SISTEMI OPERATIVI, CORSI A e B - QUINTO APPELLO 2007- 20/7/2007

ESERCIZIO A-1 (4 punti)

Si considerino due thread linux A e B che scambiano messaggi attraverso un buffer condiviso a n posizioni. Ogni messaggio occupa una posizione del buffer. A tal fine A e B condividono un semaforo di mutua esclusione *mux* (inizializzato a 1), due variabili di condizione *NonPieno* e *NonVuoto* e una variabile *n_elem*, che rappresenta il numero di posizioni occupate nel buffer, e si sincronizzano con funzioni della libreria *pthreads.*
La parte significativa del codice di A e B è la seguente:

Thread A:

```
a.1)    while (true) {
a.2)       <produce  v>
a.3)       pthread_mutex_lock(&mux);
a.4)       if (n_elem == n) pthread_cond_wait(&NonPieno,&mux);
a.5)       <deposita v nel buffer>
a.6)       n_elem++;
a.7)       if (n_elem == 1) pthread_cond_signal(&NonVuoto);
a.8)       pthread_mutex_unlock(&mux);
        }
```

Thread B:

```
b.1)    while (true) {
b.2)       pthread_mutex_lock(&mux);
b.3)       if (n_elem == 0) pthread_cond_wait(&NonVuoto,&mux);
b.4)       <preleva un valore v dal buffer>
b.5)       n_elem--;
b.6)       if (n_elem == n-1) pthread_cond_signal(&NonPieno);
b.7)       pthread_mutex_unlock(&mux);
b.8)       <consuma v>
        }
```

A un certo istante si ha *n_elem = 0, mux= 1*, e i thread A e B devono eseguire le istruzioni a.1 e b.1, rispettivamente. I due thread avanzano eseguendo la seguente sequenza di operazioni:

1) B esegue b.1, b.2, b.3
2) A esegue a.1, a.2, a.3
3) A esegue a.4
4) A esegue a.5, a.6, a.7
5) A esegue a.8, a.1, a.2
6) B esegue b.4, b.5, b.6
7) A esegue a.3
8) B esegue b.7

Si chiede lo stato (non sospeso, sospeso su *mux*, sospeso su *NonPieno*, sospeso su *NonVuoto*) dei thread A e B subito dopo 1), subito dopo 2), subito dopo 3), subito dopo 4), subito dopo 5) e subito dopo 6).

SOLUZIONE

	stato A	Stato B
1	non sospeso	sospeso su NonVuoto
2	non sospeso	sospeso su NonVuoto
3	non sospeso	sospeso su NonVuoto
4	non sospeso	non sospeso
5	non sospeso	non sospeso
6	non sospeso	non sospeso
7	sospeso su mux	non sospeso
8	non sospeso	non sospeso

SISTEMI OPERATIVI, CORSI A e B - QUINTO APPELLO 2007- 20/7/2007

ESERCIZIO A.2 (4 punti)

Un sistema che gestisce il processore combinando le politiche a priorità e RoundRobin con la tecnica delle code multiple (una coda FIFO per ogni valore di priorità; i processi pronti di uguale priorità sono inseriti in una stessa coda; il processore viene assegnato al processo che occupa la prima posizione nella coda non vuota di massima priorità; ai processi pronti di uguale priorità si applica la politica Round Robin).
Il quanto di tempo è di *10 msec*.
La politica prevede il prerilascio, che avviene immediatamente dopo l'evento che lo provoca, senza attendere l'esaurimento del quanto di tempo corrente. Quando un processo va in esecuzione gli viene assegnato un intero quanto di tempo, indipendentemente dal tempo consumato nel precedente turno di esecuzione.
Al tempo T, nel sistema sono presenti i seguenti processi:

- Processo A, con priorità 2, che al tempo T è in stato di attesa sul semaforo Sem1;
- Processo B, con priorità 3, che al tempo T è in stato di attesa sul semaforo Sem1;
- Processo C, con priorità 1, che al tempo T è in stato di pronto;
- Processo D, con priorità 2, che al tempo T passa in stato di esecuzione.
- Processo E, con priorità 1, che al tempo T è in stato di pronto;

Al tempo T la coda corrispondente alla priorità 1 contiene: C->E (C è in testa), e tutte le altre code sono vuote. La coda del semaforo Sem1 contiene invece A->B (A è in testa).

Si chiede quale è il processo in esecuzione e la composizione delle 3 code al tempo T+30
se si verificano le seguenti sequenze di eventi (**da considerare in alternativa**):

1. Al tempo T+8 il processo in esecuzione esegue una *signal* sul semaforo Sem1; al tempo T+12 il processo in esecuzione esegue una *wait* sul semaforo Sem1; al tempo T+14 termina il processo in esecuzione, al tempo T+26 il processo in esecuzione esegue una *wait* sul semaforo Sem1.
2. Al tempo T+4 il processo in esecuzione esegue una *wait* sul semaforo Sem1; al tempo T+16 il processo in esecuzione esegue una *signal* sul semaforo Sem1; al tempo T+20 il processo in esecuzione esegue una *wait* sul semaforo Sem1; al tempo T+27 termina il processo in esecuzione

SOLUZIONE

Sequenza 1

	In esec.	Coda priorità 1	Coda priorità 2	Coda priorità 3	Sem1: <valore,coda>
T	D	C->E			<0,A->B>
T+8	D	C->E	A		<0,B>
T+10	A	C->E	D		<0,B>
T+12	D	C->E			<0,B->A>
T+14	C	E			<0,B->A>
T+24	E	C			<0,B->A>
T+26	C				<0,B->A->E>
T+30	C				<0,B->A->E>

Sequenza 2

	In esec.	Coda priorità 1	Coda priorità 2	Coda priorità 3	Sem1: <valore,coda>
T	D	C->E			<0,A->B>
T+4	C	E			<0,A->B->D>
T+14	E	C			<0,A->B->D>
T+16	A	C->E			<0,B->D>
T+20	C	E			<0,B->D->A>
T+27	E				<0,B->D->A>
T+30	E				<0,B->D->A>

SISTEMI OPERATIVI, CORSI A e B - QUINTO APPELLO 2007- 20/7/2007

ESERCIZIO A.3 (3 punti)

In un sistema con risorse R1, R2, R3, R4 e R5, tutte con molteplicità 2, sono presenti i processi P1, P2 e P3 che inizialmente non possiedono risorse e successivamente avanzano senza interagire reciprocamente e alternandosi nello stato di esecuzione con velocità arbitrarie.
Nel corso della propria esistenza, ciascun processo esegue una propria sequenza di richieste, che si intercalano in modo arbitrario con quelle degli altri processi. Dopo aver ottenuto e utilizzato le risorse che richiede, ogni processo termina rilasciando tutte le risorse ottenute.
Si considerino, in alternativa, le sequenze di richieste sotto riportate:

Sequenza 1)

Processo	Prima richiesta	Seconda richiesta	Terza richiesta	Quarta richiesta	Terminazione
P1	2 istanze di R1	1 istanza di R3	1 istanza di R4	1 istanza di R5	Rilascia
P2	2 istanze di R2	1 istanza di R3	2 istanze di R4	1 istanza di R3	Rilascia
P3	2 istanze di R1	1 istanza di R3	1 istanza di R4	2 istanze di R5	Rilascia

Sequenza 2)

Processo	Prima richiesta	Seconda richiesta	Terza richiesta	Quarta richiesta	Terminazione
P1	2 istanze di R1	1 istanza di R3	1 istanza di R4	1 istanza di R5	Rilascia
P2	2 istanze di R2	1 istanza di R3	2 istanze di R4	1 istanza di R5	Rilascia
P3	2 istanze di R1	1 istanza di R2	1 istanza di R3	2 istanze di R5	Rilascia

Sequenza 3)

Processo	Prima richiesta	Seconda richiesta	Terza richiesta	Quarta richiesta	Terminazione
P1	1 istanza di R1	1 istanza di R3	1 istanza di R4	1 istanza di R5	Rilascia
P2	2 istanze di R2	1 istanza di R3	2 istanze di R4	1 istanza di R5	Rilascia
P3	1 istanza di R1	1 istanza di R4	1 istanza di R5	2 istanze di R2	Rilascia

Per ogni sequenza, si chiede se i processi evitano la possibilità di stallo e la motivazione della risposta.

SOLUZIONE

Sequenza 1) La sequenza evita la possibilità di stallo? NO

Motivazione della risposta: La quarta richiesta di P2 viola il vincolo di assegnazione ordinata.

Sequenza 2) La sequenza evita la possibilità di stallo? SI

Motivazione della risposta: Tutti i processi rispettano il vincolo di assegnazione ordinata.

Sequenza 3) La sequenza evita la possibilità di stallo? NO

Motivazione della risposta: La quarta richiesta di P3 viola il vincolo di assegnazione ordinata.

SISTEMI OPERATIVI, CORSI A e B - QUINTO APPELLO 2007- 20/7/2007

ESERCIZIO A.4 (2 punti)

In un sistema UNIX, vengono eseguite in sequenza le seguenti operazioni:

a) il processo P esegue con sucesso una chiamata *pipe,* che restituisce i descrttori *fd[0]* e *fd[1];*
b) il processo P esegue con successo una *fork,* generando il processo P1 che immediatamente esegue con successo una *exec;*
c) il processo P esegue con successo una *fork,* generando il processo P2 che immediatamente esegue con successo una *exec;*
d) il processo Q, padre del processo P, esegue con sucesso una chiamata *pipe,* che restituisce i descrttori *fd'[0]* e *fd'[1]];*
e) il processo Q esegue l'operazione *write(fd'[1], 'buona sera', 10)*
f) il processo P1 esegue l'operazione *write(fd[1], 'buonanotte', 10)*
g) il processo P2 esegue l'operazione *read(fd[0], &buffer, 10).*

Qual è il contenuto di *buffer* dopo l'operazione g)?

SOLUZIONE

Dopo l'operazione g), il contenuto di *buffer* è *'buonanotte'*

ESERCIZIO A.5 (2 punti)

In un sistema dove i processi operano in ambiente locale e dispongono di una libreria per la realizzazione dei thread a livello utente, sono presenti il processo P1 con priorità 10, il processo P2 con priorità 5 e il processo P3 con priorità 10. Lo scheduling dei processi avviene con una politica a priorità (va in esecuzione il processo pronto con il massimo valore di priorità) e prevede il prerilascio. Per ogni processo, i thread alternano tra lo stato di esecuzione e quello di pronto. Il thread che passa dallo stato di esecuzione a quello di pronto viene inserito nell'ultima posizione della coda dei thread pronti. Lo scheduling dei thread avviene con politica FIFO, senza prerilascio.

Al tempo *T* è in esecuzione il processo P1, il processo P2 è pronto e il processo P3 è sospeso in seguito all'esecuzione della primitiva *receive(&mess, P1)*. Inoltre sono stati creati seguenti thread:

Processo P1: thread T11, in stato di esecuzione;

Processo P2 : thread T21, T22, tutti in stato di pronto con il seguente ordinamento della coda: <primo> → T21 → T22 ;

Processo P3 : thread T31, T32, T33, con il seguente ordinamento della coda: <primo> → T31 → T32 → T32.

Si chiede qual è il thread in esecuzione dopo ciascuno degli eventi della sequenza riportata in tabella.

SOLUZIONE

	Sequenza di eventi	Thread in esecuzione dopo l'evento
a)	il thread in esecuzione esegue la primitiva (non bloccante) *send(&messaggio, P1)*	T11
b)	il thread in esecuzione esegue la primitiva (bloccante) *receive(&risposta, P1)*	T11 (riceve il messaggio inviato con a) dallo stes thread)
c)	Il thread in esecuzione esegue l'operazione *thread_yield*	T11 (unico thread di P1)
d)	il thread in esecuzione commette un errore di indirizzamento non recuperabile	T21 (P1 soppresso, P3 sospeso)
e)	il thread in esecuzione esegue la primitiva (non bloccante) *send(&risp, P1)*	T21 (operazione non bloccante)

SISTEMI OPERATIVI, CORSI A e B - QUINTO APPELLO 2007- 20/7/2007

ESERCIZIO B.1 (4 punti)

In un sistema che gestisce la memoria con paginazione a domanda, le pagine logiche e i blocchi fisici hanno una lunghezza di 2^8= 256 byte e l'ampiezza della memoria fisica è di 2^{16} blocchi. Gli indirizzi logici hanno una lunghezza di 16 bit, e pertanto ogni processo dispone di una memoria virtuale di 2^8 pagine.
Per la gestione della memoria si utilizzano tabelle delle pagine a 2 livelli, tutte di uguale lunghezza, pari a 2^4= 16 elementi. La tabella di primo livello è caricata permanentemente nel blocco 0 della memoria principale, mentre le tabelle di secondo livello sono caricate a domanda. Quando risiede in memoria, ogni tabella di primo o di secondo livello occupa 1 blocco.
Nella tabella di primo livello, il generico elemento è l'indice del blocco fisico oppure 0, a seconda che la corrispondente tabella di secondo livello risieda o non risieda in memoria. In ogni tabella di secondo livello, il generico elemento è l'indice del blocco fisico oppure 0, a seconda che la corrispondente pagina logica risieda o non risieda in memoria.
Al tempo t è in esecuzione il processo P e i contenuti della sua tabella delle pagine di primo livello e di alcune delle sue tabelle delle pagine di secondo livello sono mostrati in figura. Nella memoria fisica sono disponibili alcuni blocchi, che al verificarsi di *page faults* sono utilizzabili per caricare tabelle di secondo livello o pagine logiche. Questi blocchi sono ordinati nella coda <PrimoBlocco> → 40 → 41 → 42 → 43 → …….. e, in caso di *page fault*, sono utilizzati secondo questo ordinamento.

	Blocco
0	0
1	01
2	0
3	02
4	0
5	0
6	04
7	0
8	0
9	07
A	0
B	0
C	0B
D	0
E	0
F	0

Tab. 1° livello

	Blocco
0	10
1	0
2	0
3	0
4	11
5	0
6	0
7	0
8	12
9	0
A	0
B	14
C	0
D	0
E	0
F	0

Tab 2 liv. indice 0 Blocco ??

	Blocco
0	0
1	0
2	13
3	0
4	0
5	15
6	0
7	0
8	0
9	16
A	0
B	0
C	0
D	0
E	0
F	0

Tab 2 liv. indice 3 Blocco 02

	Blocco
0	30
1	0
2	0
3	18
4	0
5	0
6	0
7	17
8	0
9	0
A	0
B	19
C	0
D	0
E	0
F	0

Tab 2 liv. indice 4 Blocco ??

	Blocco
0	0
1	1A
2	0
3	0
4	0
5	0
6	1B
7	1C
8	0
9	0
A	0
B	1E
C	0
D	0
E	0
F	0

Tab 2 liv. indice 9 Blocco 07

	Blocco
0	1D
1	0
2	0
3	0
4	20
5	0
6	0
7	0
8	21
9	0
A	0
B	0
C	22
D	0
E	0
F	0

Tab 2 liv. indice A Blocco ??

	Blocco
0	0
1	0
2	23
3	0
4	0
5	0
6	0
7	24
8	0
9	0
A	0
B	0
C	0
D	0
E	0
F	0

Tab 2 liv. indice D Blocco ??

A partire dal tempo t il processo P accede alla memoria con i seguenti indirizzi (gli indirizzi e gli indici di blocco sono espressi con notazione esadecimale; si ricorda che una cifra decimale corrisponde a 4 bit della rappresentazione binaria):

1. indirizzo 3B4D
2. indirizzo A813
3. indirizzo 9B0C
4. indirizzo DA59

Per ciascun accesso alla memoria, si chiede:

- se l'accesso alla tabella di secondo livello determina page fault
- il blocco fisico che contiene (dopo l'eventuale caricamento in memoria) la tabella di secondo livello
- se l'accesso alla pagina riferita determina page fault
- blocco fisico che contiene (dopo l'eventuale caricamento in memoria) la pagina riferita.

SOLUZIONE

1) Indirizzo 3B4D

l'accesso alla tabella di 2° livello determina *page fault?*	NO
blocco fisico che contiene la tabella di secondo livello	02
l'accesso alla pagina riferita determina *page fault?*	SI
blocco fisico che contiene la pagina riferita	40

2) Indirizzo A813

l'accesso alla tabella di 2° livello determina *page fault?*	SI
blocco fisico che contiene la tabella di secondo livello	41
l'accesso alla pagina riferita determina *page fault?*	NO
blocco fisico che contiene la pagina riferita	21

3) Indirizzo 9B0C

l'accesso alla tabella di 2° livello determina *page fault?*	NO
blocco fisico che contiene la tabella di secondo livello	07
l'accesso alla pagina riferita determina *page fault?*	NO
blocco fisico che contiene la pagina riferita	1E

4) Indirizzo DA59

l'accesso alla tabella di 2° livello determina *page fault?*	SI
blocco fisico che contiene la tabella di secondo livello	42
l'accesso alla pagina riferita determina *page fault?*	SI
blocco fisico che contiene la pagina riferita	43

SISTEMI OPERATIVI, CORSI A e B - QUINTO APPELLO 2007- 20/7/2007

ESERCIZIO B.2 (4 punti)

Un disco con 8 facce, 200 settori per traccia e 400 cilindri ha un tempo di seek proporzionale al numero di cilindri attraversati e pari a 1 ms per ogni cilindro. Il periodo di rotazione è di 10 msec: conseguentemente il tempo impiegato per percorrere un settore è di 0,1 msec. La politica di scheduling è la SCAN.
A un certo tempo (convenzionalmente indicato come t= 0) termina l'esecuzione dei comandi sul cilindro 243 e pervengono, nell'ordine, le seguenti richieste di lettura o scrittura:

tempo	cil:	sett:	faccia:
0	288	71	7
19	320	5	1
20	270	100	6
28	245	190	5
36	300	55	3
91	390	5	1
200	230	180	2
250	390	60	0

Calcolare il tempo necessario per eseguire tutte queste operazioni, tenendo presente che il tempo di esecuzione di ogni operazione è uguale alla somma dell'eventuale tempo di *seek,* del ritardo rotazionale (tempo necessario per raggiungere il settore indirizzato) e del tempo di percorrenza del settore indirizzato. Per il ritardo rotazionale dopo un'operazione di *seek* si assume sempre il valore di caso peggiore, pari a un periodo di rotazione (10 msec).
Il controllore è dotato di sufficiente capacità di buffering ed è sempre in grado di accettare senza ritardo i dati letti dal disco o quelli da scrivere sul disco.

SOLUZIONE

op. su cilindro:	288	settore:	71					Fase:	SALITA
inizio:	0	seek:	45	rotazione:	10	percorrenza:	0,05	fine:	55,05
op. su cilindro:	300	settore:	55					Fase:	SALITA
inizio:	55,05	seek:	12	rotazione:	10	percorrenza:	0,05	fine:	77,1
op. su cilindro:	320	settore:	5					Fase:	SALITA
inizio:	77,1	seek:	20	rotazione:	10	percorrenza:	0,05	fine:	107,15
op. su cilindro:	390	settore:	5	faccia1				Fase:	SALITA
inizio:	107,15	seek:	70	rotazione:	10	percorrenza:	0,05	fine:	187,2
op. su cilindro:	270	settore:	100	faccia0				Fase:	DISCESA
inizio:	187,2	seek:	120	rotazione:	10	percorrenza:	0,05	fine:	317,25
op. su cilindro:	245	settore:	190					Fase:	DISCESA
inizio:	317,25	seek:	25	rotazione:	10	percorrenza:	0,05	fine:	352,3
op. su cilindro:	230	settore:	180					Fase:	DISCESA
inizio:	352,3	seek:	15	rotazione:	10	percorrenza:	0,05	fine:	377,35
op. su cilindro:	390	settore:	60					Fase:	SALITA
inizio:	377,35	seek:	160	rotazione:	10	percorrenza:	0,05	fine:	547,4

SISTEMI OPERATIVI, CORSI A e B - QUINTO APPELLO 2007- 20/7/2007

ESERCIZIO B.3 (3 punti)

Data la seguente matrice di protezione:

	File 1	File 2	File 3	File 4	Scanner	Stampante
Utente Mario	R,W	X	R,X		R	
Utente Franco		R,X	W			
Utente Rosa	R	W				W
Utente Nina	R		R,X	R,W	R	W

convertirla in:

1. liste di capability
2. liste di controllo degli accessi

SOLUZIONE

1. liste di capability:

Utente Mario: <File1:R,W> → <File2:X> → <File3:R,X> → <Scanner:R>
Utente Franco: <File2:R,X> → <File3:W>
Utente Rosa: <File1:R> → <File2:W> → <Stampante:W>
Utente Nina: <File1:R> → <File3:R,X> → <File4:R,W> → <Scanner:R> → <Stampante:W>

2. liste di controllo degli accessi:

File1: <Mario:R,W> → <Rosa:R> → < Nina:R>
File2: <Mario:X> → <Franco:R,X> → <Rosa:W>
File3: <Mario:R,X> → <Franco:W> → <Nina:R,X>
File4: <Nina:R,W>
Scanner: <Mario:R> → <Nina:R>
Stampante: <Rosa:W> → <Nina:W>

SISTEMI OPERATIVI, CORSI A e B - QUINTO APPELLO 2007- 20/7/2007

ESERCIZIO B.4 (2 punti)

In un sistema UNIX che gestisce la memoria con segmentazione lo spazio logico di ogni processo è suddiviso nei segmenti codice, dati e pila, e il caricamento avviene partizioni variabili. La rilocazione è dinamica e si avvale di una terna di registri base, denominati B_{cod}, B_{dati} e B_{pila}, e di una terna di registri limite, denominati L_{cod}, L_{dati} e L_{pila},.
Al tempo t è presente unicamente il processo P, i cui segmenti codice, dati e pila sono caricati rispettivamente nella partizione Part1 con origine 20.000 e lunghezza 10.000, Part2 con origine 32.000 e lunghezza 18.000 e Part3 con origine 51.000 e lunghezza 9.000. Il sistema operativo occupa la partizione P0 con origine 0 e lunghezza 20.000, ed esiste una partizione libera con origine 60.000 e lunghezza 40.000, La politica si assegnazione della memoria è la *first fit.*
Al tempo *t* il processo P esegue una *fork,* generando il processo F. Si chiede:

1. Il contenuto dei registri base B_{cod}, B_{dati} e B_{pila}, e dei registri limite, denominati L_{cod}, L_{dati} e L_{pila}, al tempo t;
2. Il contenuto dei registri base B_{cod}, B_{dati} e B_{pila}, e dei registri limite, denominati L_{cod}, L_{dati} e L_{pila}, al tempo t; quando il processo F va in esecuzione per la prima volta.

SOLUZIONE

1. Al tempo *t* (è in esecuzione il processo P):

B_{cod} = 20.000 B_{dati} = 32.000 B_{pila},= 51.000
L_{cod} = 10.000 L_{dati} = 18.000 L_{pila},= 9.000

2. Quando il processo F va in esecuzione per la prima volta.

B_{cod} = 20.000 B_{dati} = 60.000 B_{pila},= 78.000
L_{cod} = 10.000 L_{dati} = 18.000 L_{pila},= 9.000

ESERCIZIO B.5 (2 punti)

In un disco con capacità di 64 Gbyte (= 2^{36} byte) e blocchi di 4 Kbyte (= 2^{12} byte), è definito un file system FAT. Gli elementi della FAT sono in corrispondenza biunivoca con i blocchi fisici del disco e ogni elemento contiene l'indice di un blocco del disco.

Si chiede:

1. il numero di elementi che compongono la FAT;
2. la lunghezza (in bit e in byte) di ogni elemento della FAT;
3. lo spazio (in numero di byte) occupato dalla FAT;
4. Supponendo che la FAT sia caricata per intero in memoria e che la lunghezza del blocco di memoria sia pari a 4 Kbyte, il numero di blocchi occupati dalla FAT.

SOLUZIONE

1. numero di elementi che compongono la FAT: $2^{36}/2^{12} = 2^{24}$ elementi;
2. lunghezza (in bit e in byte) di ogni elemento della FAT : 24 bit (3 byte);
3. lo spazio (in numero di byte) occupato dalla FAT: $3 * 2^{24}$ byte (48 Mbyte)
4. il numero di blocchi di memoria occupati dalla FAT $3* 2^{24}/ 2^{12}= 3 *2^{12}$ blocchi.