio (gestito dal controller). L'interfaccia tra CPU e controller è composta da registri di controllo (porte di I/O) e un buffer (per memorizzare i dati durante operazioni I/O). I registri di controllo vengono usati dalla CPU per inviare comandi al controller e dal controller per comunicare alla CPU i risultati dei comandi e lo stato del dispositivo.La comunicazione tra CPU e dispositivi avviene tramite un'interfaccia costituita da registri di controllo e un buffer, gestita dai driver del sistema operativo. Esistono due modalità di comunicazione: le **istruzioni macchina ad hoc**, che richiedono codice in assembly, e il **memory mapped I/O**, in cui le porte di I/O sono mappate su indirizzi di memoria, semplificando l'uso di linguaggi come C ma complicando la gestione della cache.

Per eseguire operazioni I/O si utilizzano tre approcci principali:

- **Programmed I/O**, dove il trasferimento avviene un byte alla volta e il sistema operativo utilizza la CPU per attendere lo stato pronto del dispositivo (busy waiting), risultando poco efficiente.

- **Interrupt Driven I/O**, che riduce il busy waiting: il primo byte viene trasferito normalmente, ma i successivi vengono gestiti tramite interrupt, riducendo l'attesa attiva ma introducendo un overhead dovuto alla gestione degli interrupt.

- **Direct Memory Access (DMA)**, dove il trasferimento è effettuato direttamente dal DMA, senza coinvolgere la CPU. La CPU viene liberata per altre operazioni e il DMA gestisce i dati, notificando la CPU solo alla fine del trasferimento. Questo approccio è il più efficiente per grandi quantità di dati.

12) cosa è una system call ?

È una richiesta al SO effettuata da un programma. Sono quindi l'interfaccia tra i programmi utente e il SO. Sono realizzate con la TRAP e ne esiste una per ogni valore consentito del parametro della TRAP, quanti e quali sono esattamente le system call dipende dal SO. Il programma che invoca la system call ottiene un risultato, memorizzato in un registro.

Per invocare le system call si usa l'istruzione TRAP. Oltre al suo parametro, ciascuna system call richiede dei propri parametri. In C è disponibile una libreria di funzioni associate alle system call, sono scritte in assembly.

In UNIX/Linux si hanno circa 100 system call, alcune servono per interagire con il file subsystem e altre riguardano il process control subsystem. Alcuni esempi sono *create* (creare un file)*, open* (creare un accesso a un file)*, read* (leggere da un file)*, write* (scrivere in un file)*, close* (invalidare un accesso a un file)*.*

13) cosa si intende con MEMORY LAYOUT DEI PROGRAMMI

L'esecuzione di un programma necessita di 3 aree di memoria:

1. Area di testo contiene il testo del programma, cioè le istruzioni in linguaggio macchina e non è modificabile dal programma
2. Area dati contiene le variabili globali, cioè condivise da tutte le procedure e le strutture dati dinamiche che si trovano nell'area heap. L'area dati ha contenuto variabile e dimensione variabile.

3. Area di stack contiene i record di attivazione(frame) delle procedure già chiamate ma non ancora terminate. Ciascun record di attivazione contiene i parametri attuali, le variabili locali, il return address(RA, indirizzo della prossima istruzione da eseguire al termine della procedura) e un puntatore alla cima del record. Quest'area ha contenuto variabile e dimensione variabile.

14) cosa si indica con INDIRIZZI DI MEMORIA ?

Il codice di memoria contiene indirizzi di memoria che si riferiscono alle tre aree. Essi però non sono indirizzi reali ma virtuali. Infatti, quando un programma viene scritto/tradotto in linguaggio macchina non si può sapere quali parti della RAM verranno assegnate al programma per l'esecuzione. Essi partono da 0.

Per tradurre gli indirizzi da virtuali a reali si usa il **Relocation Register (RR)**.

15) spiega il concetto di processo?

Un processo `e un'esecuzione di un programma (M.J. Bach).Questa definizione prevede che possa lanciare pi`u volte un programma non necessariamente attendendo la terminazione del programma stesso. Un processo `e un'istanza di un programma in esecuzione (D.M. Dhamd-here, A.S Tanebaum). Possiamo avere pi`u processi relativi al medesimo programma, in quanto possiamo avere pi`u esecuzioni, anche in contemporanea, del programma.

• Il programma `e un'entit`a passiva che non esegue nessuna azione di per s`e

• L'esecuzione del programma, chiamata processo, concretizza le azioni specificate nel programma

• Il S.O. schedula processi, non programmi

Diciamo inoltre che un processo consiste delle seguenti componenti:

• Stato della CPU: caratterizza lo stato di avanzamento del nostro processo

(PC)

• Area di testo

• Area di dati

• Area di stack

• Risorse logiche e fisiche assegnate al processo

15bis) **quali sono i tipi di interazioni tra i processi**

Inanzitutto, definiamo quando due processi sono interagenti o indipendenti.

Dati due processi P1 e P2, se l'intersezione delle operazioni di lettura/scrittura svolta da tali processi NON è nulla, si dicono interagenti, se invece tale intersezione e NULLA, si dicono indipendenti.

Esistono principalmente 4 tipi di interazioni tra processi:

Data sharing: condivisione di dati tra processi

Control Synchronization: azioni di verifica per la sincronizzazione.

Message passing: scambio di messaggi tra processi.

Signals: invio di un segnale da parte di un processo ad un altro per notificare una situazione particolare.

16) COSA SI INTENDE PER Parallelismo e concorrenza

Due eventi sono paralleli se occorrono nello stesso momento. La concorrenza è l'illusione del parallelismo, cioè se si ha l'illusione che vengano eseguite in parallelo. Su un sistema con una singola CPU, il SO può assegnarla a turno ai vari processi (interleaving), realizzando l'esecuzione concorrente di tali processi: il parallelismo è solo simulato, perché in ogni istante la CPU può essere usata al max da un solo processo. Se, invece, è presente un DMA controller, possono avvenire in parallelo l'esecuzione di un processo e un'operazione di I/O richiesta da un altro processo.

La velocità di avanzamento di un processo non è uniforme nel tempo, perché in alcuni momenti esso dispone della CPU, mentre altri no. La sua velocità dipende dal numero e dal comportamento di tutti i processi presenti, quindi non è riproducibile.

Se sono sistemi con più CPU, si distingue in **multiprocessing**(più processi possono eseguire su una macchina dotata di più CPU) e **distributive processing**(più processi possono eseguire su più macchine distribuite e indipendenti). Questo parallelismo però è limitato dal numero di CPU.

17) STATI DI UN PROCESSO

In ogni istante un processo si trova in un certo stato:

Running : Il processo è in esecuzione sulla CPU. Ci può essere un processo running alla volta, oppure nessuno. Quando un processo viene interrotto da un interrupt, esso si considera running anche durante l'esecuzione dell'handler e dello scheduler, finchè quest'ultimo non seleziona un processo diverso da eseguire.

Ready : il processo non è in esecuzione, ma sarebbe in grado di eseguire se ottenesse la CPU. Ciò avviene quando il numero di processi che potrebbero eseguire è maggiore del numero di CPU disponibili.

Waiting : il processo non è in esecuzione e non sarebbe in grado di eseguire se ottenesse la CPU, ma acquisirà la capacità di eseguire se e quando si verificherà un determinato evento, che il processo sta aspettando.

Ci sono 4 possibili transizioni da stato a stato:

1. **Ready - running**: lo scheduler seleziona il processo per l'esecuzione, e assegna a esso la CPU. Un processo non può richiedere di essere schedulato in un certo momento, infatti se ci sono più processi ready, la scelta viene effettuata in base allo scheduling.
2. **Running - ready**: viene sottratta la CPU al processo per assegnarla a un altro. Ciò avviene quando il processo subisce una preemption e in un sistema con timesharing, quando scade il time slice del processo. È un evento passivo.
3. **Running - waiting**: il processo si trova impossibilitato a eseguire e si blocca in attesa di un evento sbloccante. Ad esempio, quando il processo richiede un'operazione di I/O, esso non può sfruttare la CPU finché l'operazione non è completata.
4. **Waiting- ready**: si verifica l'evento sbloccante. In base allo scheduling e agli altri processi, potrebbe che il processo venga immediatamente schedulato. Ad esempio, se lo scheduler è basato sulle priorità e il processo diventato ready ha priorità maggiore di quello running, che subisce una preemption, perdendo la CPU.

18) Descrivi gli Stati dei processi in UNIX System V

READY e WAITING sono divisi in due sottostati: in memory e swapped per la gestione della memoria virtuale. Il SO può eseguire più processi di quanti ce ne stiano in RAM: alcuni hanno aree di testo, dati e stack caricate interamente in memoria, altri le hanno interamente su uno swap device che è un dispositivo logico che corrisponde a una porzione del disco riservata. Quando serve memoria, il SO esegue lo swapping out, spostando un certo numero di processi dalla RAM allo swap device; lo swapping in è invece lo spostamento dallo swap device alla RAM, in base alla quantità di memoria libera e solo sui processi ready.

RUNNING è diviso in user running e kernel running. Un processo è user running quando la CPU sta eseguendo il suo codice. Se si ha un interrupt si passa al kernel running e viene eseguito il codice dell'interrupt handler, il quale chiama lo scheduler. Se quest'ultimo seleziona lo stesso processo, si ritorna allo user running, altrimenti diventa ready o waiting. L'esecuzione di un interrupt handler può essere interrotta da un interrupt di priorità maggiore.

CREATED e ZOMBIE quando viene chiesto di eseguire un programma, il SO deve costruire un processo e assegnarlo a tale programma. Durante questa fase, il processo è in stato created e poi diventa ready in memory se c'è spazio in RAM, se no swapped out. Un processo appena creato non può andare in stato di waiting perché è sicuramente pronto a eseguire la sua prima istruzione. Quando un processo termina, entra in stato di zombie, che serve per raccogliere le statistiche sulla sua esecuzione prima che esso venga cancellato completamente. Quando il processo ha finito, l'ultima istruzione che esegue è la system call exit; se il processo si arresta a causa di un'anomalia è comunque il SO a terminarlo.

19) COSA SI INTENDE CON PCB (Process control block)

Contiene l'identificativo del processo (PID), lo stato del processo, lo stato della CPU (in particolare i valori dei registri), la priorità, dei puntatori a varie zone di memoria che contengono info per acc edere all'area di testo, all'area dati, all'area di stack, ai file aperti e ai dispositivi aperti. E contiene anche uno o più **PCB pointer**, che permettono la realizzazione delle strutture dati usate per gestire lo scheduling.

La memoria fisica è paginata, cioè divisa in più page frame, tutti della stessa dimensione.

21) cosa è il contesto di un processo e il CONTEXT SWITCH?

Il contesto di un processo, detto anche ambiente è costituito da register context (contenuto dei registri della CPU), user-level context (aree di testo, dati e stack a cui il processo può accedere), system-level context (info che riguardano il processo, gestite dal SO).

Il SO effettua 3 operazioni sui processi per gestirne l'esecuzione:

1. Context save quando il processo running perde la CPU, il SO salva nel PCB tutti i dati che serviranno a far ripartire il processo
2. Scheduling il SO sceglie un processo da eseguire tra quelli ready
3. Dispatching il SO ripristina il contesto del processo schedulato, sfruttando i dati salvati nel PCB al momento del context save.

Si ha un context switch quando il SO effettua il context save per un processo P, schedula un processo P' diverso da P ed effettua il dispatching per P'. il processo è sempre in running durante il context switch. Può causare un overhead significativo. Esso causa un rallentamento del sistema.

22) cosa è la pcb e quale è il suo ruolo?

La **PCB (Process Control Block)** è una struttura dati essenziale nei sistemi operativi, utilizzata per gestire i processi in esecuzione o in attesa. Ogni processo ha la sua PCB, che contiene tutte le informazioni necessarie per il suo controllo e gestione da parte del sistema operativo.

La PCB identifica un processo in modo univoco grazie a un identificatore (PID) e memorizza il suo stato attuale, come "pronto", "in esecuzione" o "in attesa". Contiene inoltre il contesto del processo, cioè i valori dei registri della CPU e il program counter, in modo che il sistema operativo possa sospendere un processo e riprenderlo successivamente esattamente dal punto in cui era stato interrotto.

Oltre al contesto, la PCB gestisce tutte le risorse allocate al processo, come la memoria utilizzata, i file aperti e i dispositivi I/O associati. In più, memorizza informazioni sulla priorità del processo, utili per decidere l'ordine con cui i processi devono essere eseguiti, e sul tempo di esecuzione, per monitorarne le prestazioni.

La PCB è cruciale soprattutto durante il **context switch**, il meccanismo che consente al sistema operativo di passare l'esecuzione da un processo all'altro. Grazie alla PCB, ogni processo può essere ripreso esattamente dallo stato in cui era stato interrotto, garantendo un funzionamento corretto ed efficiente del sistema operativo.

22bis) Definire come la memoria viene "riusata"

La memoria libera, o liberata dai dati dei processi, deve poter essere utilizzata. Abbiamo principalmente tre tecniche per allocare dati:

First fit: si cerca la prima area di dimensione "sufficiente" e l'area rimanente rimane "free list" (libera). Si creerà però un problema di "frammentazione", il quale è un problema in cui parti di memoria non possono essere utilizzati perché troppo "piccoli".

Best Fit: si cercano le aree di dimensione sufficientemente maggiore rispetto a quella da impiegare e si sceglie quella più "piccola". Questa ricerca e scelta finale comporta un notevole impiego di tempo e in più a lungo andare il problema di frammentazione presentato nella free list si ripresenterà.

Next Fit: si addotta la first fi partendo però, dal punto in cui era stata effettuata l'ultima locazione.

Come definito precedentemente, introduciamo il problema della "frammentazione" in cui esistono zone di memoria in un sistema di computazione inutilizzabili. Per contrastare tale problema abbiamo: boundary tags, le aree libere confinanti vengono collegate, e le aree libere sono collegate da una catena di pointer, memory compaction, a determinati intervalli di tempo tutte le aree libere vengono unite in un'unica area libera.

23) cosa si intende con creazione di processi ?

In generale, un processo ne può creare un altro, mediante un'apposita system call. Ad esempio, su UNIX/Linux, si usa la system call **fork**. In alcuni SO, si crea una relazione gerarchica tra il processo che esegue la system call (processo padre) e il processo creato (processo figlio). I processi figli sono quasi uguali ai processi padre. I due processi hanno PCB distinti con PID diversi, valori uguali nei registri e aree di testo, dati e stack distinte, ma con gli stessi contenuti. Al termine della system call fork, padre e figlio sono entrambi ready, ma non si sa quale verrà schedulato per primo.

In UNIX/Linux, la creazione di un nuovo processo che esegue un altro programma si effettua in due fasi:

1. Il processo corrente si clona, usando la fork per creare un processo figlio
2. Il processo figlio si cambia il programma, mediante la system call **exec**.

Gli effetti principali della exec sul processo chiamante sono:

- Le aree di testo, dati e stack vengono sostituite con quelle del nuovo programma
- I registri di controllo assumono i valori necessari per l'esecuzione del nuovo programma
- I registri generali vengono azzerati
- Vengono chiusi eventuali file aperti e rilasciati eventuali dispositivi in uso.

24) cosa si intende con terminazione di un processo ?

La terminazione di un processo è effettuata sempre dal SO, ma avviene per vari motivi:

- Il processo può richiedere di terminare tramite un'apposita system call
- Può essere forzata dall'interrupt handler delle eccezioni
- Un processo può richiedere di farne terminare altri

Quando un processo termina, rilascia tutte le risorse (CPU, memoria, file…), i suoi eventuali figli possono essere o meno terminati, a seconda del SO, la distruzione del PCB non è sempre immediata.

25) cosa si intende per fork , exec, wait ?

Nei sistemi operativi, **fork**, **exec** e **wait** sono chiamate di sistema utilizzate per creare e gestire i processi.

Con **fork**, un processo crea una copia di sé stesso, dando origine a un processo figlio. Dopo la chiamata, sia il processo padre che il figlio continuano l'esecuzione dal punto successivo, ma con un identificatore (PID) differente. Il padre riceve il PID del figlio, mentre il figlio riceve il valore 0. È spesso usato per avviare nuovi processi paralleli.

La chiamata **exec** consente a un processo di sostituire il proprio programma in esecuzione con uno nuovo. Il processo rimane lo stesso (stesso PID), ma il contenuto della memoria viene completamente sovrascritto. Solitamente, exec viene usato dopo un fork per far sì che il figlio esegua un programma diverso da quello del padre.

Infine, con **wait**, il processo padre può sospendersi in attesa che il processo figlio termini. Questo consente al padre di sincronizzarsi con il figlio e raccogliere il suo stato di uscita.

Queste tre chiamate lavorano insieme per creare e coordinare processi, rendendo possibile la gestione del multitasking.

26) quali sono i diversi tipi di processi ?

In UNIX, esistono tre tipi di processi:

- **User process**
- **Daemon process**
- **Kernel process**

Uno user process è associato a un utente che lavora a un terminale. Esegue la CPU in modalità user, ha bisogno di usare la system call per richiedere servizi del SO. Es. shell

La shell è uno user process che è normalmente in waiting, in attesa dell'input dell'utente. Quando viene inserito il comando, la shell avvia il programma. Può eseguirlo in due modi diversi:

1. Esecuzione classica: la shell fa una fork; il figlio fa la exec, mentre il padre fa la wait per attendere la terminazione del figlio; quando il figlio termina, il padre si mette in attesa di un altro comando da parte dell'utente.
2. Esecuzione in background: fork ed exec vengono fatte come per l'altra esecuzione, ma poi la shell non aspetta la terminazione del figlio e si mette subito in attesa di un altro comando.

I daemon process non sono associati a nessun utente. Essi tipicamente non terminano, svolgono operazioni di routine vitali per il sistema, passano quasi sempre la loro vita in stato di waiting, eseguono in modalità user. Un esempio è il processo *getty* che gestisce il processo di login di un utente e se va a buon fine avvia la shell e si mette in attesa che essa termini.

I kernel process sono daemon che eseguono in modalità kernel e quindi possono accedere alle procedure e alle strutture dati del kernel senza bisogno di invocare le system call e sono potentissimi. Ad esempio, lo **swapper process**, che ad intervalli regolari esegue lo swap out dei processi con maggiore anzianità in memoria e lo swap in dei processi che sono da più tempo in stato ready swapped. Un altro esempio è lo **stealer process**, che ad intervalli regolari esegue lo swap out dei processi che non sono stati utilizzati di recente.

27) cosa si intende per task, thread e thread switching ?

Un'applicazione può avere più compiti, chiamati **task** da portare avanti contemporaneamente. Implementare varie attività in parallelo risulta molto difficile.

Invece se il linguaggio utilizzato offre i costrutti per la programmazione concorrente, cioè i costrutti per definire flussi di esecuzione sequenziale indipendenti per i vari task ed eseguire flussi concorrentemente, si può guadagnare in termini di semplicità ed efficienza di esecuzione se si ha parallelismo:

- Tra il lavoro della CPU di un task e il lavoro di I/O di un altro task
- Tra il lavoro di CPU di due o più task.

Una soluzione adatta alla programmazione concorrente è avere gruppi di processi light che condividano memoria, file, dispositivi e altre risorse e che si differiscano solo per lo stato della CPU.

Un thread è un'esecuzione di un programma che usa le risorse di un processo. Ci possono essere più thread associati a uno stesso processo. Le aree di test, dati e le risorse logiche e fisiche sono condivise tra tutti i thread di uno stesso processo. Ogni thread ha i propri identificatori (TID) e stato della CPU, stack e stato (running, waiting, ready…). Queste info sono contenute nel **Thread control Block (TCB)**, che è situato in una thread table.

Il thread switching si basa sullo stesso principio del context switching, ma ha effetti diversi. Introduce meno overhead, a vantaggio dell'efficienza, perché: non vanno aggiornati i dati della MMU relative alle aree di testo e dati; è possibile che la cache della memoria richieda meno aggiornamenti.

28) cosa si intende per processi iterangenti e indipendenti ?

Due processi concorrenti Pi e Pj processi interagenti se vale almeno una delle seguenti due proprietà

• Ri e Wj hanno intersezione non vuota,

• Rj e Wi hanno intersezione non vuota

Quindi se il processo i invia informazioni al processo j, oppure il processoj invia informazioni al processo i, oppure entrambe le cose.Nell'esempio delle prenotazioni aeree, gli n processi sono due a due interagenti, perché leggono e modificano la variabile next seat. (next seat è nel read set e nel write set di ogni processo.). un processo di dice ibndipendente se non è iterangente

29) Definizione formale di race condition?

Siano *d* un dato condiviso dai processi/thread P e P', *o* e *o'* operazioni su *d* eseguite da P e P', *f* e *f'* funzioni: le operazioni *o* e *o'* danno luogo a una **race condition (corsa critica)** sul dato condiviso *d* se, eseguendo *o* e *o'* con *d* avente un valore iniziale *v*, accade che *d* assume un valore *v'* diverso da *f(f'(v))* e da *f'(f(v))*.

Le race condition si possono verificare anche in una gestione di una pila condivisa, implementata mediante un array stack e un indice top che punta all'elemento in cima.

Le race condition hanno 2 conseguenze:

- Il comportamento dei processi/thread coinvolti non è corretto

- I dati su cui ha luogo la race condition diventano inconsistenti, cioè assumono uno stato non previsto.

Quando si eseguono più volte questi programmi, le race condition si verificano solo alcune delle volte, quindi non è facile verificarne la correttezza.

30) cosa significa sezioni critica ?

Una sezione critica CS per un dato condiviso d è una porzione di codice che viene eseguita non concorrentemente con se stessa o con altre sezioni critiche per d. Si parla anche di mutua esclusione, perché le CS su un dato d sono mutualmente esclusive: quando un processo accede a una di esse, impedisce a qualunque altro processo di accedere a tutte le CS sullo stesso dato d.

Le implementazioni delle CS devono soddisfare alcune proprietà:

- Correttezza le CS non possono essere eseguite concorrentemente cioè deve essere garantita la mutua esclusione
- Progresso se nessun processo sta eseguendo una CS e altri processi vorrebbero farlo, allora uno di essi deve poter eseguire la propria CS.
- Attesa limitata dopo che un altro processo manifesta la volontà di accedere a una CS, il numero di accesso alle CS da parte di un qualsiasi processo P' che precedono l'accesso di P deve essere minore o uguale a un dato intero k.

Le ultime due proprietà prevengono la starvation, cioè l'attesa infinita da parte dei processi.

30bis) Definizione di operazione indivisibile e i vari tipi di approcci usati per le implementazioni delle sezioni critiche

Intendiamo con mutua **esclusione**, che le CS su un dato d sono mutualmente esclusive (si **escludono a vicenda**).

Le sezioni critiche possono essere implementate con: costrutti ad hoc per programmazione concorrente, approccio algoritmico o uso di primitive software (system call).

Esistono diverse proprietà richieste per l'implementazione delle CS:

Correttezza: deve essere garantita la mutua esclusione.

Progress: se qualcuno vuole operare sul dato condiviso e nessun'altro sta lavorando, deve poterlo fare

Bounded wait: il numero di processi di cui devo attendere l'esecuzione deve essere controllato e non infinito.

Un'operazione indivisibile su un dato condiviso d è un'operazione che è con certezza eseguita in modo **NON concorrente** rispetto ad altre operazioni su d. (es**. test and set**, test valore registro 0 e lo setta con sequenza di 1, oppure istruzione **swap** che scambia il contenuto di due locazioni di memoria).

31) OPERAZIONE INDIVIIBILE

Un'operazione indivisibile su un dato condiviso d`e un'operazione
che `e con certezza eseguita in modo non concorrente rispetto ad altre operazioni
su d. Un'operazione indivisibile viene talvolta detta atomica. Per definizione, le operazioni indivisibili su d non possono dar luogo a race condition su d.

32) COSA È L ISTRUZIONE TS

L'istruzione **test and set (TS)** ha come operando una locazione di memoria ed esegue due azioni:

1. Controlla se tale locazione di memoria ha valore 0, ponendo il risultato nel bit CC della PSW
2. Imposta la locazione di memoria a una sequenza di 1, indipendentemente dall'esito del controllo.

TS è un'istruzione indivisibile sulla locazione di memoria:

- Su un'architettura uniprocessore, gli interrupt non possono interrompere una singola istruzione
- Su un'architettura multiprocessore, bisogna evitare che vengano eseguite contemporaneamente più TS.

Questa istruzione permette di implementare le sezioni critiche, mediante variabili di lock condivise, poiché permette di eseguire in modo indivisibile il controllo dello stato di una variabile lock e l'assegnamento a tale variabile. Questa soluzione però è disponibile solo se il programma è in linguaggio macchina.

In architetture diverse, la TS si può chiamare TSL (test and set lock) e può avere un secondo parametro, che specifica il registro in cui memorizzare il valore del test. Oppure, invece della TS, può essere presente un'istruzione indivisibile swap, che scambia il contenuto di due locazioni di memoria.

33) COME VENGONO IMPLEMENTATE LE CS ?

Nei linguaggi ad alto livello, esistono tre strategie per l'implementazione delle CS:

1. Approccio algoritmico i processi effettuano una serie di controlli per determinare se una CS è libera prima di entrarvi
2. Uso di primitive software (system call o chiamate di libreria) per garantire la mutua esclusione
3. Uso di costrutti ad hoc per la programmazione concorrente

L'**approccio algoritmico** prevede di implementare le CS senza usare istruzioni hardware ad hoc e system call e costrutti specializzati forniti dal linguaggio di programmazione. Prima di entrare in una CS, un processo effettua un controllo: se esso è superato, il processo entra nella CS, se no il processo ripete il controllo eseguendo busy wait.

L'**algoritmo di Dekker** usa una variabile di turno (*turn*) per decidere quale processo accede alla CS quando entrambi vogliono farlo. Questa soluzione rispetta la proprietà del progresso, ma è valida solo per 2 processi.

L'**algoritmo di Peterson** è valido solo per 2 processi: ogni processo ha una flag booleana che usa per dichiarare di voler accedere alla CS; per gestire il caso in cui entrambi i processi vogliono entrare, viene usata la variabile di turno, ma ciascun processo assegna il turno al "rivale" immediatamente prima di controllare se può entrare e aspetta che il rivale finisca di eseguire la CS.

Due algoritmi per implementare delle CS con più di 2 processi sono l'algoritmo di Eisenberg e McGuire e l'algoritmo di Lamport.

34bis) Dare le definizioni di deadlock e livelock

DeadLock: situazione in cui i processi sono bloccati e si bloccano uno in attesa dell'altro.

Livelock: situazione in cui i processi, pur non essendo bloccati, impediscono l'un l'altro all'infinito di eseguire.

35) COSA SI INTENDE PER SEMAFORO ? Definire le operazioni possibili al suo interno

Un semaforo è una variabile interna condivisa che può assumere solo valori non negativi e su cui sono possibili tre operazioni:

1. Inizializzazione, con un valore>=0
2. Operazioni indivisibile wait:
    - Se il semaforo ha valore > 0, viene decrementato
    - Se il semaforo ha valore = 0, il processo va in waiting
1. Operazione indivisibile signal:

    Se ci sono processi bloccati sul semaforo, uno di essi va in ready

    Se nessun processo è bloccato sul semaforo, viene incrementato il valore del semaforo

Quando ci sono processi bloccati su un semaforo, il valore di quest'ultimo è sempre 0.

Un semaforo può essere usato per garantire la mutua esclusione su un altro dato condiviso:

- Il semaforo viene inizializzato a 1; ***semaphore sem = 1***
- Prima di accedere alla CS, si effettua una wait sul semaforo
- Dopo aver eseguito la CS, si effettua una signal sul semaforo

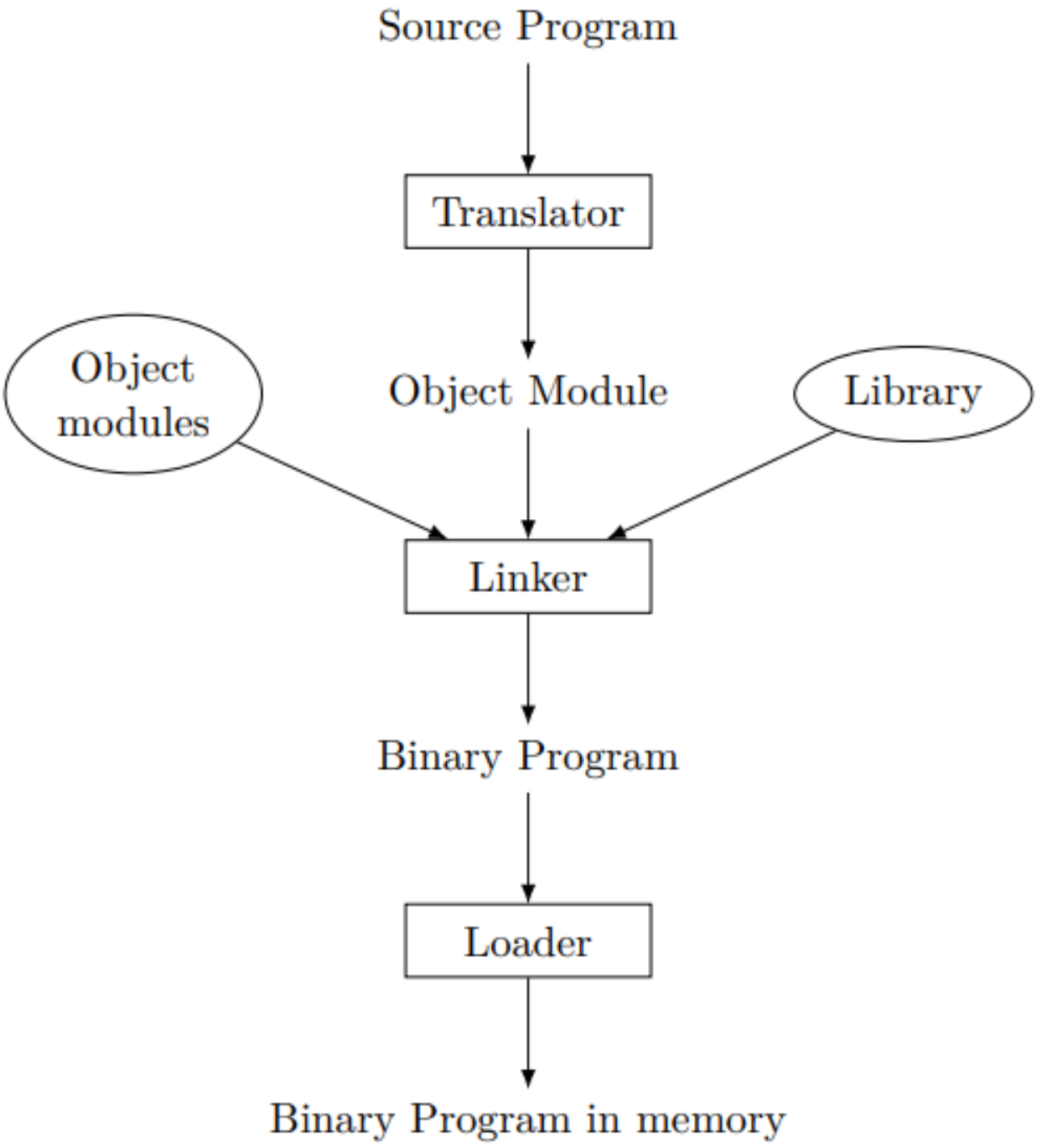
Se il semaforo fosse inizializzato a un valore n > 1, non si rispetterebbe la proprietà di correttezza, perché nelle CS entrerebbero in concorrenza n processi. Se il semaforo fosse inizializzato a 0, nessun processo accederebbe alle CS. È compito del programmatore rispettare la sequenza *wait-{CS}-signal*. A una variabile semaforo usata per implementare le sezioni critiche, si dà il nome *mutex*.

39) COME VIENE GESTITA LA MEMORIA?

Gli obbiettivi del SO sono: allocare più processi in memoria, per avere parallelismo e migliorare la performance del sistema; proteggere la memoria dei processi, impedendo che ciascuno di essi acceda alle aree degli altri processi. Il SO offre la **memoria virtuale**.

Il passaggio da programma sorgente a programma binario in memoria è composto da 3 fasi:

1. **Traduzione**
2. **Linking**
3. **Loading**



Source program è il programma sorgente, scritto in un linguaggio come assembly o C.

Il **translator** riceve in input il programma sorgente e produce un object module: un programma in linguaggio macchina non ancora eseguibile perché può contenere riferimenti a funzioni/dati definiti in altri moduli e librerie con indirizzi non ancora noti.

Il **linker** collega vari moduli oggetto e le librerie, producendo il programma eseguibile; in questa fase gli indirizzi di memoria assegnati possono essere rilocati/cambiati per evitare overlapping con indirizzi di altri moduli o librerie. Quando si vuole eseguire il programma, il **loader** lo carica in memoria, associandolo a un processo.

41) COSA È COME SI SUDDIVIDE L ALLOCAZIONE/ RILOCAZIONE ?

L'allocazione in un sistema operativo riguarda il modo in cui viene gestita la memoria per i processi in esecuzione. Si occupa di assegnare lo spazio necessario ai programmi e ai dati affinché possano funzionare correttamente. Questa gestione può essere suddivisa principalmente in due categorie: **non contigua** e **contigua** .

L'**allocazione contigua** della memoria avviene quando ogni processo è allocato in una singola area di memoria contigua. La protezione della memoria è semplice, attraverso registri LBR/UBR. La rilocazione dei processi è semplice, mediante il registro RR. Si ha però il problema della frammentazione, perché nessuna singola area libera è abbastanza grande per il processo da allocare. Questa allocazione può essere integrata con lo swapping, cioè i processi che non ci stanno in RAM vengono spostati sullo swap device. Lo *swap out* (spostamento dalla RAM allo swap device) e lo *swap in* (dallo swap device alla RAM) sono eseguiti da un processo di sistema, dove la macchina è inutilizzabile.

L'**allocazione non contigua** della memoria avviene se ogni processo può essere allocato in più aree di memoria non adiacenti. Ogni porzione del processo si chiama componente. non si ha il problema della frammentazione. La protezione e traduzione da indirizzi logici a fisici sono più complicate dell'allocazione contigua.

42) cosa si intende per Gerarchia di memoria

Le unità di memoria più veloci sono anche più costose, con capacità minori. Si utilizza quindi una **gerarchia di memoria**, costituita da unità piccole e veloci e da unità grandi e lente. Nei sistemi con memoria virtuale un'area di memoria può essere parzialmente in RAM e parzialmente sul disco. Questa memoria è trasparente al programmatore poiché gestita dal SO. Per velocizzare gli accessi, una parte della RAM viene ricopiata nella cache cosicché l'indirizzo venga cercato prima in cache e solo se non viene trovato si accede alla RAM. La cache viene gestita direttamente a livello hardware.

Il problema dell'allocazione della memoria può essere visto a due livelli:

1. Il SO deve allocare memoria per i processi
2. All'interno della memoria di un processo, il RTS del linguaggio deve allocare memoria per i dati PCD.

Bisogna comunque riutilizzare la memoria liberata. Per poterlo fare, bisogna tener traccia di quali blocchi di memoria sono occupati e quali liberi. Una soluzione è l'uso di una **free list**: si ha un puntatore al primo blocco libero; ogni blocco contiene un campo che ne indica la dimensione, un puntatore al blocco libero successivo e opzionalmente un puntatore al blocco libero precedente.

42bis) Definire i vari tipi di dati che si allocano durante l'esecuzione di un programma e dare la definizione di RTS

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

Le variabili inizializzate dai programmi (String a, int b….)

I dati creati dinamicamente (es. Costruttori int [] a=new int […). Questi dati vengono allocati nell'area heap con PCD, Program Controlled Dynamic data.

Durante l'esecuzione di un programma, tutte le operazioni svolte per immagazzinare i valori delle variabili o dati, prendono il nome di Runtime Support (RTS), un esempio tipico è il Garbage Collector in Java.

43) si spieghino i concetti di linking statico e loading statico

Il **linking statico** è il processo in cui, durante la fase di compilazione, il codice oggetto del programma viene combinato con le librerie statiche per creare un eseguibile completo. In altre parole, tutte le funzioni e variabili necessarie per l'esecuzione del programma vengono incorporate direttamente nell'eseguibile. Una volta creato, l'eseguibile non dipende da librerie esterne al momento dell'esecuzione, poiché contiene già tutto il codice necessario. Questo rende il programma indipendente dalle librerie, ma la dimensione del file eseguibile risulta maggiore. Inoltre, se una libreria viene aggiornata, l'eseguibile deve essere ricompilato.

Il **loading statico**, invece, riguarda il processo di caricamento dell'intero programma, creato con il linking statico, in memoria. Durante il loading statico, il sistema operativo trasferisce il programma in RAM, dove sarà eseguito. Questo processo avviene in modo che l'intero eseguibile sia pronto per l'esecuzione, senza dover attendere caricamenti o risoluzioni di riferimenti esterni.

44) cosa si intende per frammentazione

La **frammentazione** è l'esistenza di aree di memoria non utilizzabili in un sistema di computazione. Per prevenirla si usano 4 tecniche:

- **Boundary tags**: le aree libere sono collegate da una catena di puntatori; all'inizio e fine di ogni area viene messo un tag che indica stato e dimensione; quando viene liberata un'area, la si unisce con aree libere adiacenti.
- **Memory compaction**: le aree libere sono collegate da una catena di puntatori; ogni tot tempo la memoria viene compattata spostando aree occupate e unendo quelle libere; questa compattazione implica un cambiamento degli indirizzi dei processi; quando il SO compatta la RAM la macchina è inutilizzabile.
- **Powers-of-two allocator**
- **Buddy systems**

45) quali sono le tecnice per effettuare una ricerca ?

Esistono 3 tecniche per effettuare la ricerca:

1. **First fit**: si cerca la prima area libera di dimensione sufficiente >= k; un blocco si può spezzare più volte di seguito, quindi si creano aree libere troppo piccole.
2. **Best fit**: si cerca il blocco libero più piccolo tra quelli di dimensione >= k; ricerca costosa, preserva aree libera grandi che a seguire possono diventare inutili.
3. **Next fit**: come il first fit, ma la scansione della free list parte dal blocco successivo a quello in cui è stata effettuata l'ultima allocazione; evita l'effetto della first fit senza aumentare il costo.

46) cosa è la paginazione + come la mmu gestisce i bit

L'idea della **paginazione** è dividere sia programmi sia la memoria in componenti di dimensioni non arbitrarie, ma fisse. La memoria è divisa in 2 page frame, ciascuno di capacità 2 byte. $2^{}$ corrisponde alla dimensione totale della memoria. Ognuno dei $2^{}$ byte della memoria è individuato da un indirizzo fisico di f+s bit, dove gli f bit più significativi individuano il page frame e gli s bit meno significativi individuano l'offset del byte nel page frame. Ogni processo è diviso in pagine della stessa dimensione dei page frame. Ognuno dei byte dello spazio del processo è individuato da un indirizzo logico di l+s bit (l bit più significativi, s bit meno significativi). Allocare un processo in memoria significa associare un page frame a ogni sua pagina. Ogni pagina può essere messa in un frame qualsiasi. Con essa si elimina quasi completamente la frammentazione. Per ogni processo, il SO mantiene una **page table** (array con indice il numero della pagina), che indica il page frame in cui è allocata ciascuna pagina di tale processo. I page frame liberi vengono organizzati in una lista chiamata **free frame list**: la dimensione della lista varia con l'allocazione e la terminazione dei processi; quando viene allocato un processo, si utilizzano i primi frame nella lista.

Per convertire un indirizzo logico di l+s bit in un indirizzo fisico di f+s bit, la MMU:

1. Estrae dall'indirizzo logico il numero di pagina, costituito da l bit più significativi
2. Ottiene il numero del frame corrispondente, usando il numero di pagina come indice per accedere alla page table
3. Sostituisce gli l bit del numero di pagina con gli f bit del numero di frame, lasciando invariati gli s meno significativi.

47) spiega la segmentazione

La **segmentazione** è un metodo di gestione della memoria che divide la memoria di un programma in blocchi di dimensione variabile, chiamati **segmenti**. Ogni segmento corrisponde a una parte logica del programma, come il **codice** (dove risiedono le istruzioni), i **dati** (dove sono memorizzate le variabili), lo **stack** (che gestisce le chiamate di funzione e le variabili locali) e l'**heap** (per l'allocazione dinamica della memoria).

A differenza della paginazione, che suddivide la memoria in blocchi di dimensione fissa, la segmentazione cerca di riflettere la struttura logica del programma. Per esempio, un programma potrebbe avere un segmento dedicato solo al codice eseguibile e un altro per i dati, il che rende la gestione della memoria più intuitiva.

In pratica, ogni segmento ha un **indirizzo base** (il punto di partenza in memoria) e una **lunghezza** (la sua dimensione). Quando il programma viene eseguito, il sistema operativo gestisce questi segmenti separatamente, allocando spazio in memoria per ciascuno di essi. Una caratteristica importante della segmentazione è che la dimensione dei segmenti non è fissa, il che consente ai segmenti di crescere o contrarsi in base alle esigenze del programma. Ad esempio, lo **stack** può crescere man mano che vengono fatte più chiamate di funzione, mentre l'**heap** può espandersi se il programma richiede più memoria dinamica.

La segmentazione ha alcuni vantaggi, come una gestione della memoria che segue la logica del programma, rendendo più semplice la comprensione e l'allocazione dei vari blocchi di memoria. Tuttavia, uno degli svantaggi principali è che, se non gestita correttamente, può portare a problemi di **frammentazione esterna**. In altre parole, se un segmento cresce o si riduce troppo, potrebbero esserci spazi di memoria inutilizzati che non possono essere riutilizzati da altri segmenti, riducendo l'efficienza della memoria.

48) traduzione con tlb

La procedura di traduzione è simile alla paginazione normale senza memoria virtuale, ma in più gestisce la possibilità che la pagina richiesta non sia in memoria e debba essere cercata dal paging device: gli l bit più significativi determinano la entry della page table da considerare; se validity bit=0 viene sollevato un page fault; il page fault handler usa il valore del campo disk address per individuare il blocco da leggere e accede alla free frame list per assegnare un frame libero alla pagina; dopo aver copiato la pagina nel page frame, il page fault handler imposta il validity bit a 1 e aggiorna il campo frame number; quindi se validity bit=1, l'indirizzo fisico viene determinato concatenando gli f bit del frame number con gli s bit meno significativi dell'indirizzo logico.

Se la MMU effettuasse la traduzione usando direttamente la page table, per ogni accesso alla memoria servirebbero due cicli di memoria: uno per leggere una entry della page table, due per l'accesso vero e proprio. Per risparmiare uno dei due cicli, la MMU può usare un dispositivo hardware chiamato **Translation Look-aside Buffer (TLB)**. Esso è una **memoria associativa** contenente coppie (page number, frame number): per accedervi si specifica un page number e se esso è presente in una delle coppie, l'hardware restituisce il frame number corrispondente. Queste memorie sono molto veloci, ma anche costose e piccole. Per la traduzione, la MMU accede al TLB: se nel TLB è presente la coppia per il page number considerato, la MMU utilizza il frame number ricavato per costruire l'indirizzo fisico; se invece la coppia non è presente nel TLB, si ha un **TLB miss** e la MMU deve accedere alla page table in memoria per effettuare la traduzione.

In generale la traduzione di un indirizzo prevede 3 casi:

1. Viene usata la entry del TLB dedicata alla pagina
2. Viene sfruttata la entry relativa alla pagina nella page table, se la entry ha validity bit=1 e il primo caso non è applicabile. Si aggiorna poi il TLB, inserendo una coppia corrispondente alla pagina.
3. Si genera un page fault. Dopo aver caricato in RAM la pagina, vengono aggiornati la page table e il TLB.

Esiste un registro chiamato **Page Table Address Register (PTAR)**: il valore da assegnare a esso è memorizzato nel PCB di ogni processo e l'assegnamento viene effettuato solo durante il context switch dal SO; quando la MMU deve accedere alla page table, il valore di questo registro viene sommato al numero di pagina per ottenere l'indirizzo della entry richiesta.

Cosa è il tlb e quale è il suo ruolo

Il **Translation Lookaside Buffer (TLB)** è una memoria cache **specializzata** presente nella CPU, il cui ruolo principale è quello di velocizzare la traduzione degli indirizzi virtuali in indirizzi fisici. Si tratta di un componente

essenziale nelle architetture che utilizzano la **gestione della memoria virtuale** e il **meccanismo di paginazione**, riducendo significativamente il tempo necessario per accedere alla memoria.

Per comprendere il ruolo del TLB, dobbiamo partire dalla gestione della memoria virtuale:In un sistema a memoria virtuale, i processi lavorano su indirizzi virtuali che devono essere convertiti in indirizzi fisici prima che i dati possano essere letti o scritti nella RAM. La traduzione degli indirizzi virtuali avviene tramite la **Tabella delle Pagine** (Page Table), che associa ogni pagina virtuale a un frame fisico in memoria. Tuttavia, accedere continuamente alla Tabella delle Pagine (che risiede nella memoria principale) è **costoso in termini di tempo**. Ogni accesso richiederebbe due operazioni di memoria: una per tradurre l'indirizzo virtuale e un'altra per accedere al dato vero e proprio. Il **TLB** risolve questo problema fungendo da cache per la Tabella delle Pagine. Quando il processore cerca di accedere a un indirizzo virtuale, verifica prima se l'informazione necessaria per la traduzione è già presente nel TLB.

- **TLB Hit:** Se il TLB contiene l'associazione tra la pagina virtuale richiesta e il frame fisico corrispondente, la traduzione è immediata e non è necessario accedere alla Tabella delle Pagine nella memoria.

- **TLB Miss:** Se il TLB non contiene l'associazione, il processore deve consultare la Tabella delle Pagine, caricare l'associazione mancante nel TLB e poi procedere con la traduzione.

In questo modo, il TLB riduce il numero di accessi alla memoria principale, migliorando le prestazioni del sistema. Nonostante i vantaggi, l'uso del TLB introduce alcune complessità:

- **Context Switch:** Quando il sistema operativo cambia il processo in esecuzione, il contenuto del TLB diventa obsoleto (poiché appartiene al processo precedente). Questo fenomeno è chiamato **TLB Flush** e può ridurre temporaneamente le prestazioni.

- **Page Table Walk:** In caso di TLB Miss, il tempo necessario per consultare la Tabella delle Pagine può essere elevato, specialmente in architetture con tabelle delle pagine multi-livello.

Descrivere il funzionamento del TLB nel contesto paging

Nel contesto della **paginazione** (paging), il **Translation Lookaside Buffer (TLB)** gioca un ruolo cruciale nella traduzione degli indirizzi virtuali in indirizzi fisici. La paginazione è una tecnica di gestione della memoria in cui lo spazio degli indirizzi virtuali di un processo è suddiviso in blocchi di dimensioni fisse, chiamati **pagine** (pages). Ciascuna pagina virtuale viene mappata a un **frame fisico** (frame) nella memoria fisica. Senza il TLB, ogni traduzione richiederebbe un accesso alla **Tabella delle Pagine** (Page Table), che è memorizzata nella memoria principale. Il TLB migliora le prestazioni memorizzando un sottoinsieme delle traduzioni più utilizzate, evitando così l'accesso ripetuto alla Tabella delle Pagine. Nel paging, ogni indirizzo virtuale è composto da due parti principali: **Numero della pagina virtuale** (**VPN**, Virtual Page Number): Identifica una pagina nel contesto dello spazio degli indirizzi virtuali. **Offset**: Specifica la posizione all'interno della pagina.

l TLB è una memoria cache che memorizza direttamente alcune delle traduzioni più recenti dalla VPN al PFN. TLB nel contesto del paging funziona nel seguente modo: **a. Generazione dell'indirizzo virtuale**,

b. Lookup nel TLB(**Caso di TLB Hit:** Se la VPN è presente nel TLB, il TLB restituisce immediatamente il corrispondente **PFN**. **Caso di TLB Miss:** Se la VPN non è presente nel TLB, il sistema deve consultare la Tabella delle Pagine nella memoria principale per ottenere la traduzione. La traduzione ottenuta viene poi inserita nel TLB (se c'è spazio o attraverso un meccanismo di rimpiazzo), per futuri utilizzi.)

c. Accesso alla memoria fisica.

L'uso del TLB nel contesto del paging offre numerosi vantaggi: **Riduzione del tempo di traduzione:** Con un **TLB Hit**, la traduzione è immediata, senza dover accedere alla memoria principale per consultare la Tabella delle Pagine. **Riduzione del carico sulla memoria:** Il TLB diminuisce il numero di accessi alla Tabella delle Pagine, liberando risorse della memoria principale. **Prestazioni migliorate:** Poiché la cache del TLB è estremamente veloce (di solito progettata per essere vicina al processore), il tempo medio di accesso alla memoria viene significativamente ridotto.

49) demand paging e memoria virtuale

La **memoria virtuale** è quando il SO crea l'illusione di avere più memoria di quella effettiva. L'obbiettivo è rendere il processo indipendente dalla capacità di memoria del sistema. La sua implementazione si basa sull'uso del disco e sulla paginazione.

Con la tecnica di **demand paging**:

- Si usa la paginazione, cioè la memoria è divisa in page frame e ciascun processo è suddiviso in pagine
- L'intero spazio logico dei processi è memorizzato su un paging device, che è un disco o una porzione di un disco
- L'area del paging device allocata per un processo è detta swap area di questo processo
- Quando inizia l'esecuzione di un processo, viene allocato solo un page frame nel quale viene caricata la pagina che contiene la prima istruzione
- Quando il processo in esecuzione cerca di accedere a una pagina alla quale non è assegnato alcun page frame, essa viene copiata dallo swap space a un frame libero
- Quando un processo modifica una pagina, la copia diventa obsoleta
- Se servono page frame, il SO ne libera alcuni, aggiornando eventuali copie obsolete.

Il demand paging si basa su alcuni concetti:

- **Page fault**: eccezione sollevata dalla MMU quando un processo cerca di accedere a una pagina che non è caricata in nessun frame;
- **Page-in**: il SO copia la pagina richiesta dallo swap space a un frame libero; se non esistono ne libera uno;
- **Page-out**: quando il SO libera un page frame, la pagina associata modificata dopo il suo page-in più recente, essa deve essere copiata nello swap device
- **Page replacement**: libera un page frame e nel successivo page-in di una pagina diversa nello stesso frame.

Il page-in e page-out formano il **page I/O** che è trasparente al programma. La movimentazione delle pagine può causare rallentamenti.

Per implementare il demand paging, è necessario che ciascuna entry della page table contenga più informazioni: **validity bit** (vale 1 se la pagina è caricata in un page frame), **frame number** (numero del page frame associato alla pagina, è significativo solo se validity bit=1), **disk address** (indirizzo della pagina sul paging device), **modified bit (mod)** (vale 1 se la pagina è stata modificata dopo l'ultimo page-in), **reference info (ref)** (info sugli accessi recenti alla pagina), **protection info (prot)** (permessi per accedere alla pagina in lettura/scrittura).

49bis) Definire le "regioni" necessitate per l'esecuzione di un programma, definire i concetti di indirizzi virtuali e reali dando la definizione di "MMU".

Le "regioni" necessitate a un programma per l'esecuzione sono:

Area testo: contiene il testo del programma

Area dati: contiene le variabili globali del programma

Area di stack: contiene i record di attivazione di tutte le procedure già chiamate ma non ancora terminate (a chiama b, b chiama c…).

Il codice di un programma contiene indirizzi di memoria che riferiscono queste tre aree citate. Questi indirizzi possono essere: indirizzi reali, in cui l'indirizzo visualizzato corrisponde al registro reale in cui il dato è contenuto, mentre indirizzi virtuali, sono indirizzi tradotti dalla MMU, memory management unit, in cui l'indirizzo reale corrisponde all'indirizzo visualizzato con in aggiunta un'"operazione".

50) come viene gestita la protezione della memoria

Se un processo prova ad accedere a un indirizzo al di fuori del suo spazio logico, si genera una **memory protection exception**. Questo controllo è semplice se l'architettura offre un registro chiamato **Page Table Size Register (PTSR)**, che memorizza il numero dell'ultima pagina del processo running: per ogni accesso alla memoria, l'hardware controlla che il numero di pagina sia minore o uguale al valore del PTSR, se no solleva un'eccezione. Se un processo cerca di eseguire un tipo di accesso per cui non ha il permesso (codificato dal campo prot), la MMU genera una memory protection exception. A ogni processo la MMU deve poter confrontare il valore di prot con il tipo diaccesso effettuato e il campo prot deve essere presente nel TLB.

51)descrivi il principio di localita

Il **principio di località** afferma che un indirizzo logico generato eseguendo un'istruzione ha probabilità elevata di essere vicino agli indirizzi logici generati da altre istruzioni recenti. Esso vale perché gli accessi alle parti di una struttura dati sono effettuati spesso in istruzioni ravvicinate e le istruzioni di salto sono non più del 10-20% del totale. Quando bisogna effettuare lo swap out di una pagina, per evitare i page fault, conviene sceglierla tra quelle visitate meno di recente. Per capire quali pagine sono state visitate di recente si sfrutta il campo *ref* della page table, che è un semplice bit: al momento del page-in, il bit ref della pagina caricata viene impostato a 1; a intervalli regolari, vengono azzerati i bit ref delle pagine; quando si accede a una pagina, il suo bit ref viene aggiornato a 1. Quindi se *ref*=1 la pagina è stata visitata di recente e in base al principio, il frame che la contiene non deve essere liberato.

52) COSA SUCCEDE SE LE PAGE TABLE SONO GRANDI?

Se le page table dei processi sono grandi, possono occupare porzioni significative della RAM, riducendo quindi il numero di pagine che possono essere caricate in memoria. Per risparmiare memoria, esistono 2 strategie:

1. **Inverted Page Table (IPT)**
2. **Doppia paginazione**

53) cosa si intende per ipt

L'IPT ha una entry per ogni frame che contiene l'indicazione che il frame è libero, oppure la coppia (PID, page number) se il frame è occupato. La dimensione dell'IPT è fissa, perché dipende dal numero di frame. Lo svantaggio è il costo della traduzione degli indirizzi: la MMU deve cercare tra tutti

i frame quello che ospita la pagina richiesta. Nel caso peggiore, quando la pagina non è caricata in memoria, si genera un page fault solo dopo una scansione completa dell'IPT. Per evitare ciò, si usa la **funzione hash**. Con questa funzione, ogni entry ha almeno 4 campi:

- Stato libero/occupato
- Coppia
- Frame number
- Puntatore a un'altra entry per la costruzione di una linked list

Si fissa un numero primo *a*, maggiore del numero di frame e si costruisce una **hash table** contenente *a* puntatori verso l'IPT. Questa hash table sfrutta la funzione hash *h:xx mod a*

Dato un numero intero x, essa da come risultato un altro intero h(x)=y, tale che $0 \leq y \leq a - 1$ con y valido per la hash table. Come input della funzione si può usare una coppia (P,p) di PID e page number: concatenando P e p, si ottiene una stringa di bit *h(Pp)=y*

Tutte le coppie con stesso hash y sono concatenate in una linked list. Il puntatore situato all'indice y della hash table punta alla prima entry di tale lista. Quando un processo con PID P cer a di accedere alla sua pagina p:

1. Viene calcolato l'hash h(Pp)=y
2. Si accede all'indice y della hash table
3. Se il puntatore situato all'indice y non è nullo, si accede tramite esso a una delle linked list formate dalle entry della IPT
4. Si esegue la scansione della lista: se si trova la entry corrispondente alla coppia, la ricerca termina e si usa il frame number ottenuto per costruire l'indirizzo fisico; se non si trova, significa che la pagina corrispondente non è caricata in memoria e viene generato un page fault.

54) cosa si intende per doppia impaginazione

Doppia paginazione: per evitare di avere in memoria page table di grandi dimensioni, si realizza una struttura a due livelli formata da page table di alto livello (higher level page table) che contengono gli indirizzi di page table di basso livello (lower level page table). Le lower level page table sono paginate, quindi si può effettuare lo swap out per liberare spazio in RAM. Con la doppia paginazione, la traduzione di un indirizzo avviene mediante l'accesso a due page table:

1. Si accede alla entry della higher level page table specificata dai bit più significativi dell'indirizzo logico, per ottenere l'indirizzo della lower level
2. All'interno della lower level page table situata all'indirizzo ottenuto, si accede alla entry specificata dai bit centrali dell'indirizzo logico che contiene il numero di frame in cui è caricata la pagina richiesta.

55) cosa è il tlb search ?

Il **TLB reach** è la quantità di RAM coperta dal TLB. Esso si calcola come prodotto tra la dimensione di una pagina e il numero di entry che il TLB può contenere: TLB reach=page size * TLB entries

Una tecnica per migliorare il TLB reach senza dover aumentare la dimensione di tutte le pagine è l'uso di **superpagine**: è come una pagina, ma ha dimensione pari a 2*page size. Quando ci sono più pagine contigue con accessi frequenti, il SO le promuove a una superpagina (promotion); se alcune pagine della superpagina non hanno più accessi, la superpagina viene smembrata, separando le singole pagine che la compongono (demotion).

36) PROBLEMA CLASSICO PROUTTORI E CONSUMATORI

Si hanno un pool finito di buffer (pieno o vuoto), un insieme di processi produttori (scrivono info nei buffer), un insieme di processi consumatori (leggono ed eliminano le info nei buffer).

Una soluzione è valida se l'accesso ai singoli buffer avviene in mutua esclusione, i produttori non possono riempire i buffer pieni (cioè sovrascrivere info non ancora lette dal consumatore), i consumatori non possono svuotare i buffer vuoti. È richiesta una politica FIFO, cioè i buffer devono essere svuotati nello stesso ordine in cui sono stati riempiti.

Il pool è una coda di stampa, i produttori sono processi utente, il consumatore è il daemon di stampa.

Una possibile soluzione è realizzare il pool con un array gestito in modo circolare, cioè lo si riempie e svuota nella stessa direzione. Si suppone che i buffer siano 100 e che ciascuno di essi possa contenere un numero intero. Il pool è quindi un array di 100 interi.

La variabile *count* indica il numero di buffer pieni; *i* è l'indice del primo buffer vuoto, se esiste; *j* è l'indice del primo buffer pieno, se esiste. Per gestire in modo circolare l'array, gli incrementi di i e j sono effettuati modulo 100. La variabile *item* è privata; *produce_item()* è un'operazione di costruzione di un'info da inserire in un buffer. Per prevenire la race condition, tutti gli accessi ai dati condivisi avvengono all'interno di sezioni critiche.

37) PROBLEMA DEI LETTORI E SCRITTORI

Si hanno un insieme di dati condivisi, un insieme di processi scrittori (modificano i dati) e un insieme di processi lettori (accedono ai dati in sola lettura). Una soluzione valida deve rispettare le seguenti regole:

- Sono consentite le letture concorrenti
- Non è ammessa la concorrenza tra le scritture
- Non è ammessa la concorrenza tra scrittura e lettura.

Prima di leggere, un reader deve aspettare che nessun writer stia scrivendo; un writer deve aspettare che nessun altro processo stia lavorando.

38) PROBLEMA DEI FILOSIFI

5 filosofi sono seduti al tavolo e alternano momenti in cui pensano a momenti in cui mangiano. Per mangiare hanno bisogno di due bacchette, ma sul tavolo ne sono disponibili solo 5: ciascun filosofo ne ha una condivisa con il filosofo alla sua destra e una con quello di sinistra. Quindi mentre un filosofo mangia, quello di sx e dx non possono mangiare.

Un filosofo attraversa ciclicamente 3 stati: pensa, ha fame, mangia.

Una soluzione valida deve far sì che ogni filosofo affamato riesca prima o poi a mangiare. I filosofi sono quindi processi che competono con altri per delle risorse condivise, le quali possono essere usate solo in mutua esclusione e ogni processo deve acquisire contemporaneamente più risorse per lavorare.