

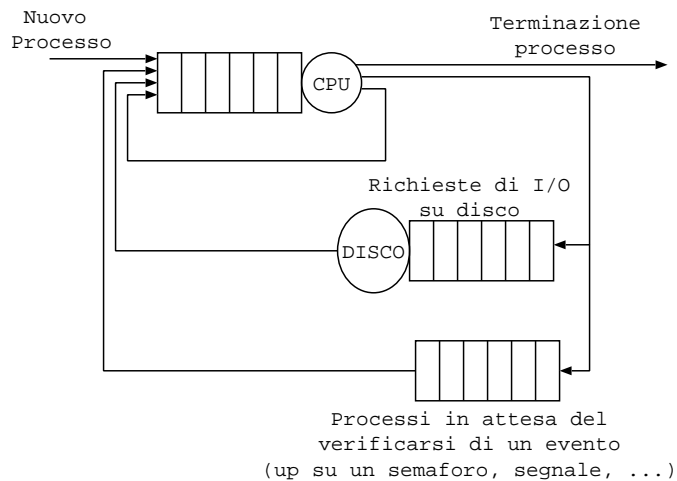
Corso di Laurea in Informatica
Corso di Sistemi Operativi I
Soluzioni del Compitino del 4/2/2000

Corso di Laurea in Informatica
Corso di Sistemi Operativi I
Compito del 4/2/2000

PRIMA PARTE

Esercizio 1

Si consideri lo schema (semplificato) in figura: si tratta di una rappresentazione (non molto dettagliata) di un sistema di calcolo, visto come insieme di risorse (CPU, DISCO, ...) che sono a disposizione di clienti (i processi) che entrano nel sistema, richiedono servizi, si sospendono in attesa del verificarsi di eventi, ed infine escono dal sistema. Poiché le risorse sono limitate può accadere che i processi debbano attendere in coda per ottenere un servizio da parte di una risorsa.



- In quale stato si trovano i processi nella coda associata alla risorsa CPU?
- In quale stato si trova il processo che sta ottenendo il servizio richiesto dalla CPU?
- In quale stato si trova il processo che sta ottenendo il servizio richiesto dal DISCO?
- In quale stato si trovano i processi nella coda associata alla risorsa DISCO?
- In quale stato si trovano i processi nella coda di attesa di un evento?
- A quale struttura dati del Sistema Operativo corrisponde la coda associata alla risorsa CPU?
- A quale struttura dati del Sistema Operativo corrisponde la coda associata alla risorsa DISCO?
- Come si può calcolare il livello di programmazione a partire da uno stato del sistema descritto sullo schema dato sopra (ovvero in termini di numero di richieste presenti in ciascuna coda)?
- Come si chiama il componente del sistema operativo che sceglie il primo processo da servire fra quelli in coda alla CPU?
- Come si chiama il componente del sistema operativo che predispone tutti i registri della CPU (ovvero che effettua il *context switch* nel momento in cui viene terminato il servizio di un processo ed iniziato il servizio di un altro processo in coda)?

Risposta:

- a) ready
- b) running
- c, d, e) waiting (o blocked)
- f) alla ready queue
- g) la coda delle richieste pendenti per quel disco (gestita dal driver del disco)
- h) corrisponde al numero di *clienti* presenti nel sistema, quindi il livello di multiprogrammazione é dato dalla somma degli elementi in coda piú gli elementi che stanno ottenendo servizio (nell'ipotesi che le system call corrispondenti alle operazioni di I/O siano bloccanti e che vi possa essere al piú una richiesta di I/O pendente per ogni processo)
- i) scheduler
- l) dispatcher

Esercizio 2

In che senso si può dire che il funzionamento di un sistema operativo é *guidato dagli interrupt*?

Risposta: il funzionamento del sistema operativo é guidato dagli interrupt nel senso che una volta avviato il sistema (fase di bootstrap), tutte le richieste di servizio da parte degli utenti o da parte dei vari dispositivi collegati al sistema avvengono attraverso interrupt (stiamo supponendo che tutte le interazioni con i dispositivi avvengano tramite interrupt: in qualche caso cio' potrebbe non essere vero, nel senso che per qualche device il sistema potrebbe effettuare un polling periodico).

Quando un processo ha bisogno di un servizio dal sistema operativo, esegue una system call, che causa una *trap*, ovvero un interrupt software, a cui il sistema operativo reagisce eseguendo il servizio richiesto.

L'avvicendamento dei processi nell'uso della CPU avviene o a causa di una system call (trap) bloccante, o a causa dell'arrivo di un interrupt dal timer (fine del time slice).

Quando un dispositivo ha bisogno dell'attenzione del sistema (per esempio quando é stata completata una operazione di I/O su disco) cio' viene segnalato attraverso un interrupt che verrà gestito dall'opportuno driver, e che causerá la riattivazione del processo in attesa della fine dell'operazione.

Esercizio 3 Consideriamo un sistema operativo che supporta i *thread*. Indicare quali delle seguenti affermazioni sono vere, e quali sono false, motivando la risposta:

- a) ogni processo é composto da uno o piú thread;
- b) i thread appartenenti ad uno stesso processo devono dichiarare esplicitamente quali strutture dati intendono condividere;
- c) esiste un solo stack condiviso da tutti i thread di uno stesso processo;
- d) é piú costoso il context switch fra thread di processi diversi del context switch fra thread dello stesso processo.
- e) non ci possono mai essere situazioni di *corsa critica* fra thread di uno stesso processo.

Risposta:

- a) vero, i thread permettono di realizzare in modo naturale applicazioni concorrenti, costituite da piú flussi di controllo interagenti (nel caso dei thread l'interazione avviene attraverso la condivisione dell'immagine in memoria).
- b) falso, solitamente tutte le strutture dati di uno stesso processo sono condivise da tutti i thread dello stesso processo

- c) falso, deve esserci uno stack per ogni thread dato che ciascuno di essi esegue indipendentemente, in modo asincrono rispetto agli altri thread, e ciascuno può richiamare procedure. Quindi ogni thread avrà il suo program counter, stack pointer, ecc. I thread di uno stesso processo invece condivideranno la stessa tabella delle pagine.
- d) vero, costa di più il context switch tra thread di processi diversi perché in questo caso cambia lo spazio di indirizzi, mentre se avviene il context switch tra thread di uno stesso processo tutte le informazioni necessarie alla MMU per effettuare la traduzione da indirizzo logico a indirizzo fisico non devono essere aggiornate.
- e) falso, dato che c'è condivisione di memoria e che i thread eseguono in modo asincrono e indipendente, si possono verificare situazioni di corsa critica.

Esercizio 4

Cinque processi $P1, \dots, P5$ sono in *ready queue* e chiedono di eseguire un CPU-burst di durata 9, 6, 3, 5 e 4 unità di tempo rispettivamente. In quale ordine devono essere schedulati (ovvero quale algoritmo di scheduling si deve usare) se si vuole minimizzare il tempo medio di risposta? Qual è il valore minimo di tale tempo medio di risposta? Mostrare l'ordine di schedulazione ottimo attraverso un diagramma di Gantt. Scegliere un qualsiasi altro algoritmo di scheduling e far vedere che effettivamente il tempo medio di risposta è più alto.

Risposta: Quando le richieste arrivano tutte insieme, l'algoritmo che ottimizza (minimizzando) il tempo medio di risposta (inteso come tempo dall'arrivo della richiesta al termine del servizio) è lo SJF (Shortest Job First).

L'ordine in cui i processi vengono schedulati dallo SJF è:

$[0, 3)P3, [3, 7)P5, [7, 12)P4, [12, 18)P2, [18, 27)P1$

il tempo medio di *turnaround* è:

$$\bar{T} = \frac{3 + 7 + 12 + 18 + 27}{5} = 13.4$$

Se si fosse usato un altro algoritmo, il tempo medio di *turnaround* sarebbe stato certamente maggiore, per esempio con il FIFO si il diagramma di Gantt sarebbe stato il seguente:

$[0, 9)P1, [9, 15)P2, [15, 18)P3, [18, 23)P4, [23, 27)P5$

corrispondente ad un tempo medio di turnaround di

$$\bar{T} = \frac{9 + 15 + 18 + 23 + 27}{5} = 18.4$$

Esercizio 5

Si consideri la seguente soluzione al problema di n produttori ed m consumatori che condividono una struttura dati (una lista che non può contenere più di N elementi) per scambiarsi informazioni. I dati prodotti da un qualsiasi produttore possono essere elaborati da qualsiasi consumatore. Il modo in cui sono implementate le operazioni di inserzione/prelievo richiede che tali operazioni siano effettuate in modo mutualmente esclusivo. Per sincronizzarsi i processi usano tre semafori: *PIENE* inizializzato a 0, *VUOTE* inizializzato a N , e *MUTEX* inizializzato a 1.

La soluzione presenta un problema: quale?

Produttore_i	Consumatore_j
<pre>while (true) begin <produci dati> down(MUTEX) down(VUOTE) inserzione(dati) up(PIENE) up(MUTEX) end</pre>	<pre>while (true) begin down(MUTEX) down(PIENE) prelievo(dati) up(VUOTE) up(MUTEX) <elabora dati> end</pre>

Risposta: Questa soluzione può facilmente portare a deadlock: supponiamo per esempio che uno dei consumatori esegua la prima *down(MUTEX)* in una situazione di buffer vuoto (semaforo PIENE uguale a 0): il semaforo MUTEX viene portato a 0, poi il processo si sospende sulla *down(PIENE)*. Siamo così giunti ad una situazione di deadlock, infatti gli unici processi che possono incrementare il semaforo PIENE sono i produttori, che però non possono entrare in sezione critica (perché MUTEX vale 0).

L'altra possibile situazione di deadlock (simmetrica alla precedente) si ha quando un produttore esegue la *down(mutex)* in una situazione di buffer pieno (semaforo VUOTE=0).

Il problema si risolve semplicemente scambiando le prime due down sia nei produttori che nei consumatori.

Esercizio 6

a) Un modo per risolvere il problema della sezione critica, è di eseguire una procedura *enter_section* prima di entrare in sezione critica e una *exit_section* in uscita dalla sezione critica. Una possibile implementazione di tali procedure è riportata qui di seguito:

```
enter_section:
    tsl register, lock    // copia il contenuto di lock in register, e imposta lock a 1
    cmp register, #0     // confronta il valore nel registro con la costante 0
    jne enter_section    // se il registro non contiene 0, cicla (il lock valeva 1,
                        // quindi la sezione critica e' occupata
    retn                 // ritorna dalla procedura
exit_section:
    move lock, #0        // imposta lock a 0
    retn                 // ritorna dalla procedura
```

La istruzione assembler *tsl registro, lock* (Test and Set Lock) copia nel *registro* il valore della parola di memoria *lock*, e contemporaneamente (in modo non interrompibile) scrive il valore 1 in *lock*. La variabile *lock* è condivisa fra i processi/thread che devono eseguire una sezione critica in mutua esclusione ed è inizializzata a 1.

Supponiamo ora che un dato processore non disponga della istruzione *tsl*, ma abbia invece una istruzione *swap register, lock* capace di scambiare in modo atomico il contenuto di *register* e quello di *lock*. E' ancora possibile implementare le procedure *enter_section* ed *exit_section*? Se sì come?

b) Il codice che realizza le operazioni *up* e *down* su un semaforo costituisce una sezione critica: per garantire l'atomicità di tali operazioni è possibile disabilitare gli interrupt, oppure garantire la mutua esclusione attraverso l'uso di istruzioni come la *Test and Set Lock*, capace di eseguire in modo atomico la lettura del valore di una parola di memoria (lock) e la scrittura del valore 1 nella stessa parola. Quale di queste due tecniche è appropriata in un sistema monoprocesso e quale in un sistema multiprocesso.

c) Perché per permettere ai processi utente di eseguire in modo mutualmente esclusivo delle sezioni critiche é preferibile mettere a disposizione i semafori con le operazioni *up* e *down* anziché semplici variabili di *lock* con le procedure *enter_section* ed *exit_section*?

Risposta:

a) Sì, é ancora possibile: la *enter_section* si modifica come segue:

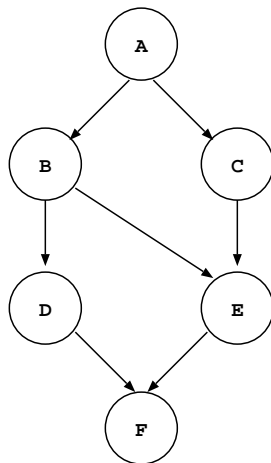
```
enter_section: move register, #1
               l1: swap register, lock
               cmp register, #0
               jne l1
               ret
```

mentre la *exit_section* non cambia.

- b) la tecnica della disabilitazione degli interrupt funziona solo in un sistema monoprocesso: infatti in un sistema multiprocesso anche se uno dei processori disabilita gli interrupt, gli altri processori possono ancora accettare interrupt, e quindi eseguire up/down sullo stesso semaforo. La tecnica che fa uso di TSL permette di assicurare l'esecuzione in mutua esclusione anche fra processori diversi: quindi é appropriata per un sistema multiprocesso (e naturalmente funziona anche se vi é un solo processore).
- c) perché la soluzione basata sull'uso di variabili di lock presenta il problema dell'attesa attiva (busy waiting) che é accettabile solo se le sezioni critiche sono *brevi* (come nel caso del codice di *up* e *down*).

Esercizio 7

Fare un esempio di grafo di precedenza realizzabile con il costrutto per l'attivazione di processi paralleli *fork-join* ma non con *cobegin-coend*. Scrivere lo pseudo codice che usa il costrutto *fork-join* e realizza il grafo. Spiegare perché il grafo non si riesce a realizzare con *cobegin-coend*.



```
P1 := fork C
B
P2 := fork D
join P1
E
join P2
F
```

Il grafo non si può realizzare con *cobegin-coend* perché affinché ciò sia possibile é necessario che i cammini che si diramano dallo stesso nodo si ricongiungano in uno stesso nodo. Nell'esempio il cammino XYZ non soddisfa questo requisito CONTROLLARE !!!!!!!!!!!!!!!

SECONDA PARTE

Esercizio 8

- a) Che differenza c'è fra uno stato di deadlock e uno stato non sicuro?
- b) Perché l'algoritmo del banchiere e quello di rilevazione del deadlock sono molto simili fra loro?
- c) Supponiamo di avere tre processi $P1$, $P2$ e $P3$ che condividono due tipi di risorse A e B . In un certo stato è stato applicato l'algoritmo del banchiere *separatamente per i due tipi di risorse* (ovvero, prima è stato applicato come se esistessero solo risorse di tipo A , e poi come se esistessero solo risorse di tipo B). L'algoritmo riconosce lo stato come sicuro in entrambi i casi. Si può concludere che lo stato è sicuro senza riapplicare l'algoritmo considerando i due tipi di risorse congiuntamente? Perché? (Se ciò vi facilita le cose, rispondete semplicemente con un esempio).

Risposta:

- a) Uno stato non sicuro può essere o meno uno stato di deadlock. Da uno stato non sicuro può succedere che si arrivi ad un deadlock (dipende da quali richieste per l'acquisizione di risorse verranno avanzate dai processi e in quale ordine).
Uno stato di deadlock è uno stato in cui è possibile individuare un (sotto)insieme di processi, ciascuno dei quali è bloccato in attesa di risorse che possono essere rilasciate unicamente da altri processi nello stesso (sotto)insieme.
- b) perché l'algoritmo del banchiere di fatto controlla se la situazione che si verrebbe a creare se *tutti i processi richiedessero contemporaneamente il massimo di risorse che hanno dichiarato di poter richiedere* è un deadlock (cioè controlla se nel caso peggiore tra tutte le possibili richieste future, si verificherebbe deadlock).
Quindi la struttura dei due algoritmi è identica, solo che nell'algoritmo di deadlock la matrice delle richieste corrisponde a richieste realmente avanzate dai processi, mentre nell'algoritmo del banchiere tale matrice corrisponde a richieste ipotetiche, che i processi *potrebbero* avanzare (ma non è detto che avanzeranno).
- c) lo stato è sicuro solo se esiste una sequenza sicura *in comune* ai due tipi di risorse. Quindi in generale non si può concludere che lo stato è sicuro rispetto ad entrambi i tipi di risorse se lo è per ciascun tipo di risorsa separatamente.

Esercizio 9

Un sistema dotato di memoria virtuale organizzata secondo la tecnica di segmentazione paginata, ha indirizzi logici composti da 2 bit per identificare il segmento, 2 bit per identificare la pagina (all'interno del segmento) e un offset di 11 bit. La dimensione della memoria fisica è di 32.768 parole. Ogni segmento può essere a sola lettura, oppure a lettura/esecuzione, oppure a lettura/scrittura oppure a lettura/scrittura/esecuzione. Supponiamo che in un dato istante la situazione delle tabelle delle pagine sia la seguente:

Segm.0 sola lett.		Segm.1 lett./esec.		Segm.2 lett./esec./scrit.		Segm.3 lett./scrit.	
N.Pag.	Frame	N.Pag.	Frame	N.Pag.	Frame	N.Pag.	Frame
0	9	0	non.val.	Tabella	non	0	14
1	3	1	0	presente	in	1	1
2	non val.	2	15	memoria	fisica	2	6
3	12	3	8			3	non.val.

Come si può osservare, anche le tabelle delle pagine sono contenute in memoria virtuale e quindi possono non essere presenti in memoria fisica. Per ognuno dei seguenti accessi alla memoria virtuale (indicati con

tipo di accesso, indirizzo logico) si dica quale indirizzo fisico viene calcolato e quando si verifica fault. In caso di fault si indichi anche il tipo di fault. (NB: l'indirizzo logico e' dato da una coppia (**numero di segmento, offset**)) a) prelievo dati (0,2049), b) prelievo dati (3,8191), c) memorizza dati (0, 2052), d) memorizza dati (3, 2050), e) salta all'istruzione (1, 6244), f) prelievo dati (2,5),

Risposta: Poiché ogni segmento é paginato, per poter effettuare la traduzione occorre ricavare dall'offset $il.offset_s$ relativo al segmento (seconda componente dell'indirizzo logico $il = (ns, offset_s)$) il numero di pagina np e l'offset relativo alla pagina $offset_p$: questi si ottengono effettuando la divisione di $il.offset_s$ per la dimensione della pagine ($2^{11} = 2048$). Il quoziente della divisione corrisponde al numero di pagina, mentre il resto corrisponde all'offset. L'indirizzo fisico a questo punto si ottiene andando a cercare il numero di frame nf corrispondente alla pagina np nella tabella delle pagine del segmento $il.ns$: se la pagina é marcata come non presente in memoria si verificherá un page fault, altrimenti l'indirizzo fisico verrà calcolato come segue: $if = nf * 2048 + offset_p$. Poiché il testo specifica che anche le tabelle delle pagine sono mantenute in memoria virtuale, anch'esse possono non essere presenti in memoria: in questo caso si ha page fault nel momento in cui si tenta di effettuare la traduzione dell'indirizzo. La traduzione potrà avvenire solo dopo che la tabella delle pagine é stata caricata in memoria centrale.

Naturalmente, prima ancora di effettuare la traduzione occorre verificare che il tipo di operazione richiesta sia ammessa dal segmento in questione: in caso non sia cosí si verificherá un fault di protezione e il sistema terminerà il processo che ha richiesto l'operazione.

- a) lettura OK per segm. 0, $np = 1, offset_p = 1, if = 3 * 2048 + 1 = 6145$
- b) lettura OK per segm. 3, $np = 3, offset_p = 2047$, Page Fault
- c) scrittura non permessa per segm. 0, Fault di protezione
- d) scrittura OK per segm 3, $np = 1, offset_p = 2, if = 1 * 2048 + 2 = 2050$
- e) fetch ok per segm. 1 (permesso di esec.), $np = 3, offset100, if = 8 * 2048 + 100 = 16484$
- f) lettura OK per segm. 2, $np = 0, offset_p = 5$, Page Fault per mancanza della tabella delle pagine del segmento 2

Esercizio 10

Calcolare per la stringa di riferimenti alle pagine di un processo riportata sotto, il numero di page fault che si verificano in un sistema il cui numero di frame in memoria centrale é F e che utilizza LRU come algoritmo di rimpiazzamento delle pagine. Il calcolo deve essere effettuato per i seguenti valori di F : 1, 2, 3, 4, 5 (Nota: é preferibile utilizzare la caratteristica di LRU di appartenere alla classe degli algoritmi a stack per calcolare in una sola *passata* il numero di page fault relativi a tutti i valori di F richiesti). Stringa di riferimenti alle pagine: 5, 3, 2, 1, 2, 5, 1, 2, 4, 3, 2, 1

Risposta:

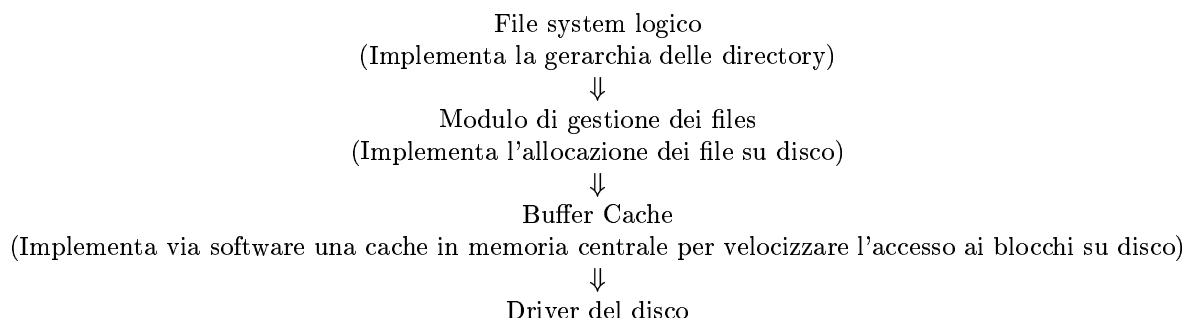
Rif.	5	3	2	1	2	5	1	2	4	3	2	1
Stack	5	3	2	1	2	5	1	2	4	3	2	1
		5	3	2	1	2	5	1	2	4	3	2
			5	3	3	1	2	5	1	2	4	3
				5	5	3	3	3	5	1	1	4
									3	5	5	5
Dist.	∞	∞	∞	∞	1	3	2	2	∞	4	2	3

$$C[1] = 1, C[2] = 3, C[3] = 2, C[4] = 1, C[\geq 5] = 5$$

N. Frame	1	2	3	4	≥ 5
N. Page Fault	12	11	8	6	5

Esercizio 11

Cosa accade all'interno del sistema operativo quando un processo chiede di eseguire la system call *read* (da file)? Indicare i parametri passati alla system call (potete inventarvi la sintassi ma dovete spiegare il significato di ciascun parametro e del valore restituito dalla suystem call), e spiegare i passaggi dal momento in cui viene sottoposta la richiesta al momento in cui il processo ha a disposizione i dati richiesti. Nel fornire la spiegazione, indicare chiaramente quali "strati" della parte di sistema operativo che implementa il file system sono coinvolti in ogni passaggio (fare riferimento all'organizzazione in strati riportata qui di seguito):



Esercizio 12

Le testine di un disco sono posizionate al cilindro 20. Il tempo necessario per spostare le testine di un cilindro é di 6 *msec*. Se la lista delle richieste pendenti contiene richieste per i seguenti cilindri: 10, 22, 20, 2, 40, 6, 38, calcolare il tempo totale di seek necessario a servire tutte le richieste per ciascuno dei seguenti algoritmi: a) FIFO, b) Shortest Seek Time First, e c) Ascensore (supponendo che la direzione corrente sia *su*)

Risposta:

Esercizio 13 A cosa serve la tabella dei file aperti?

Risposta: La tabella dei file aperti comprende una entry per ciascun file aperto, contenente le informazioni necessarie a reperire i blocchi del file su disco (mapping tra blocchi logici di file e blocchi fisici del disco): serve a velocizzare le operazioni di read e write successive (che non dovranno piú ripercorrere il percorso attraverso le directory del pathname per trovare le informazioni necessarie a localizzare il file su disco).

Esercizio 14 Indicare quali delle seguenti affermazioni sono vere e quali sono false, spiegando il perché:

- a) La paginazione può é utile solo se si vuole realizzare la memoria virtuale (non serve se abbiamo tanta memoria centrale a disposizione)
- b) La memoria virtuale é utile anche se i processi hanno una immagine in memoria che é sempre di dimensione piú piccola della memoria fisica (cioé anche se il numero di bit nell'indirizzo logico é inferiore al numero di bit nell'indirizzo fisico).
- c) Un sistema paginato non può avere frammentazione esterna
- d) Il problema della frammentazione interna/esterna si ha solo nella gestione della memoria centrale e non nella gestione dello spazio su disco

Risposta:

- a) Falso: é utile anche se non si vuole realizzare la memoria virtuale perché facilita l'allocazione dello spazio ai processi.

- b)** Vero: in un sistema multiprogrammato (o comunque multitasking), permette di mantenere piú processi contemporaneamente in memoria
- c)** Vero: la frammentazione esterna é un problema legato all'allocazione contigua
- d)** Falso: i problemi di allocazione dello spazio in memoria centrale o su disco sono simili. Su disco si ha frammentazione interna ogni volta che la dimensione di un file non é un multiplo della dimensione del blocco. Si ha frammentazione esterna se i file sono allocati in blocchi contigui.