

**Descrivere brevemente alcuni modi per evitare di avere troppi livelli nel multilevel paging (per esempio su sistemi a 64 bit).**

Vi sono sostanzialmente due modi per risolvere il problema. Usare un sistema di indicizzazione mediante un hash, oppure ribrutare il problema e usare una frame table. Nel primo caso si usa una funzione di hash per ricavare un numero dal numero di pagina che stiamo cercando; questo numero viene usato come indice di un vettore; l'elemento selezionato è l'inizio di una lista concatenata di elementi, contenenti come "dati", la coppia numero di pagina/numero di frame; si scorre tutta la lista fino a trovare l'elemento desiderato. Nel secondo caso la frame table ha una dimensione fissa (visto che il numero massimo di frames è fissato dall'architettura) ed è unica (non cambia da un processo all'altro); ogni elemento della tabella contiene il numero di pagina e (per esempio) il process ID del processo a cui si riferisce; si scorre il vettore fino a trovare la pagina desiderata; l'indice dell'elemento della tabella è il numero di frame desiderato.

**Discutere in quale grado i seguenti algoritmi di gestione della cpu favoriscono il tempo di risposta: FCFS, SJF, SJF con preemption, Round Robin.**

Prima di cominciare, ricordiamo cosa si intende per tempo di risposta: non è il tempo necessario all'esecuzione di un programma. È il tempo necessario perché un programma cominci a rispondere a delle sollecitazioni esterne (p. es. quando scrivi con un editor di testo, la prima risposta del programma consiste nel far comparire i caratteri che state digitando sullo schermo. Questa risposta la volete il più in fretta possibile, mentre le altre funzionalità del programma possono richiedere più tempo). Questa quantità è particolarmente importante per i sistemi interattivi (vedi esempio precedente).

FCFS: chiaramente è il meno indicato. Se ci sono molti processi in coda ready, un nuovo processo deve aspettare la fine dell'esecuzione di tutti quelli davanti a lui nella coda.

SJF, SJF con preemption, RR: qui la situazione non è chiara. I primi due pongono l'accento sulla richiesta di cpu burst usandola come una priorità: un nuovo processo (o un processo che sta riprendendo l'elaborazione dopo l'attesa di un evento esterno) potrebbe essere avvantaggiato e entrare subito in esecuzione (almeno cominciare l'esecuzione). Anche nel caso del RR un processo appena arrivato potrebbe cominciare l'esecuzione velocemente, a seconda di quanti altri processi ci sono e della durata del quanto di tempo. In pratica non si può dire a priori quali tra questi tre sia il più indicato: quello più facile da implementare è il RR, e quindi viene usato abitualmente nei sistemi interattivi.

**Quando si dice che un insieme di processi è in stallo?**

Un insieme di N processi è in stallo quando sono soddisfatte contemporaneamente quattro condizioni: mutua esclusione (almeno una risorsa di ogni processo deve essere usata in modo "non sharable", cioè non può essere condivisa), hold and wait (i processi non chiedono tutte le risorse in una volta sola, ma le chiedono man mano che servono), assenza di preemption per le risorse (solo il processo che sta usando una risorsa la può rilasciare, rendendola nuovamente disponibile), attesa circolare (ogni processo dell'in-sieme ha bisogno di almeno una risorsa posseduta da un altro processo dell'insieme).

**Descrivere come viene calcolato l'indirizzo assoluto nel caso di memoria segmentata.**

Come nel caso della memoria paginata, occorre avere un'aiuto hardware: la MMU.

In questo caso l'indirizzo logico è formato da un numero di segmento e un offset.

A differenza della paginazione, i segmenti non hanno tutti la stessa dimensione, quindi i numeri di bit da dedicare all'offset sono calcolati sulla base della massima dimensione possibile per un segmento. Anche qui abbiamo una tabella (segment table) in cui si usa come indice il numero di segmento, mentre il contenuto dell'elemento selezionato ci restituisce l'indirizzo di memoria fisica a cui comincia il segmento. In più, visto che nella segmentazione la lunghezza di un segmento è importante, di solito il contenuto dell'elemento restituisce anche la dimensione di quel particolare segmento. Quindi le operazioni sono le seguenti: la MMU divide l'indirizzo in numero di segmento e offset; usa il numero di segmento per ricavare dalla tabella la posizione del segmento nella memoria fisica e dimensione di quel segmento; controlla che l'offset sia minore o uguale alla dimensione del segmento (altrimenti genera un page fault); ricombina offset e posizione dell'inizio del segmento in memoria per formare l'indirizzo fisico.

**Cosa si intende per frammentazione della memoria? Se possibile fate esempi.**

In generale si parla di frammentazione della memoria quando si ha della memoria che non è possibile usare. Si distingue tra frammentazione interna ed esterna a seconda che si tratti di memoria riservata in eccesso a quanto necessario, oppure di memoria non riservata che non si riesce a utilizzare perché è suddivisa in pezzi di piccole dimensioni non contigui tra loro. Un esempio di frammentazione interna si ha, per esempio, nel buddy system, con cui si alloca un blocco la cui dimensione è la potenza di 2 minore-tante superiore al quantitativo richiesto. Sempre nel buddy system si ha frammentazione esterna nel momento in cui sono liberi due blocchi da 64k non contigui e il sistema necessita di, per esempio, 100k.

**Descrivere brevemente in che modo avviene il calcolo dell'indirizzo assoluto in caso di segmentazione della memoria**

Come nel caso della memoria paginata, occorre avere un'aiuto hardware: la MMU. In questo caso l'indirizzo logico è formato da un numero di segmento e un offset. A

differenza della paginazione, i segmenti non hanno tutti la stessa dimensione, quindi i numeri di bit da dedicare all'offset sono calcolati sulla base della massima dimensione possibile per un segmento. Anche qui abbiamo una tabella (segment table) in cui si usa come indice il numero di segmento, mentre il contenuto dell'elemento selezionato ci restituisce l'indirizzo di memoria fisica a cui comincia il segmento. In più, visto che nella segmentazione la lunghezza di un segmento è importante, di solito il contenuto dell'elemento restituisce anche la dimensione di quel particolare segmento. Quindi le operazioni sono le seguenti: la MMU divide l'indirizzo in numero di segmento e offset; usa il numero di segmento per ricavare dalla tabella la posizione del segmento nella memoria fisica e dimensione di quel segmento; controlla che l'offset sia minore o uguale alla dimensione del segmento (altrimenti genera un page fault); ricombina offset e posizione dell'inizio del segmento in memoria per formare l'indirizzo fisico.

**Descrivere cos'è il TLB (translation lookaside buffer, qualche volta chiamato associative memory) e perché lo si usa.**

Con la paginazione della memoria, ogni richiesta di accesso alla memoria comporta due accessi alla memoria: uno per leggere la page table, e uno per l'accesso vero e proprio. Il TLB è una specie di cache che contiene gli elementi della page table che sono stati usati più recentemente. Il TLB è costituito da dell'hardware dedicato: memoria associativa, di accesso più veloce della memoria ordinaria. Deve essere riarzzerato ad ogni context switch.

**Descrivere brevemente cosa sono gli i-node.**

Gli inode sono il sistema usato principalmente dai sistemi operativi di tipo unix per gestire l'allocazione dello spazio disco. Si tratta di un sistema di allocazione di tipo indexed, cioè con una tabella (index table) che contiene l'elenco dei settori utilizzati, che utilizza più di un settore per ottenere una tabella a più livelli (schema noto come "ombedded scheme"). In pratica un inode contiene i riferimenti ai primi settori del file (per esempio i primi 15 settori); poi contiene dei riferimenti (per esempio tre riferimenti) ad altri inode che rappresentano i vari livelli aggiuntivi. Il primo livello è un altro inode che contiene riferimenti a settori. Il secondo livello è un inode che contiene riferimenti ad altri inode i quali contengono riferimenti a settori. Il terzo livello è un inode con riferimenti ad altri inode, che contengono riferimenti ad altri inode, i quali finalmente contengono il riferimento ai settori. Per certi versi questo tipo di schema è simile a quello usato per la multilevel page table.

**Cosa si intende per starvation in un algoritmo di scheduling?**

Con questo termine (che letteralmente si traduce in "morte per fame"), si indica la situazione in cui un processo che è pronto (cioè in coda ready) non riesce mai ad andare in esecuzione. Un esempio si ha con l'algoritmo SJF nel caso in cui si abbia un processo cpu-bound e una miriade di processi I/O-bound. Poiché il processo cpu-bound ha (in media) dei cpu burst più lunghi dei processi I/O-bound, e poiché l'algoritmo favorisce i processi con cpu-burst minore, il processo cpu-bound potrebbe trovarsi sempre scavalcato da uno degli altri processi e non riuscire ad andare mai in esecuzione.

**Si consideri un sistema interattivo che utilizza scheduling di tipo round robin con quanto di tempo X. Quanto deve essere X per massimizzare l'efficienza (throughput) del sistema? Spiegate perché questa soluzione non è adatta per lo scheduling di un sistema interattivo.**

Come abbiamo visto anche in uno degli esercizi fatti a lezione, per massimizzare l'efficienza occorre ridurre al minimo il tempo perso a causa dei context switch. Questo si ottiene con un tempo X infinito (cioè con i context switch solo quando un processo è finito o in pausa e si passa al successivo). Questo però vuol dire che un processo cpu-bound non rilascerà la cpu, e quindi il sistema non può rispondere ad eventuali richieste di interazione da partedell'utilizzatore. Questo modo di procedere chiaramente non è adatto ad un sistema interattivo.

**Descrivere brevemente cosa succede durante un context switch (cambio di contesto).**

Con context switch si intende la transizione da uno stato ad un altro di un processo (di solito perché si cambia il processo in esecuzione). Quando questo avviene, il contesto nel quale il processore lavora (registri, process state, memory info) viene modificato. In generale la cpu sospende l'esecuzione del processo, salva il suo PCB, carica il PCB di un altro processo e riprende l'esecuzione del nuovo processo secondo le informazioni del suo PCB.

**Cosa si intende quando si dice che un programma è "rilocabile"? Perché si usano programmi rilocabili?**

Un programma si dice "rilocabile" quando fa riferimento a locazioni di memoria virtuali.

In pratica considera la sua memoria come se iniziasse dalla locazione 0. All'atto dell'esecuzione o del caricamento in memoria, il sistema operativo stabilisce quale porzione di memoria il programma userà realmente e provvederà a trasformare gli indirizzi rilocabili in indirizzi assoluti. La necessità di avere codice rilocabile deriva dal volere implementare il multiprogramming assegnando a ciascun processo un suo spazio di memoria. Dato che non è noto al momento della compilazione qual'è la zona di memoria a disposizione, si ricorre al codice rilocabile e a un algoritmo che traduca

gli indirizzi del programma in indirizzi "veri".

#### Descrivere brevemente cos'è una system call.

Una system call sostanzialmente è una interfaccia per permettere al processo di contattare il sistema operativo.  
(oppure) Si tratta di un interrupt software mediante il quale un programma richiama le funzioni del sistema operativo.

In pratica le system call vengono eseguite nel monitor mode del processore per avere accesso a tutte le istruzioni privilegiate e all'intero sistema (al contrario delle funzioni che vengono eseguite in user mode). Le system call sono fornite dal sistema operativo stesso.

#### Cosa si intende per "working set" e qual'è la relazione con il thrashing.

Formalmente il working set al tempo  $t$  (W.D.  $t$ ) è definito come il numero di pagine utilizzate nelle ultime  $D$  unità di tempo. Chiaramente il working set è un indice della località di un programma. Se il sistema operativo assegna ad un programma un numero di frames sufficienti a contenerne il working set, non c'è thrashing in quanto la sua località può essere caricata in memoria. Invece nel caso in cui il numero di frames non è sufficiente a contenerne la località ci dobbiamo aspettare una maggiore attività di swap che può degenerare in thrashing.

#### Cos'è (a cosa serve) la Memory Management Unit?

Si tratta di un dispositivo hardware (di solito un chip, oppure una parte di un chip del cosiddetto chipset della scheda madre) che serve per aiutare il sistema operativo nel mapping indirizzo virtuale - indirizzo fisico, rendendo di fatto possibile la allocazione. Nella implementazione più semplice la MMU non fa altro che prendere l'indirizzo di memoria che gli comunica la CPU, sommare ad una parte di questo indirizzo (di solito la più significativa) il contenuto di un suo registro e utilizzare il numero così ottenuto come indirizzo fisico.

#### Che cos'è la FAT e per quale motivo la si usa?

La File Allocation Table è un metodo per l'allocazione dello spazio disco, una variante del metodo della linked list. Una porzione di disco all'inizio di ogni partizione viene lasciata da parte per contenere la tabella. Questa contiene una entry per ogni settore, indicizzata attraverso il numero del settore.

#### Cosa si intende quando si dice che un programma è "rilocabile"?

Un programma si dice "rilocabile" quando fa riferimento a locazioni di memoria virtuali. In pratica considera la sua memoria come se iniziasse dalla locazione 0. All'atto dell'esecuzione o del caricamento in memoria, il sistema operativo stabilisce quale porzione di memoria il programma userà realmente e provvederà a trasformare gli indirizzi rilocabili in indirizzi assoluti.

#### Cosa si intende per "starvation" in un algoritmo di scheduling? Fare un esempio.

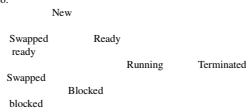
Con questo termine (che letteralmente si traduce in "morte per fame"), si indica la situazione in cui un processo che è pronto (cioè in coda ready) non riesce mai ad andare in esecuzione. Un esempio si ha con l'algoritmo SJF nel caso in cui si abbia un processo cpu-bound e una miriade di processi I/O-bound. Poiché il processo cpu-bound ha (in media) dei cpu burst più lunghi dei processi I/O-bound, e poiché l'algoritmo favorisce i processi con cpu-burst minore, il processo cpu-bound potrebbe trovarsi sempre scavalcato da uno degli altri processi e non riuscire ad andare mai in esecuzione.

#### Spiegare cos'è una "inverted page table", perché la si usa, vantaggi e svantaggi.

Una inverted page table è una page-table indicizzata mediante il numero di frame, e non mediante il numero di pagina, che in questo caso è il contenuto della tabella. Dato che i frame (cioè la memoria fisica) sono unici, si ha una sola tabella per tutti i processi, e questo comporta che nel contenuto della tabella oltre al numero di pagina ci sia anche una informazione sul processo a cui appartiene quel frame (per esempio il process id). Il vantaggio è che c'è una sola tabella (anche se con un numero di elementi dipendente dalla massima memoria fisica indirizzabile), e questo può risultare vantaggioso, per esempio, nel caso di un sistema operativo a 64 bit in cui una tabella di paging multilivello comporterebbe un numero eccessivo di livelli (con conseguente calo di prestazioni). Lo svantaggio consiste nel fatto che, essendo la tabella indicizzata per numero di frame e la ricerca effettuata per numero di pagina, può essere necessario scorrere tutta la tabella prima di trovare l'elemento desiderato.

#### Descrivere in che modo l'aggiunta dello swapper modifica i possibili stati di un processo.

Sostanzialmente si passa da una situazione in cui gli stati possibili di un processo sono 5, a una situazione con 7 stati. Le due aggiunte sono: swapped ready e swapped blocked. Il primo si riferisce a processi che sono pronti per andare in esecuzione ma che sono stati tolti dalla memoria RAM del computer e messi su storage secondario (di solito disco rigido). Il secondo si riferisce a processi che sono in attesa di un qualche evento, e che comunque si trovano su storage secondario. Le possibili transizioni sono:



#### Cosa si intende per frammentazione della memoria? Se possibile fare esempi.

In generale si parla di frammentazione della memoria quando si ha della memoria che non è possibile usare. Si distingue tra frammentazione interna ed esterna a seconda che si tratti di memoria riservata in eccesso a quanto necessario, oppure di memoria non riservata che non si riesce a utilizzare perché è suddivisa in pezzi di piccole dimensioni non contigui tra loro. Un esempio di frammentazione interna si ha, per esempio, nel buddy system, con cui si alloca un blocco la cui dimensione è la potenza di 2 immediatamente superiore al quantitativo richiesto. Sempre nel buddy system si ha frammentazione esterna nel momento in cui sono liberi due blocchi da 64k non contigui e il sistema necessita di, per esempio, 100k.

#### Descrivere brevemente i vantaggi e gli svantaggi dei tre metodi fondamentali di allocazione dei file su disco.

Allocazione contigua: ogni file è allocato in un insieme di blocchi contigui. Vantaggi: l'accesso ai vari blocchi è veloce ed è possibile sia l'accesso sequenziale che diretto. Svantaggi: occorre trovare spazio contiguo per ogni file (frammentazione) e si rende necessaria una ricompattazione periodica. Quando il file aumenta di dimensione ci sono problemi.

Allocazione concatenata (sequenziale): ogni blocco contiene un puntatore al blocco successivo. Vantaggi: non c'è frammentazione esterna, non serve ricompattare. Svantaggi: l'accesso diretto è altamente inefficiente; se un blocco viene danneggiato si rischia di perdere tutta la parte finale di quel file.

Allocazione indicizzata: per ogni file si usa un blocco come indice, contenente cioè la lista dei blocchi usati per quel file. Vantaggi: niente frammentazione e possibilità di accesso diretto e sequenziale. Svantaggi: è necessaria una struttura multilivello quando un singolo blocco non è sufficiente come indice.

#### Spiegare cosa si intende per "deadlock"

Si intende una situazione di stallo di un gruppo di processi (o dell'intero sistema). Questa situazione si verifica quando si verificano contemporaneamente 4 condizioni: mutua esclusione, attesa circolare, risorse non pre-entable, hold and wait

#### Descrivere brevemente cosa sono i semafori a cosa servono.

I semafori servono per poter avere una sincronizzazione tra due o più processi. I semafori sono un meccanismo di sincronizzazione "generale" e il loro uso non garantisce che l'accesso a memoria condivisa sia mutuamente esclusivo o l'eliminazione dei deadlock. Questo perché il modo in cui vengono usati dipende dal programmatore (cioè non ci sono protezioni contro errori di programmazione). I semafori sono forniti dal sistema operativo e concettualmente si può considerare come una variabile intera su cui possono essere svolte solo due operazioni indivisibili (atomiche): wait e signal. Signal incrementa di uno il valore della variabile. Wait controlla se la variabile è maggiore di 0, se è maggiore, la decrementa e prosegue; se la variabile è minore o uguale a zero aspetta (questo viene fatto senza spreco di cicli di cpu, cioè il processo viene messo in attesa).

#### Spiegare cosa si intende per "processo" e a cosa serve il PCB.

Per processo si intende un programma in esecuzione. Formalmente un processo è una coppia di elementi: il programma e lo stato dell'esecuzione. Per stato dell'esecuzione si intende un insieme di informazioni tra cui i valori dei registri, della memoria in uso, dei dispositivi di I/O, ecc. Per poter gestire più processi, ad ogni processo viene associato un descrittore (cioè un insieme di informazioni) detto PCB che contiene le informazioni sullo stato dell'esecuzione più, eventualmente, altre informazioni necessarie al sistema operativo (per es.: user id e group ID da usare durante l'esecuzione, parametri di scheduling per quel processo, ecc.)

#### Dei seguenti elementi, dire quali sono condivisi tra thread di uno stesso processo e quali invece sono privati:

- file aperti
- spazio di indirizzamento (memoria contenente i dati)
- stack
- program counter

Dire inoltre se esistono dei meccanismi che permettono ai thread di proteggere gli eventuali elementi privati da altri thread dello stesso processo. In caso affermativo spiegare quali sono tali meccanismi; in caso negativo spiegare perché non esistono.

a) e b) sono condivisi. c) e d) sono privati. Non esistono meccanismi di protezione: si suppone che, facendo parte dello stesso processo, i thread siano programmati in modo da non interferire tra loro.

Supponiamo di avere due processi, A e B, che effettuano operazioni di I/O su disco rigido. Il processo A esegue un numero elevato di richieste verso settori compresi tra i cilindri 10 e 50 mentre il processo B effettua un numero più ridotto di accessi verso settori compresi tra i cilindri 90 e 130. Illustrare brevemente cosa succede nel caso si usino come politiche di scheduling delle richieste del disco SSTF e SCAN.

Con SSTF si rischia una più o meno totale starvation per uno dei due processi: infatti se SSTF sta servendo richieste del processo A e altre richieste del processo A continuano ad arrivare, l'algoritmo privilegerà queste in quanto più vicine alla sua posizione attuale, costringendo all'attesa il processo B. Ovviamente questa situazione può essere completamente simmetrica rispetto ai due processi.

Nel caso di SCAN invece i due processi verranno trattati in maniera sostanzialmente equa e non c'è un rischio di starvation.

Si consideri un sistema operativo che usa per lo scheduling della CPU l'algoritmo Round Robin. Raddoppiare la durata del quanto di tempo automaticamente migliora le prestazioni? Come si può scegliere il quanto di tempo?

Raddoppiare il quanto di tempo non significa automaticamente migliorare le prestazioni. In generale si vuole avere un quanto di tempo che sia maggiore del tempo di switch tra due processi in esecuzione e minore del cpu burst medio dei processi.

In quale grado i seguenti algoritmi di gestione dei processi favoriscono processi corti (che richiedono poco tempo di elaborazione)

a) FCFS b) Shortest Job First c) SJF con preemption d) Round Robin e) multilevel feedback queues

a) Il fifo non favorisce i processi corti, anzi tende a sfavorirli: se un processo corto viene messo in coda dopo un processo lungo, è costretto ad aspettare la fine del processo lungo.

b) SJF favorisce i processi corti, in quanto ci si aspetta che questi abbiano dei burst time più piccoli di processi lunghi o "cpu intensive".

c) SJF con preemption favorisce ancora di più i processi corti in quanto la preemption può forzare il riposo di un lungo processo in esecuzione anche senza aspettare la fine del suo cpu burst.

d) il round robin non favorisce né sfavorisce i processi corti rispetto a quelli lunghi.

Vengono tutti trattati allo stesso modo.

e) in questo caso dipende da come è esattamente implementata. Un sistema con un paio di RR a priorità decrescenti seguiti da un FCFS come il sistema visto a lezione o spiegato sul libro di testo favoriscono molto limitatamente i processi corti (non tanto quanto il SJF).

Discutere in quali modi si può cercare di prevenire le situazioni di deadlock.

Affinché ci sia un deadlock devono essere soddisfatte contemporaneamente le seguenti 4 condizioni: non c'è la preemption delle risorse, i processi richiedono le risorse man mano che queste gli servono, le risorse vengono usate in esclusiva, esiste una situazione di attesa circolare. Per prevenire i deadlock occorre far sì che (almeno) una di queste condizioni non valga.

L'esclusività non è sempre eliminabile in quanto non tutte le risorse possono essere condivise: per esempio un file può essere condiviso in lettura, ma una richiesta di scrittura necessita della mutua esclusione.

Anche la mancanza di preemption non è facilmente eliminabile in quanto non è sempre possibile portare via di forza una risorsa a un processo. In alternativa si può togliere una risorsa di forza solo se l'altro processo è in attesa di altre risorse. Questo però è fattibile solo per risorse il cui stato può essere facilmente salvato e ripristinato più tardi come la CPU o la memoria.

Il fatto che i processi richiedano le risorse solo quando ne hanno bisogno può essere aggirato forzando i processi ad allocare subito tutte le risorse di cui hanno bisogno (uso non efficiente delle risorse, senza contare che non sempre un processo sa a priori di cosa avrà bisogno), oppure obbligando il processo, ogni volta che necessita di una nuova risorsa, a rilasciare tutte le risorse già in suo possesso per richiederle tutte in blocco.

Per finire l'attesa circolare può essere eliminata costruendo una gerarchia delle risorse e effettuando una gestione gerarchica.

Un computer ha uno spazio di indirizzi virtuali a 64 bit, e pagine di 2048 bytes (211). Una entry nella page table richiede 4 bytes. Se il sistema usa multilevel paging perché ogni tabella deve essere contenuta in una pagina, quanti livelli sono necessari?

Considerando una page table grande quanto una pagina, il numero massimo di entries che questa può contenere è dato da  $2^{114} = 2^{11} \cdot 2^{103} = 2^9$ . Questo significa che, al massimo, un livello può usare 9 dei bit destinati al numero di pagina. Il numero totale di bits destinati al numero di pagina è  $64 - 11 = 53$ . Il numero di livelli è dato da  $53 / 9$  arrotondato all'intero superiore: 6 livelli.

Cosa sono i threads?

Per thread si intende una unità di impiego della cpu all'interno di un processo (o un flusso di esecuzione all'interno di un processo). In pratica si tratta di una entità (nel senso di parte di un programma) che viene messa in esecuzione. Si possono così avere più thread in esecuzione all'interno dello stesso processo. La situazione è simile all'esecuzione di più processi in contemporanea, con la differenza che le thread riguardano lo stesso processo e quindi hanno in comune tutte le risorse (a partire dalla memoria).

2) Cosa si intende per frammentazione della memoria? Se possibile fate esempi.

In generale si parla di frammentazione della memoria quando si ha della memoria che non è possibile usare. Si distingue tra frammentazione interna ed esterna a seconda che si tratti di memoria riservata in eccesso a quanto necessario, oppure di memoria non riservata che non si riesce a utilizzare perché è suddivisa in pezzi di piccole dimensioni non contigui tra loro. Un esempio di frammentazione interna si ha, per esempio, nel buddy system, con cui si alloca un blocco la cui dimensione è la potenza di 2 immediatamente superiore al quantitativo richiesto. Sempre nel buddy system si ha frammentazione esterna nel momento in cui sono liberi due blocchi da 64k non contigui e il sistema necessita di, per esempio, 100k.

Illustrate brevemente i metodi fondamentali di allocazione della memoria, citandone le varianti principali.

Partizioni fisse: primi sistemi operativi, primi sistemi multiprogramming; occorre impostare le dimensioni delle partizioni.

Partizioni multiple: evoluzione delle partizioni fisse; le partizioni vengono decise durante lo svolgersi dei vari programmi. Con le partizioni multiple si possono avere: best fit, first fit, worst fit, buddy system.

Per tenere sotto controllo i problemi di frammentazione si ricorre a paginazione e segmentazione.

Elencate sinteticamente i vantaggi e svantaggi della segmentazione della memoria.

La segmentazione è una alternativa alla paginazione per l'allocazione della memoria.

Come la paginazione ha il vantaggio di eliminare la frammentazione interna (i segmenti sono della misura strettamente necessaria) e fornire un meccanismo per la protezione della memoria; allo stesso tempo si può avere della frammentazione esterna (due segmenti quasi contigui, con dello spazio libero tra i due che non si riesce ad usare).

Descrivere che differenze vi sono tra kernel threads e user threads.

Le user threads si ottengono con l'uso di librerie apposite (es. pthread) esterne al S.O. che forniscono tutto il supporto per la creazione, gestione e scheduling. Il S.O. non è a conoscenza delle threads: per il S.O. esistono solo i processi. Viceversa nelle kernel threads è il S.O. stesso a mettere a disposizione il supporto per le threads mediante system call specifiche. Le user threads si possono implementare su qualunque S.O. (nel senso che non serve un supporto specifico del S.O., basta "portare" la libreria in quel particolare S.O.), dispongono di un contesto switch tra le thread estremamente veloce, ma non portano vantaggi nel caso di elaboratore multiprocessore e la chiamata ad una system call bloccante da parte di una thread provoca la messa in attesa di tutto il processo. Per contro le kernel threads possono sfruttare appieno un elaboratore multiprocessore eseguendo threads diverse su cpu diverse e la chiamata ad una system call da parte di una thread provoca la messa in attesa solamente di quella thread.

Discutere in che cosa consistono e che obiettivo hanno gli algoritmi di scheduling dei dischi: FCFS, SSTF, SCAN.

FCFS: cerca le tracce esattamente nello stesso ordine con cui sono arrivate le richieste (fifo). Non assegna nessuna priorità e tratta tutte le richieste allo stesso modo.

SSTF: cerca di minimizzare il seek time scegliendo ogni volta la traccia più vicina alla posizione attuale tra quelle nella lista delle richieste. Assicura il minor seek time, ma può portare a starvation ed è sensibile a tempo di inversione del moto delle testine.

SCAN: si spazzola in continuazione il disco da un estremo all'altro per poi invertire il senso di marcia. In questo modo si minimizza il numero di volte che si inverte il moto delle testine, e si evita starvation.

Si dica che cos'è, nell'ambito della gestione della memoria virtuale, il bit di validità.

Si dica poi dove viene memorizzato e come viene utilizzato.

Il bit di validità viene associato a ciascun elemento della tabella delle pagine. Tale bit impostato a "valido", indica che la pagina corrispondente è nello spazio d'indirizzi logici del processo, quindi è una pagina valida; impostato a "non valido", indica che la pagina non è nello spazio d'indirizzi logici del processo. Il bit di validità consente quindi di riconoscere gli indirizzi illegali e di notificarne la presenza attraverso un segnale di eccezione al sistema operativo (riferimento di pagina non valido)

Descrivere in che modo avviene il calcolo dell'indirizzo assoluto in una memoria paginata.

L'indirizzo virtuale è composto da un page number e un offset. Partendo dal page number, si ricerca il numero di frame fisico corrispondente prima usando il TLB (se il sistema ne è provvisto) e poi usando la page table (la page table è sostanzialmente un vettore e il numero di pagina viene usato come indice dell'elemento del vettore da usare). A seconda dei casi oltre al numero di frame può esserci un identificatore (ASID) per essere certi che la pagina appartenga al processo, e/o un bit di validazione della pagina. Inoltre può anche esserci un registro che mi indica quanto è grande la page table e che mi evita di "sfiorare" oltre i limiti assegnati al processo (di solito indicato come PTLR, page table length register). Alla fine il numero di frame così ottenuto viene sostituito al page number nell'indirizzo di partenza per ottenere l'indirizzo fisico.

