SISTEMI OPERATIVI, CORSI A e B Prova del 15/7/2009 CORSO |A| |B| Fila…..…... Posto ……

NOME……MATRICOLA …… CORSO |A| |B|

PARTE A

ESERCIZIO A-1

Si consideri la seguente formulazione del problema dei *filosofi a cena* con tre filosofi, denominati F0, F1 e F3. I filosofi sono thread realizzati a livello utente e interagiscono mediante i protocolli *lock, unlock*

```
                Filosofo F0
                {
F0.1 (1 msec)   while (true)    {
F0.2 (10 msec)    pensa
F0.3 (1 msec)     lock(bast0)
F0.4 (1 msec)     lock(bast1)
F0.5 (10 msec)    mangia
F0.6 (1 msec)     unlock(bast0)
F0.7 (1 msec)     unlock(bast1)
                  }
                }
```

```
                Filosofo F1
                {
F1.1 (1 msec)   while (true)    {
F1.2 (10 msec)    pensa
F1.3 (1 msec)     lock(bast1)
F1.4 (1 msec)     lock(bast2)
F1.5 (5 msec)     mangia
F1.6 (1 msec)     unlock(bast1)
F1.7 (1 msec)     unlock(bast2)
                  }
                }
```

```
                Filosofo F2
                {
F2.1 (1 msec)   while (true)    {
F2.2 (5 msec)     pensa
F2.3 (1 msec)     lock(bast2)
F2.4 (1 msec)     lock(bast0)
F2.5 (10 msec)    mangia
F2.6 (1 msec)     unlock(bast2)
F2.7 (1 msec)     unlock(bast0)
                  }
                }
```

I comandi eseguiti dal filosofo *Fi (i= 0 .. 2*) sono numerati *Fi.1 .. Fi.7* e i rispettivi tempi di esecuzione sono indicati in parentesi. La realizzazione dei protocolli lock(K) e unlock(K) sulla chiave di indirizzo K è la seguente:

```
lock(K)
{
loop        TSL K, R1
            JNZ R1, SezCrit
            CALL thread_yield
            JMP loop
SezCrit     RET
}
```

```
unlock(K)
{
MOV K, #1
RET
}
```

dove l'istruzione TSL copia nel registro generale R1 il contenuto della chiave di indirizzo K e, in modo indivisibile, assegna il valore 0 alla chiave.

Ogni thread alterna tra lo stato di pronto e quello di esecuzione. Il processore è gestito con politica *Round Robin,* con quanto di tempo di 8 msec; inoltre lo scheduler interviene anche quando il thread in esecuzione esegue la funzione *thrad_yield.*

Al tempo t= 0 le chiavi di indirizzo bast0, bast1 e bast2, associate ai tre bastoncini condivisi, hanno il valore 1 e pertanto i tre bastoncini sono disponibili. Il filosofo F0 è in esecuzione con il contatore di programma che punta all'inizio del comando F0.3, mentre i filosofi F1 e F2 sono in stato di pronto e i rispettivi contatori di programma puntano all'inizio dei comandi F1.2 e F2.2, rispettivamente.

Utilizzando la tabella sotto riportata, si chiede di descrivere l'evoluzione di F0, F1 e F2 dal tempo *t=0* al tempo *t= 60.* In questo intervallo di tempo, dire quando, per ogni filosofo, inizia e termina una fase *mangia*.

SOLUZIONE

Thread in esecuzione:	Dal tempo t=	Al tempo t=	Comandi eseguiti (indicare in parentesi per quanto tempo ogni comando viene eseguito)	Valore di bast0	Valore di bast1	Valore di bast2

- Il filosofo F0 inizia a mangiare al tempo t= …… e termina al tempo t= ……
- Il filosofo F1 inizia a mangiare al tempo t= …… e termina al tempo t= ……
- Il filosofo F2 inizia a mangiare al tempo t= …… e termina al tempo t= ……

SOLUZIONE

Thread in esecuzione:	Dal tempo t=	Al tempo t=	Comandi eseguiti (indicare in parentesi per quanto tempo ogni comando viene eseguito)	Valore di bast0	Valore di bast1	Valore di bast2
F0	1	8	F0.3 (1), F0.4 (1), F0.5 (6)	0	0	1
F1	9	16	F1.2 (8)	0	0	1
F2	17	23	F2.2 (5), F2.3 (1), F2.4 (1)	0	0	0
F0	24	31	F0.5 (4), F0.6 (1), F0.7 (1), F0.1 (1), F0.2 (1),	1	1	0
F1	32	35	F1.2 (2); F1.3 (1), F1.4 (1)	1	0	0
F2	36	43	F2.4 (1), F2.5 (7)	0	0	0
F0	44	51	F0.2 (8)	0	0	0
F1	52	52	F1.4 (1)	0	0	0
F2	53	60	F2.5 (3), F2.6 (1), F2.7 (1), F2.1 (1); F2.2 (2)	1	0	1

- Il filosofo F0 inizia a mangiare al tempo t= 3 e termina al tempo t= 27
- Il filosofo F2 inizia a mangiare al tempo t= 37 e termina al tempo t= 55

ESERCIZIO A-2 (4 punti)

Un ristorante Fast-Food che somministra un unico tipo di hamburger è gestito da un inserviente e da un cuoco, che interagiscono attraverso uno scaffale, dove possono essere accumulati fino a 10 hamburger pronti per essere serviti. II cuoco prepara gli hamburger in sequenza e li depone sullo scaffale, eventualmente attendendo che ci sia un posto disponibile. Il generico cliente fa il suo ordine all'inserviente e quindi attende la consegna dell'hamburger. L'inserviente riceve in sequenza gli ordini, preleva gli hamburger dallo scaffale (eventualmente attendendo la disponibilità) e li consegna ai clienti, riattivandoli.

Il ristorante è un processo, i cui thread sono i clienti, l'inserviente e il cuoco. Lo scaffale è un buffer di 10 celle, ciascuna capace di contenere un hamburger.

Per l'interazione tra i thread si utilizzano le operazioni della libreria standard *p_thread,* operando sulle variabili *MutexOrdini* e *MutexScaffale* di tipo *mutex,* tutte inizializzate al valore 1, e sulle variabili *AttesaOrdine, consegna, HamburgerNelloScaffale* e *PostoLiberoNelloScaffale* di tipo *condition,* tutte inizializzate con coda vuota.

Si utilizzano inoltre le variabili intere (condivise) *OrdiniPendenti* e *HamburgerPronti,* con valore iniziale 0.

Il generico cliente esegue la seguente funzione:

thread GenericoCliente

```
……
<entra nel negozio>;
pthread_mutex_lock(&MutexOrdini);
OrdiniPendenti ++;
<ordina un hamburger e paga>;
pthread_condition_signal (&AttesaOrdine); //ha effetto solo se l'inserviente è sospeso//
pthread_mutex_unlock(&MutexOrdini);
pthread_mutex_lock(&MutexFittizio);
pthread_condition_wait (&consegna, &MutexFittizio)
//attende incondizionatamente la consegna//
pthread_mutex_unlock(&MutexFittizio);
……
```

dove la variabile *MutexFittizio* è introdotta unicamente per motivi sintattici (la funzione ***pthread_condition_wait*** ha sempre due parametric, di cui il secondo di tipo mutex), e non viene utilizzata altrove.

Si chiede di completare le funzioni eseguite dal thread *Inserviente* e dal thread *Cuoco, inserendo negli schemi sotto riportati* le operazioni sulle variabili di tipo *mutex* e sulle variabili di tipo *condition*.

SOLUZIONE

```
thread Inserviente
do while true{

        while (OrdiniPendenti== 0) //attende un ordine//


        OrdiniPendenti -- //gestisce gli ordini com politica FIFO//;


        while (HamburgerPronti== 0)
                        //attende la disponibilità di almeno un hamburger nello scaffale//

        <preleva un hamburger dallo Scaffale>
        HamburgerPronti --;

        //consegna un hamburger, riattivando i clienti in ordine FIFO//



}

thread Cuoco
do while true{
        //prepara un hamburger//

        while (HamburgerPronti== 10)
                        //attende la disponibiltà di almeno un posto libero nello scaffale //

        <deposita un hamburger sullo Scaffale>
        HamburgerPronti ++;
```

```
        }
```

SOLUZIONE

```
thread Inserviente
do while true{
        pthread_mutex_lock(&MutexOrdini);
        while (OrdiniPendenti== 0) pthread_condition_wait (&AttesaOrdine, &MutexOrdini);
                                //attende un ordine//
        OrdiniPendenti --; //gestisce gli ordini com politica FIFO//;
        pthread_mutex_unlock(&MutexOrdini);
        pthread_mutex_lock(&MutexScaffale);
        while (HamburgerPronti== 0)
                        pthread_condition_wait (&HamburgerNelloScaffale, &MutexScaffale)
                        //attende la disponibilità di almeno un hamburger nello scaffale//
        <preleva un hamburger dallo Scaffale>
        HamburgerPronti --;
        pthread_condition_signal (&PostoLiberoNelloScaffale);
        pthread_mutex_unlock(&MutexScaffale);
        pthread_condition_signal (&consegna)  //consegna un hamburger, riattivando i
                                              //clienti in ordine FIFO//
}

thread Cuoco
do while true{
        //prepara un hamburger//
        pthread_mutex_lock(&MutexScaffale);
        while (HamburgerPronti== 10)
                        pthread_condition_wait (&PostoLiberoNelloScaffale, &MutexScaffale);
                        //attende la disponibiltà di almeno un posto libero nello scaffale //
        <deposita un hamburger sullo Scaffale>
        HamburgerPronti ++;
        pthread_condition_signal (&HamburgerNelloScaffale); //ha effetto solo se
                                                            //l'inserviente è sospeso//
        pthread_mutex_unlock(&MutexScaffale);
        }
```

ESERCIZIO A-3 (3 punti)

In un sistema che applica l'algoritmo del banchiere sono presenti i processi A, B, C, D, E e risorse multiple dei tipi R1, R2, R3, R4. A un certo tempo si è raggiunto lo stato mostrato nelle seguenti tabelle e il processo C richiede due istanze di R2, con il vincolo che la richiesta deve essere soddisfatta per intero o altrimenti non soddisfatta, con la conseguente sospensione del processo richidente.

Assegnazione attuale				
	R1	R2	R3	R4
A	2	1		1
B	1	1	1	
C		1		2
D			1	1
E	1	1		1

Esigenza residua (considerata la molteplicità e l'assegnazione attuale				
	R1	R2	R3	R4
A	1	0	1	1
B	2	4	1	3
C	2	2	3	1
D	1	0	0	1
E	0	1	0	1

Molteplicità			
R1	R2	R3	R4
5	6	3	7

Disponibilità			
1	2	1	2

Si chiede di applicare l'algoritmo del banchiere per verificare se la richiesta del processo A può essere soddisfatta.

SOLUZIONE

Stato S raggiunto dopo l'ipotetica assegnazione di due istanze di R2 al processo C:

Assegnazione attuale

Esigenza residua (considerata la molteplicità e l'assegnazione attuale

Molteplicità

	R1	R2	R3	R4
A	2	1		1
B	1	1	1	
C		3		2
D			1	1
E	1	1		1

	R1	R2	R3	R4
A	1	0	1	1
B	2	4	1	3
C	2	0	3	1
D	1	0	0	1
E	0	1	0	1

R1	R2	R3	R4
5	6	3	7

Disponibilità			
1	0	1	2

Successivamente:
il processo A può terminare, la disponibilità diventa <3,1,1,3>
il processo D può terminare, la disponibilità diventa <3,1,2,4>
il processo E può terminare, la disponibilità diventa <4,2,2,5>

A questo punto né B né C possono terminare

Di conseguenza: lo stato S non è sicuro e la richista del processo A non può essere soddisfatta.

SOLUZIONE

Stato S raggiunto dopo l'ipotetica assegnazione di due istanze di R2 al processo C:

Assegnazione attuale				
	R1	R2	R3	R4
A				
B				
C				
D				
E				

Esigenza residua (considerata la molteplicità e l'assegnazione attuale				
	R1	R2	R3	R4
A				
B				
C				
D				
E				

Molteplicità			
R1	R2	R3	R4
5	6	3	7

Disponibilità			

Successivamente:
il processo …… può terminare, la disponibilità diventa <……>
il processo …… può terminare, la disponibilità diventa <……>
il processo …… può terminare, la disponibilità diventa <……>
il processo …… può terminare, la disponibilità diventa <……>
il processo …… può terminare, la disponibilità diventa <……>

Di conseguenza: lo stato S è [sicuro/non sicuro]: …….. e la richista del processo C non può essere [non soddisfatta/soddisfatta]: ……...

ESERCIZIO A-4 (2 punti)
Si consideri un sistema dove sono presenti i processi P1, P2 e P3 e le risorse A, B, C, D , tutte con molteplicità 3. I processi, che inizialmente non possiedono risorse, 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 combinano in modo arbitrario con quelle degli altri processi. Dopo aver ottenuto e utilizzato le risorse che gli sono necessarie, ogni processo termina rilasciando tutte le risorse ottenute.
Si consideri la sequenza di richieste sotto riportate:

Processo	Prima richiesta	Seconda richiesta	Terza richiesta	Terminazione
P1	1 istanza di B	2 istanze di C	1 istanza di A	Rilascia tutto
P2	1 istanza di C	2 istanze di D	1 istanza di A	Rilascia tutto
P3	2 istanze di B	1 istanza di C	2 istanze di A	Rilascia tutto

Dire se i processi evitano la possibilità di stallo e motivare la risposta.

SOLUZIONE
I processi evitano la possibilità di stallo? [SI/NO]……
Spiegazione: ……

……

SOLUZIONE
I processi evitano la possibilità di stallo.
Infatti assumendo per le risorse l'ordinamento B → C → D → A, tutti i processi rispettano questo ordinamento per le loro richieste.

ESERCIZIO A-5 (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 descrittori *fd[0]* e *fd[1];*
b) il processo P esegue con successo una *fork,* generando il processo F1 che immediatamente esegue con successo una *exec;*
c) il processo P esegue la chiamata *close(fd[1])*
d) il processo P esegue con successo una *fork,* generando il processo F2 che immediatamente esegue con successo una *exec;*
e) il processo F1 esegue l'operazione *write(fd[1], ciao, 5)*
f) il processo F2 esegue con sucesso una chiamata *pipe,* che restituisce i descrittori *fd'[0]* e *fd'[1];*
g) il processo F2 esegue l'operazione *write(fd'[1], babbo, 5)*
h) il processo P esegue l'operazione *read(fd[0], &buffer, 10).*

1) Qual è il contenuto di *buffer* dopo l'operazione h)?
2) Quale valore restituisce la chiamata *read?*

SOLUZIONE

1) Dopo l'operazione h), il contenuto di *buffer* è …….
2) La chiamata *read* rstituisce il valore …….

Spiegazione:
……..……..……..……
……..……..……..……

SOLUZIONE

3) Dopo l'operazione h), il contenuto di *buffer* è *"ciao"*
4) La chiamata *read* rstituisce il valore *5.*

Spiegazione: il processo F2 scrive la sua stringa su un pipe diverso da quello sul quale ha scritto il processo F1 e legge il processo P.

PARTE B

ESERCIZIO B-1 (4 punti)

Un disco con 100 cilindri, 4 facce e 50 settori per traccia ha un tempo di seek proporzionale al numero di cilindri attraversati e pari a 0,5 ms per ogni cilindro. Il periodo di rotazione è di 5 msec: conseguentemente il tempo impiegato per percorrere un settore è di 0,1 msec.
A un certo tempo (convenzionalmente indicato come t= 0) termina l'esecuzione dei comandi sul cilindro 61 e sono pervenute, nell'ordine, le seguenti richieste di lettura o scrittura:

- clindro 79, faccia 0, settore 10;
- clindro 31, faccia 2, settore 12;
- clindro 77, faccia 1, settore 5;
- clindro 98, faccia 3, settore 45.

Successivamente arricano le seguenti richieste:

- al tempo 17, per il clindro 65, faccia 0, settore 5;
- al tempo 18, per il clindro 65, faccia 2, settore 40;
- al tempo 30 per il clindro 40, faccia 1, settore 10;

Calcolare il tempo necessario per eseguire tutte queste operazioni supponendo che si adotti la politica di scheduling SSTF (Shortest Seek-time First).
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. 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.
Per il ritardo rotazionale dopo un'operazione di *seek* si assume sempre il valore di caso peggiore, pari a un intero periodo di rotazione (5 msec). Si assume inoltre che i comandi sullo stesso cilindro vengano eseguiti in ordine FIFO.

SOLUZIONE

op.su cilindro:		settore:		faccia:					
inizio:		seek:		rotazione:		percorrenza:		fine:	
op.su cilindro:		settore:		faccia:					
inizio:		seek:		rotazione:		percorrenza:		fine:	
op.su cilindro:		settore:		faccia:					
inizio:		seek:		rotazione:		percorrenza:		fine:	
op.su cilindro:		settore:		faccia:					
inizio:		seek:		rotazione:		percorrenza:		fine:	
op.su cilindro:		settore:		faccia:					
inizio:		seek:		rotazione:		percorrenza:		fine:	
op.su cilindro:		settore:		faccia:					
inizio:		seek:		rotazione:		percorrenza:		fine:	
op.su cilindro:		settore:		faccia:					
inizio:		seek:		rotazione:		percorrenza:		fine:	

SOLUZIONE

op.su cilindro: 77		settore: 5		faccia: 1					
inizio:	0	seek:	8	rotazione:	5	percorrenza:	0,1	fine:	13,1
op.su cilindro: 79		settore: 10		faccia: 0					
inizio:	13,1	seek:	1	rotazione:	5	percorrenza:	0,1	fine:	19,2

op.su cilindro: 65		settore: 5		faccia: 0					
inizio:	19,2	seek:	7	rotazione:	5	percorrenza:	0,1	fine:	31,3
op.su cilindro: 65		settore: 40		faccia: 2					
inizio:	31,3	seek:	0	rotazione:	3,4	percorrenza:	0,1	fine:	34,8
op.su cilindro: 40		settore: 10		faccia: 1					
inizio:	34,8	seek:	12,5	rotazione:	5	percorrenza:	0,1	fine:	52,4
op.su cilindro: 31		settore: 12		faccia: 2					
inizio:	52,3	seek:	4,5	rotazione:	5	percorrenza:	0,1	fine:	62,0
op.su cilindro: 98		settore: 45		faccia: 3					
inizio:	61,9	seek:	33,5	rotazione:	5	percorrenza:	0,1	fine:	100,6

ESERCIZIO B-2 (4 punti)

In un sistema che gestisce la memoria con paginazione, sono presenti i processi A, B e C. Lo stato di occupazione della memoria al tempo 11 è descritto dalla seguente *Core Map*, dove per ogni blocco si specifica il processo a cui appartiene la pagina caricata e l'indice della pagina medesima Ad esempio, nel blocco 11 è caricata la pagina 6 del processo B.

Proc		A	A		A		C	A	B	A	C	B	C	B	B	C	A	B	A	B		C		C
Pag		4	5		8		0	1	8	2	5	6	3	4	5	9	10	10	11	3		7		2
Blocco	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

A un certo tempo le tabelle delle pagine dei tre processi sono le seguenti, dove il campo *Rif* contiene il tempo virtuale (locale al processo) al quale è avvenuto l'ultimo riferimento alla pagina.

Pagina	Blocco	Rif
0		
1	7	10
2	9	11
3		
4	1	17
5	2	14
6		
7		
8	4	16
9		
10	16	18
11	18	20
Processo A		

Pagina	Blocco	Rif
0		
1		
2		
3	19	6
4	13	8
5	14	13
6	11	12
7		
8	8	15
9		
10	17	18
11		
Processo B		

Pagina	Blocco	Rif
0	6	0
1		
2	23	2
3	12	3
4		
5	10	5
6		
7	21	12
8		
9	15	8
10		
11		
Processo C		

Per la gestione della memoria si utilizza un algoritmo di sostituzione LRU locale con controllo del working set. Il working set assegnato ai tre processi (inteso come numero di blocchi a disposizione del processo, anche se momentaneamente non occupati) ha dimensione 7. Al verificarsi di un errore di pagina, la pagina riferita viene caricata in un blocco disponibile se il numero di blocchi occupati dal processo è minore del limite assegnato al suo working set; in caso contrario si applica l'algoritmo di sostituzione. I blocchi disponibili vengono assegnati in ordine crescente di indice.
Si chiede come si modificano la *Core Map* e le Tabelle delle pagine se, a partire dal tempo considerati, si verificano le seguenti sequenze di riferimenti alla memoria (considerare le due alternative):

- Alternativa 1:
  - ai tempi 21, 22, 23 e 24 il processo A riferisce le pagine 2, 3, 7, 8;
  - quindi ai tempi 13, 14, 15 e 16 il processo C riferisce le pagine 0, 1, 4, 5

- Alternativa 2:
  - ai tempi 13, 14, 15 e 16 il processo C riferisce le pagine 9, 7, 6, 5;
  - quindi ai tempi 19, 20, 21 e 22 il processo B riferisce le pagine 10, 11, 6, 7

SOLUZIONE

- Alternativa 1 (mostrare solo gli elementi che cambiano)

Proc																							
Pag																							
Blocco	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

Pagina	Blocco	Rif
0		
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
Processo A		

Pagina	Blocco	Rif
0		
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
Processo B		

Pagina	Blocco	Rif
0		
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
Processo C		

- Alternativa 2: (mostrare solo gli elementi che cambiano)

Proc																							
Pag																							
Blocco	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

Pagina	Blocco	Rif
0		
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
Processo A		

Pagina	Blocco	Rif
0		
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
Processo B		

Pagina	Blocco	Rif
0		
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
Processo C		

SOLUZIONE

- Alternativa 1:
  - ai tempi 21, 22, 23 e 24 il processo A riferisce le pagine 2, 3, 7, 8;
  - quindi ai tempi 13, 14, 15 e 16 il processo C riferisce le pagine 0, 1, 4, 5

Proc	C	A	A		A		C	A	B	A	C	B	C	B	B	C	A	B	A	B		C		C
Pag	1	4	7		8		0	3	8	2	5	6	3	4	5	9	10	10	11	7		7		4
Blocco	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

Pagina	Blocco	Rif
0		
1		
2	9	21
3	7	22
4	1	17
5		
6		
7	2	23
8	4	24
9		
10	16	18
11	18	20
Processo A		

Pagina	Blocco	Rif
0		
1		
2		
3	19	6
4	13	8
5	14	13
6	11	12
7		
8	8	15
9		
10	17	18
11		
Processo B		

Pagina	Blocco	Rif
0	6	13
1	0	14
2		
3	12	3
4	23	15
5	10	16
6		
7	21	12
8		
9	15	8
10		
11		
Processo C		

- Alternativa 2:
  - ai tempi 13, 14, 15 e 16 il processo C riferisce le pagine 9, 7, 6, 5;
  - quindi ai tempi 19, 20, 21 e 22 il processo B riferisce le pagine 10, 11, 6, 7

Proc	C	A	A		A	B	C	A	B	A	C	B	C	B	B	C	A	B	A	B		C		C
Pag	6	4	5		8	11	0	1	8	2	5	6	3	4	5	9	10	10	11	7		7		2
Blocco	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

Pagina	Blocco	Rif
0		
1	7	10
2	9	11
3	3	
4	1	17
5	2	14
6		
7		
8	4	16
9		
10	16	18
11	18	20
Processo A		

Pagina	Blocco	Rif
0		
1		
2		
3		
4	13	8
5	14	13
6	11	21
7	19	22
8	8	15
9		
10	17	19
11	3	20
Processo B		

Pagina	Blocco	Rif
0	6	0
1		
2	23	2
3	12	3
4		
5	10	16
6	0	15
7	21	14
8		
9	15	13
10		
11		
Processo C		

ESERCIZIO B-3 (3 punti)

Si consideri un sistema dove gli indirizzi logici hanno la lunghezza di 32 bit e le pagine logiche e fisiche hanno ampiezza di 4 kByte. Per la gestione della memoria con paginazione dinamica si utilizzano tabelle delle pagine a 2 livelli. La tabella di primo livello comprende 2^{10} elementi.
Gli elementi di ogni tabella di primo o secondo livello hanno una lunghezza di 4 byte. Un byte è riservato agli indicatori (pagina caricata, riferita, modificata, ecc.) mentre i rimanenti codificano un indice di blocco fisico.
Si chiede:

1. la lunghezza del campo offset, in numero di bit;
2. il numero di elementi contenuti in ogni tabella di secondo livello;
3. lo spazio occupato in memoria da ogni tabella di secondo livello (numero di byte);

4. la massima dimensione della memoria fisica, espressa in numero di blocchi e in numero di byte.

Inoltre, supponendo che la tabella di primo livello sia caricata in memoria e che nessuna delle tabelle di secondo livello interessate sia caricata, si consideri la seguente sequenza di riferimenti a pagine:

a) pagina 7268
b) pagina 20490
c) pagina 31720

si chiede quali tabelle di secondo livello devono essere caricate in memoria per portare a termine la traduzione degli indirizzi,supponendo che le informazioni per la traduzione non sono contenute nella cache della MMU

SOLUZIONE

1. lunghezza del campo offset : 12 bit
2. per la codifica degli indici di blocco sono disponibili 20 bit, di cui 10 sono indici della tabella di primo livello e i rimanenti 10 sono indici delle tabelle di secondo livello. Pertanto la tabella di primo livello e ogni tabella di secondo livello contengono 2^{10} elementi
3. spazio occupato in memoria da ogni tabella di secondo livello : $2^{10}*4= 2^{12}$ byte → 4k Byte
4. gli indici dei blocchi di memoria fisica sono codificati con 3 byte→ 24 bit
   massima dimensione della memoria fisica : 2^{24} blocchi → $2^{24} * 2^{12} = 2^{36}$ byte → 64 Gbyte

Le tabelle di secondo livello da caricare in memoria sono le seguenti:

a) per il riferimento alla pagina 7268 = 7* 2^{10}+ 100 si deve caricare la tabella di secondo livello di indice 7
b) per il riferimento alla pagina 20490 = 20* 2^{10}+ 10 si deve caricare la tabella di secondo livello di indice 20
c) per il riferimento alla pagina 31720 = 30* 2^{10}+ 1000 si deve caricare la tabella di secondo livello di indice 30.

SOLUZIONE

1. lunghezza del campo offset : ……
2. per la codifica degli indici di blocco sono disponibili …… bit, di cui …… sono indici della tabella di primo livello e i rimanenti …… sono indici delle tabelle di secondo livello.
   Pertanto la tabella di primo livello contiene …… elementi
   ogni tabella di secondo livello contiene …… elementi
3. spazio occupato in memoria da ogni tabella di secondo livello : …… Byte
4. gli indici dei blocchi di memoria fisica sono codificati con …… bit
   massima dimensione della memoria fisica : …… blocchi =…… Byte

Le tabelle di secondo livello da caricare in memoria sono le seguenti:

a) per il riferimento alla pagina 7268 si deve caricare la tabella di secondo livello di indice ……
b) per il riferimento alla pagina 20490 si deve caricare la tabella di secondo livello di indice ……
c) per il riferimento alla pagina 31720 si deve caricare la tabella di secondo livello di indice ……

ESERCIZIO B-4 (2 punti)

Un disco è organizzato con *NCilindri= 100* (numerati da 0 a 99), *NFacce= 4* (numerate da 0 a 3) e *NSettori= 50* (numerati da 0 a 49). Ogni settore contiene 2^{10}= 1024 byte e corrisponde a un blocco.
A livello logico i blocchi sono individuati con *indici di blocco,* interi compresi nell'intervallo *[0, capacità_in_blocchi)* .
Gli indici di blocco sono definiti secondo la sequenza *cilindro, faccia, settore.*
Si chiede:

1) il numero di blocchi contenuti in ogni cilindro
2) la capacità del disco, in numero di blocchi e di byte
3) quanti cilindri vengono attraversati dall'operazione di *seek* per leggere il blocco *1500* dopo aver letto il blocco *500*

SOLUZIONE

1) Ogni cilindro contiene …… blocchi;
2) la capacità del disco è : …… blocchi, pari a …… Kbyte
3) il blocco *500* è contenuto nel cilindro ……*;* il blocco *1500* è contenuto nel cilindro …… quindi si attraversano …… cilindri.

SOLUZIONE

1) Ogni cilindro contiene 4* 50 = 200 blocchi;
2) la capacità del disco è : 100 * 200 blocchi = 20.000 blocchi, pari a 20.000 Kbyte ~ 19,53 Mbyte
3) il blocco *500* è contenuto nel cilindro *500* ***div*** *200= 2;* il blocco *1500* è contenuto nel cilindro *1500* ***div*** *200= 7* quindi si attraversano 5 cilindri.

ESERCIZIO B.5 (2 punti)

Data la seguente lista di controllo degli accessi (ACL):

File1: <Alessia: RWX> → <Luigi: - - X> → <Marco: RW ->
File2: <Alessia: RW -> → <Carla: R - X> → <Luigi: RWX>
File3: <Alessia: R - -> → <Carla: RW - > → <Marco: R - ->
DirA: <Alessia: RWX> → <Luigi: R - X> → <Marco: - - X>
Stampante: <Alessia: - W -> → <Marco: - W ->

Convertirla nella corrispondente lista di capabilities (C-list).

SOLUZIONE

……
……
……
……
……
……

SOLUZIONE

Alessia: <File 1: RWX> → <File2: RW -> → <File3: R - -> →<DirA: <RWX> →< Stampante: <- W ->
Carla: <File2: R - X -> → <File3: RW ->
Luigi: <File 1: - - X> → <File2: RWX> → <DirA: <R - X>
Marco: <File 1: RW -> → <File3: R-- -> →<DirA: <- - X> →< Stampante: <- W ->