

Corso di Laurea in Informatica
Corso di Sistemi Operativi I
Compitino del 10.1.2001

Prima parte

Esercizio 1 (punti 4/ 3)

Un processo durante la sua esecuzione alterna fasi di utilizzo della CPU (dette CPU-burst) a fasi di accesso ai dispositivi di I/O (dette IO-burst). Quali caratteristiche hanno queste fasi nei processi I/O-bound? E in quelli CPU-bound?

Qual è il vantaggio della multiprogrammazione? Spiegare utilizzando il modello di esecuzione dei processi descritto sopra.

Soluzione: vedere il libro di testo.

Esercizio 2 (punti 4/3)

Un hacker ha deciso di mettere in difficoltà gli utenti di un sistema time sharing facendo eseguire un programma che opera come segue: quando diventa running il programma disabilita gli interrupt per un lungo periodo. Durante questo periodo modifica le zone di memoria appartenenti ad altri processi (scelti a caso) impostando opportunamente i registri base e limite della CPU (quelli usati per ottenere l'indirizzo fisico come base + indirizzo logico). Infine, colloquiando direttamente con i controller delle tastiere di alcuni terminali collegati al sistema, le blocca, impedendo agli utente che li stanno usando di lavorare.

Supponendo che il programma dell'hacker sia eseguito in modalità utente, è possibile che questi riesca nel suo intento? Perché?

Soluzione:

Non è possibile perchè tutte le istruzioni utilizzate dal programma dell'hacker sono privilegiate, e quindi non potranno essere eseguite in modalità utente (verrà causata una trap al sistema operativo che bloccherà il processo).

Esercizio 3 (punti 3/ 2)

In un dato istante in un sistema operativo multiprogrammato vi sono sei processi: P1 e P3 sono ready, P5 è running, P2 è waiting, in attesa della fine di una lettura da disco, P4 è waiting perchè ha eseguito una system call *sleep(2)* ed è in attesa che sia trascorso il tempo di sleep, P6 è waiting perchè ha eseguito una *down(sem)* quando il valore associato a *sem* era 0.

Quali cambiamenti di stato si verificheranno come conseguenza dei seguenti eventi ? (ciascun evento va considerato in isolamento, non in sequenza a quelli che lo precedono):

1. device interrupt: il disco ha concluso l'operazione richiesta da P2,
2. P5 esegue una *up(sem)*,
3. timer interrupt: è scaduto il quanto di P5, inoltre è trascorso il tempo di *sleep* di P4,
4. P5 esegue una *down(mutex)*, quando il valore associato a *mutex* era 1.
5. P5 esegue una *down(mutex)*, quando il valore associato a *mutex* era 0.

Soluzione:

1. device interrupt: il disco ha concluso l'operazione richiesta da P2, quindi P2 diventa ready, e in determinate situazioni lo scheduler potrebbe decidere di farlo diventare immediatamente running al posto di P5 (per esempio nel caso P2 abbia una priorità maggiore di P1, P3 e P5);
2. P5 esegue una *up(sem)*, P6 viene svegliato, quindi diventa ready: come nel punto precedente, P6 potrebbe diventare immediatamente running;
3. timer interrupt: è scaduto il quanto di P5, inoltre è trascorso il tempo di *sleep* di P4, P4 diventa ready, e a P5 viene tolta la CPU: lo scheduler deve scegliere un processo fra P1, P3 e P4 da far diventare running mentre P5 diventa ready. e renderlo immediatamente running.
4. P5 esegue una *down(mutex)*, quando il valore associato a *mutex* era 1. P5 continua ad essere running: l'unico effetto della down in questo caso è di portare il valore di *mutex* a 0.
5. P5 esegue una *down(mutex)*, quando il valore associato a *mutex* era 0. In questo caso invece P5 viene posto in stato waiting: lo scheduler deve scegliere tra i processi ready (P1 e P3) chi rendere running.

Esercizio 4 (punti 7/5)

Si consideri la seguente soluzione al problema dei lettori/scrittori senza starvation. La specifica richiede che (1) ogni scrittura avvenga in mutua esclusione rispetto a qualsiasi altra operazione, (2) le letture possono avvenire in parallelo, ma solo se non ci sono scritture in corso, (3) se un lettore arriva mentre ci sono scrittori in attesa, non può iniziare a leggere (per evitare starvation degli scrittori), e deve quindi restare in attesa del suo turno, (4) quando uno scrittore termina il suo lavoro, deve attivare tutti gli eventuali lettori in attesa se ce ne sono, altrimenti deve attivare il prossimo scrittore in coda, (5) quando un lettore termina, se non vi sono altre letture in corso deve svegliare il primo scrittore in coda (ammesso che ce ne siano).

Uno studente di informatica ha scritto il seguente codice, che sfortunatamente non realizza correttamente la soluzione sopra specificata: quale errore ha commesso? Come si può correggere?

Le procedure Inizio/Fine-Lettura/Scrittura fanno uso dei seguenti semafori e variabili, che ovviamente dovranno essere condivise da tutti i processi, lettori e scrittori.

```
semaforo mutex initial 1;
semaforo sem_priv_lettori initial 0;
semaforo sem_priv_scrittori initial 0;
int lettori_attivi, scrittori_attivi;
int lettori_bloccati, scrittori_bloccati;
```

```

InizioLettura()
{ down (mutex);
  if ( (scrittori_attivi==0) &&
      (scrittori_bloccati==0)
    { lettori_attivi++;
      up(sem_priv_lettori);}
  else
    lettori_bloccati++;
  up(mutex)
  down(sem_priv_lettori);
}

FineLettura()
{down(mutex);
  lettori_attivi--;
  if ((lettori_attivi==0) &&
      (scrittori_bloccati>0))
    { scrittori_bloccati--;
      scrittori_attivi++;
      up(sem_priv_scrittori);}
  up(mutex);
}

InizioScrittura()
{ down (mutex);
  if ( (scrittori_attivi==0) &&
      (lettori_attivi==0)
    { scrittori_attivi++;
      up(sem_priv_scrittori);}
  else
    scrittori_bloccati++;
  up(mutex)
  down(sem_priv_scrittori);
}

FineScrittura()
{down(mutex);
  scrittori_attivi--;
  if (lettori_bloccati > 0)
    { lettori_bloccati--;
      lettori_attivi++;
      up(sem_priv_lettori); }
  else if (scrittori_bloccati>0)
    { scrittori_bloccati--;
      scrittori_attivi++;
      up(sem_priv_scrittori);}
  up(mutex);
}

```

Soluzione: L'errore è nella FineScrittura, infatti qui dovrebbero essere svegliati *tutti* i lettori bloccati (se ce ne sono), non solo 1 (quindi dentro l'if ci dovrebbe essere un while (lettori_bloccati > 0)). Vedere la versione corretta del codice negli appunti integrativi su programmazione concorrente pubblicati sulla pagina del corso.

Esercizio 5 (punti 5/4)

Calcolare il tempo medio di attesa per il seguente insieme di processi utilizzando tre diversi algoritmi di scheduling: Shortest Job First, Shortest Remaining Time First, Round Robin con quanto=3 (disegnare per ciascun algoritmo il diagramma di Gantt e indicare le operazioni fatte per calcolare il tempo medio di attesa). Quali di questi algoritmi sono con prelazione, quali no? Indicare quanti context switch avvengono in ciascuno dei tre casi.

Processo	Istante Arrivo	Lunghezza CPU burst
P1	0	8
P2	2	2
P3	2	4
P4	4	2
P5	5	3

Soluzione:

SJF: Questo algoritmo non è con prelazione

Gantt: P1(0,8), P2(8,10), P4(10,12), P5(12,15), P3(15,19)

$$\bar{w} = (0 + (8 - 2) + (15 - 2) + (10 - 4) + (12 - 5)) / 5 = 32 / 5 = 6.4$$

Ci sono stati 4 context switch.

SRTF: Questo algoritmo è con prelazione

Gantt: P1(0,2), P2(2,4), P4(4,6), P5(6,9), P3(9,13), P1(13,19)

$$\bar{w} = ((19 - 8) + (4 - 2 - 2) + (13 - 2 - 4) + (6 - 4 - 2) + (9 - 5 - 3))/5 = 19/5 = 3.8$$

Ci sono stati 5 context switch.

RR, q=3: Questo algoritmo è con prelazione

Gantt: P1(0,3), P2(3,5), P3(5,8), P1(8,11), P4(11,13), P5(13,16), P3(16,17), P1(17,19)

Ci sono stati 7 context switch.

$$\bar{w} = ((19 - 8) + (5 - 2 - 2) + (17 - 2 - 4) + (13 - 4 - 2) + (16 - 5 - 3))/5 = 38/5 = 7.6$$

Tempo	0	2	3	4	5	8	11	13	16	17	19
Ready		P2(2)	P3(4)	P3(4)	P1(5)	P4(2)	P5(3)	P3(1)	P1(2)	-	-
Queue		P3(4)	P1(5)	P1(5)	P4(2)	P5(3)	P3(1)	P1(2)			
				P4(2)	P5(3)	P3(1)	P1(2)				
Running	P1	P1	P2	P2	P3	P1	P4	P5	P3	P1	-
Esce					P2			P4	P5	P3	P1

Esercizio 6 (punti 5) Solo per chi fa il primo compito

Si descriva ciò che accade al momento del richiamo di una system call. Spiegare in che modo viene coinvolto il gestore degli interrupt. In che senso il sistema operativo è *guidato dagli interrupt*?

Soluzione: vedere il libro di testo.

Esercizio 7 (punti 5) Solo per chi fa il primo compito

Il linguaggio macchina di un certo processore, dispone di una istruzione atomica *swap a1, a2*, che può scambiare il contenuto di due celle di memoria (di indirizzo a1, e a2 rispettivamente). Come è possibile utilizzare questa istruzione per risolvere il problema dell'accesso in mutua esclusione ad una sezione critica? (è ammesso il busy waiting).

Suggerimento: confrontare la *swap* con la *test and set lock*.

Soluzione:

Si usa una variabile *lock*: interpretiamo *lock* = 1 come "semaforo rosso" mentre *lock* = 0 significa "via libera".

EntraNellaSC:

a=1;

while (a==1)

swap(a, lock);

EsciDallaSC:

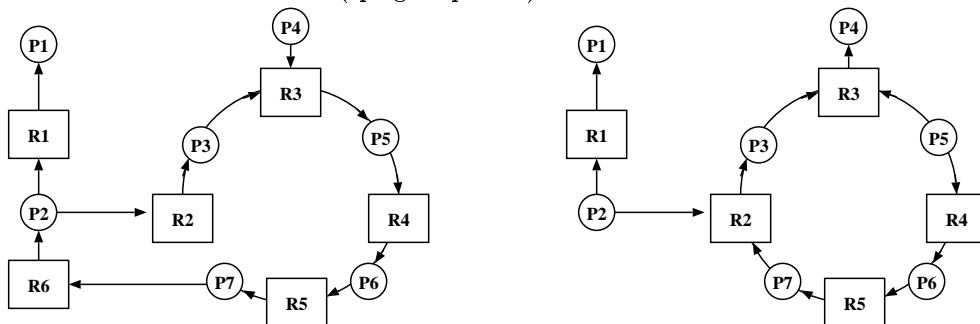
lock=0

Quando *a* dopo la swap(a,lock) vale 0, significa che *lock* valeva 0 prima della swap ed è stato impostato a 1 dalla swap (atomicamente, insieme all'assegnazione di *lock* ad *a*, quindi si può entrare in sezione critica.

Seconda parte

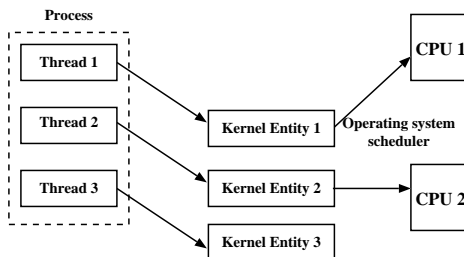
Esercizio 8 (punti 6/4)

a) Si considerino i seguenti due grafi di allocazione delle risorse. Quale dei due (destra o sinistra) corrisponde ad una situazione di deadlock? (spiegare perchè).



b) Si consideri un sistema in cui ad ogni risorsa è assegnato un numero progressivo (uno diverso per ciascuna risorsa) e in cui i processi possono chiedere di acquisire la risorsa numero i solo se non possiedono nessuna risorsa di numero maggiore di i . In questo sistema può verificarsi deadlock? Perchè?

Esercizio 9 (punti 5/3)



La figura illustra un package di thread a livello utente o a livello kernel?

Che differenza c'è tra una system call per creare un processo e una system call per creare un thread (supponiamo che la system call per creare il processo crei automaticamente anche il primo thread di quel processo)? Indicare quali operazioni vengono effettuate dal sistema operativo nei due casi.

Esercizio 10 (punti 5/3)

Un sistema operativo usa l'allocazione contigua dei processi in memoria, secondo la tecnica delle partizioni variabili. Supponiamo che inizialmente vengano attivati i processi P1, di dimensione 8Mbyte, P2, di dimensione 8Mbyte, e P3 di dimensione 2Mbyte. In seguito P2 termina e viene chiesta l'attivazione di P4 di dimensione 5 Mbyte e P5 di dimensione 7 Mbyte. Se in tutto sono disponibili 24 Mbyte di memoria centrale per i processi utente, dove verranno allocati in memoria i cinque processi se viene usato l'algoritmo First Fit per la scelta dello spazio libero da occupare? E se viene usato l'algoritmo Best Fit?

In questo tipo di allocazione della memoria quale tipo di frammentazione si può verificare (interna o esterna)?

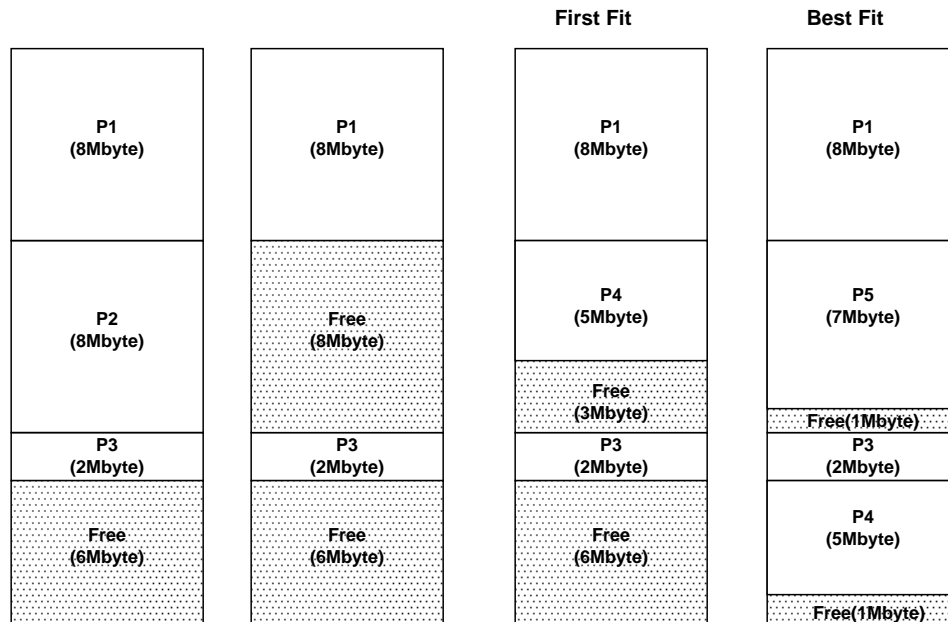
Soluzione:

Inizialmente i primi 8Mbyte liberi sono assegnati a P1, i successivi 8Mbyte a P2, e i restanti 2Mbyte vengono assegnati a P3. Rimarranno $24 - 8 - 8 - 2 = 6\text{Mbyte}$ liberi dopo l'ultimo processo.

Quando termina P2 si libera uno spazio da 8Mbyte. Ora vengono creati due nuovi processi per i quali bisognerà scegliere una allocazione:

First Fit Il First Fit alloca P4 nel primo spazio libero dopo P1, di dimensione 8Mbyte. In questo modo rimangono due spazi liberi: uno di dimensione 3 Mbyte, l'altro di 6Mbyte. Di conseguenza P5 non potrà essere più allocato, a meno che si faccia una operazione di compattazione della memoria per creare uno spazio libero contiguo di $3+6 = 9$ Mbyte.

Best Fit L'algoritmo Best Fit cerca lo spazio che meglio combacia con la dimensione del processo da allocare, pertanto allocherà P4 nello spazio da 6Mbyte, e successivamente P5 nello spazio da 8 Mbyte lasciato da P2. Alla fine rimarranno due spazi liberi, entrambi da 1Mbyte.



Il tipo di frammentazione che si può verificare con questo schema di allocazione è la **frammentazione esterna**.

Esercizio 11 (punti 6/4)

Un processo in esecuzione genera i seguenti riferimenti ad indirizzi logici (in parte per il fetch di istruzioni, in parte per accesso a dati):

10260, 4140, 8200, 2070, 10264, 4140, 12300, 10265, 14400, 12300, 10266, 12304, 10268, 4140

Sapendo che il sistema operativo implementa la memoria virtuale paginata, con pagine da 2Kb, si trasformi la sequenza di indirizzi in una sequenza di numeri di pagine logiche, quindi si calcoli il numero di page fault applicando gli algoritmi di rimpiazzamento ottimo, FIFO, ed LRU, supponendo che al processo siano stati assegnati 4 frame.

Nota: La tabellina del 2048 è 2048, 4096, 6144, 8192, 10240, 12288, 14336,...

Soluzione:

La sequenza di indirizzi logici:

10260, 4140, 8200, 2070, 10264, 4140, 12300, 10265, 14400, 12300, 10266, 12304, 10268, 4140

corrisponde alla sequenza di riferimenti a pagine seguente:

5, 2, 4, 1, 5, 2, 6, 5, 7, 6, 5, 6, 5, 2

(ottenuta dividendo ciascun indirizzo logico per la dimensione della pagina: 2048byte).

Applichiamo l'algoritmo FIFO, con 4 frame:

5	2	4	1	5	2	6	5	7	6	5	6	5	2
5	2	4	1	1	1	6	5	7	7	7	7	7	2
	5	2	4	4	4	1	6	5	5	5	5	5	7
		5	2	2	2	4	1	6	6	6	6	6	5
			5	5	5	2	4	1	1	1	1	1	6
*	*	*	*			*	*	*					*

Il numero totale di Page Fault risultante è 8.

Applichiamo l'algoritmo LRU, con 4 frame:

5	2	4	1	5	2	6	5	7	6	5	6	5	2
5	2	4	1	5	2	6	5	7	6	5	6	5	2
	5	2	4	1	5	2	6	5	7	6	5	6	5
		5	2	4	1	5	2	6	5	7	7	7	6
			5	2	4	1	1	2	2	2	2	2	7
*	*	*	*			*		*					*

Il numero totale di Page Fault risultante è 6.

Esercizio 12 (punti 4/2)

Un file è costituito da quattro blocchi logici, allocati rispettivamente nei blocchi fisici 15, 21, 18 e 24 del disco. Spiegare quali strutture dati sono mantenute dal sistema per riuscire ad associare i blocchi logici ai corrispondenti blocchi fisici nei seguenti tre casi: (a) allocazione a lista concatenata, con puntatori contenuti nei blocchi del file, (b) allocazione a lista concatenata con puntatori contenuti nella FAT, (c) allocazione indicizzata (i-node). Tali strutture dati sono memorizzate nel disco, in memoria o in entrambi? Quale delle tre organizzazioni è meno efficiente se il tipo di accesso al file è casuale (non sequenziale)? Perché?

Supponiamo che in questo sistema i blocchi siano di 1KB. Se un processo richiede una write che coinvolge 47 byte a partire dal 3068-esimo byte del file, quali blocchi logici (del file) e quali blocchi fisici (del disco) saranno modificati dalla system call? Supponendo che questo sia il primo accesso al file dalla sua apertura, quanti blocchi saranno coinvolti (non necessariamente tutti modificati) nell'operazione in ciascuno dei tre tipi di allocazione indicati sopra (a,b, e c) ? Spiegare perchè.

Soluzione:

Il file in questione è costituito da 4 blocchi (da 1Kb). Se il sistema usa uno schema di allocazione a lista concatenata, nella entry della directory corrispondente al file sarà memorizzato un puntatore al primo blocco del file su disco (ovvero 15), nel blocco 15 oltre ai dati della parte iniziale del file sarà memorizzato un puntatore al blocco successivo: 21, nel blocco 21 sarà memorizzato un puntatore al blocco successivo: 18, nel blocco 18 sarà memorizzato un puntatore al blocco successivo: 24, infine nel blocco 24 sarà memorizzato un puntatore nullo, ad indicare che questo è l'ultimo blocco del file. Quando il file in questione viene aperto, nella tabella dei file aperti viene mantenuto solo il puntatore al primo blocco (15).

Nel secondo tipo di organizzazione, che fa uso di una FAT (File Allocation Table), i quattro blocchi fisici conterranno solo dati. Nella directory sarà ancora memorizzato il puntatore al primo blocco, mentre i puntatori ai blocchi successivi saranno contenuti in un vettore, la FAT. Il puntatore al blocco successivo al numero 15 si trova nella FAT in corrispondenza dell'indice 15: $FAT[15] = 21$, il puntatore al successore del numero 21, si trova nel corrispondente elemento della FAT: $FAT[21] = 18$, e così via. Infine $FAT[24] = \text{NULL}$ ad indicare che non ci sono altri blocchi in questo file.

Se il sistema usa uno schema di allocazione con blocco indice, viene mantenuta una tabella di indici che elenca i blocchi fisici dove sono allocati i blocchi del file (una tabella simile alla tabella delle pagine). In

UNIX questa tabella è memorizzata nell'i-node del file, e viene caricata in memoria al momento della `open()`. Nel caso del file dell'esempio la tabella conterrà 15 nel primo elemento, 21 nel secondo, 18 nel terzo e 24 nel quarto.

La FAT è memorizzata su disco, ma viene portata (tutta o in parte) in memoria durante il funzionamento del sistema per velocizzare la ricerca dei blocchi dei file. Gli I-node devono essere su disco, ma il contenuto degli i-node dei file aperti viene mantenuto in memoria.

L'organizzazione a lista linkata (senza FAT) è la meno efficiente nel caso di accessi casuali, perchè per accedere al k-esimo blocco ci vogliono k accessi al disco.

Se i blocchi sono da 1Kb, il byte 3068 si trova nel blocco logico $3068/1024 = 2$, mentre il $3068+47-1 = 3114$ si trova nel blocco logico 3: quindi si dovranno leggere due blocchi per soddisfare la richiesta. In particolare si tratta dei blocchi fisici 18 e 24. Per accedere a tali blocchi, nell'organizzazione a lista concatenata servono 4 accessi (si deve percorrere la lista), in quella a lista concatenata con FAT, supponendo che la FAT sia già in memoria, ne bastano 2. Anche nell'organizzazione indicizzata bastano 2 accessi.

Esercizio 13 (punti 3)

Quali sono le quattro condizioni necessarie affinché si possa verificare il deadlock? Fare alcuni esempi di come si può prevenire il deadlock negando una delle quattro condizioni.

Soluzione: vedere il libro di testo.

Esercizio 14 (punti 4)

Cosa si intende per *working set* di un processo? Cosa accade se la somma delle dimensioni dei working set di tutti i processi presenti nel sistema supera la dimensione della memoria fisica disponibile? Il sistema operativo come può agire se si verifica questa eventualità?

Soluzione: vedere il libro di testo.