

# SISTEMI OPERATIVI, CORSI A e B - SESTO APPELLO 2007- 6/9/2007

## ESERCIZIO A-1 (4 punti)

In un sistema operativo che realizza i threads a livello utente, i thread P e C del processo P cooperano eseguendo i seguenti flussi di controllo per inserire ed estrarre dati da una pila, che ha una capacità di max elementi.

Tread P	Tread C
.....	.....
1 XXX	<b>a thread_yield</b>
1.1 <definisce_item(&NewItem);	<b>b thread_lock (&amp;Chiave);</b>
2 <b>thread_lock (&amp;Chiave);</b>	b.1 while top==0 {
2.1 while top==max {	<b>c thread_unlock(&amp;Chiave);</b>
3 <b>thread_unlock(&amp;Chiave);</b>	<b>d thread_lock(&amp;Chiave);</b>
4 <b>thread lock(&amp;Chiave);</b> }	d.1 ReceivedItem= Pila[top];
4.1 top= top+1;	d.2 top= top- 1;
4.2 Pila[top]= item;	d.3 IniziaElaborazione(ReceivedItem)
5 <b>thread unlock(&amp;Chiave)</b>	<b>e thread_yield</b>
5.1 YYY	e.1 CompletaElaborazione(ReceivedItem)
.....	<b>f thread_unlock(&amp;Chiave);</b>
	f.1 YYY
	.....

dove l'espansione assembler dei comandi **thread\_lock** e **thread\_unlock** è la seguente:

```

thread_lock(K)
loop      TSL K, R1 //copia K in R1 e scrive 0 in K//
          JNZ R1, Fine //esegue il salto se R1•0// 
          CALL thread_yield
          JMP loop //salto incondizionato//
Fine      NOP //No OPeration// 

thread_unlock(K)
MOV K, #1 //scrive1 in K//
CALL thread_yield

```

I due thread alternano tra lo stato di esecuzione e quello di pronto: il thread C passa in esecuzione quando il thread P rilascia il processore eseguendo il comando **thread\_yield** e, viceversa, il thread P passa in esecuzione quando il thread C rilascia il processore eseguendo il comando **thread\_yield**.

Al tempo  $t$  la pila è piena (e quindi si ha  $\text{top}=\text{max}$ ) e si ha  $\text{Chiave}=1$  (e quindi si può accedere alla sezioni critiche). Il thread P è in esecuzione ed ha eseguito il comando 1, mentre l'ultimo comando eseguito dal thread C, che è in stato di pronto, è il comando **a**.

Supponendo che l'esecuzione di ogni comando (compresi la **thread\_lock** e **thread\_unlock**) richieda un tempo unitario, si chiede quanto tempo impiega il thread P a completare l'inserzione di un elemento nella pila, fino all'esecuzione del comando 5. Per calcolare il tempo, utilizzare la tabella sotto riportata, compilando una riga per ciascuno dei comandi riportati in grassetto (comandi 2, 3, 4 e 5 del thread P e comandi b, c, d, e, f del thread C). Ogni riga presenta la situazione che si stabilisce dopo l'esecuzione del comando corrispondente.

## SOLUZIONE

Tempo	Comando eseguito ...	.... dal thread	Thread che eseguirà il prossimo comando.	Prossimo comando del thread P	Prossimo comando del thread C
t	1	P	P	1.1	b
t+ 2	2	P	P	2.1	b
t+ 4	3	P	C	4	b
t+ 5	b	C	C	4	b.1
t+ 10	e	C	P	4	e.1
t+ 11	4	P	C	4	e.1
t+ 13	f	C	P	4	f.1
t+ 18	5	P	C	5.1	f.1
t+					
t+					

Il comando con il quale il thread P completa l'inserzione nella pila viene eseguito al tempo t+ 18

# SISTEMI OPERATIVI, CORSI A e B - SESTO APPELLO 2007- 6/9/2007

## ESERCIZIO A.2 (4 punti)

Un sistema con 4 processi A, B, C, D e risorse dei tipi R1, R2, R3, R4, rispettivamente di molteplicità [3, 5, 5, 4] applica la politica di riconoscimento ed eliminazione dello stallo. Al tempo  $t$  si è raggiunto lo stato descritto dalle seguenti tabelle:

Assegnazione attuale				
	R1	R2	R3	R4
A	0	1	1	0
B	1	1	2	0
C	0	1	1	2
D	2	1	1	0

Esigenza residua				
	R1	R2	R3	R4
A	1	1	0	2
B	0	2	1	1
C	2	2	0	0
D	0	0	2	3

Molteplicità				
	R1	R2	R3	R4
	3	5	5	4

Disponibilità				
	R1	R2	R3	R4
	0	1	0	2

Dopo il tempo  $t$ , il processo A richiede una risorsa di tipo R1, il processo B richiede una risorsa di tipo R3, il processo C richiede una risorsa di tipo R1 e il processo D richiede una risorsa di tipo R3. E' immediato verificare che si raggiunge uno stallo. Per la sua eliminazione si considerano le seguenti alternative:

- a. soppressione del processo A;

- b. sottrazione di 2 risorse di tipo R1 al processo D (con conseguente incremento dell'esigenza residua di questo processo).

Si consideri ciascuna di queste alternative e si verifichi se consente di eliminare lo stallo.

## SOLUZIONE

Alternativa a)

Stato raggiunto dopo la soppressione del processo A:

Assegnazione attuale				
	R1	R2	R3	R4
A	-	-	-	-
B	1	1	2	0
C	0	1	1	2
D	2	1	1	0

Esigenza residua				
	R1	R2	R3	R4
A	-	-	-	-
B	0	2	1	1
C	2	2	0	0
D	0	0	2	3

Molteplicità				
	R1	R2	R3	R4
	3	5	5	4

Disponibilità				
	R1	R2	R3	R4
	0	2	1	2

Verifica:

- 1) Il processo B può terminare

Dopo la terminazione, la disponibilità di {R1, R2, R3, R4} diviene {1, 3, 3, 2}

- 2) Il processo C non può terminare

- 3) Il processo D non può terminare

Di conseguenza: l'alternativa a) non consente di eliminare lo stallo

Alternativa b)

Stato raggiunto dopo la sottrazione di 2 risorse di tipo R1 al processo D:

Assegnazione attuale				
	R1	R2	R3	R4
A	0	1	1	0
B	1	1	2	0
C	0	1	1	2
D	0	1	1	0

Esigenza residua				
	R1	R2	R3	R4
A	1	1	0	2
B	0	2	1	1
C	2	2	0	0
D	2	0	2	3

Molteplicità				
	R1	R2	R3	R4
	3	5	5	4

Disponibilità				
	R1	R2	R3	R4
	2	1	0	2

Verifica:

- 1) Il processo A può terminare

Dopo la terminazione, la disponibilità di {R1, R2, R3, R4} diviene {2, 2, 1, 2}

- 2) Il processo B può terminare

Dopo la terminazione, la disponibilità di {R1, R2, R3, R4} diviene {3, 3, 3, 2}

- 3) Il processo C può terminare

Dopo la terminazione, la disponibilità di {R1, R2, R3, R4} diviene {3, 4, 4, 4}

- 4) Il processo D può terminare

Dopo la terminazione, la disponibilità di {R1, R2, R3, R4} diviene {3, 5, 5, 4}

Di conseguenza: l'alternativa b) consente di eliminare lo stallo

# SISTEMI OPERATIVI, CORSI A e B - SESTO APPELLO 2007- 6/9/2007

## ESERCIZIO A-3 (3 punti)

In un sistema vengono generati i processi A,B,C,D con i tempi di arrivo e le durate (in millisecondi) sotto specificate:

Processo	Durata	Tempo di arrivo
A	15	0
B	15	5
C	25	18
D	5	30

La politica di scheduling è la Round Robin, con quanto di tempo pari a 10 msec. Si suppone che, dopo il passaggio in esecuzione, ogni processo avanzi senza sospendersi fino all'esaurimento del quanto di tempo o fino alla terminazione.

Mostrare nella tabella come evolve l'utilizzo del processore e calcolare per ogni processo il tempo di permanenza nel sistema, definito come differenza tra il tempo di terminazione e il tempo di arrivo. Trascurare il tempo di commutazione di contesto.

## SOLUZIONE

T=	Arriva	Termina	In esec.	Coda Pronti
0	A		A	∅
5	B		A	B
10			B	A
18	C		B	A → C
20			A	C → B
25		A	C	B
30	D		C	B → D
35			B	D → C

T=	Arriva	Termina	In esec.	Coda Pronti
40		B	D	C
45		D	C	∅
55			C	∅
60		C	∅	∅

Tempo di permanenza:

$$A : 25$$

$$B: 40- 5= 35$$

$$C: 60- 18= 42$$

$$D: 45- 30= 15$$

## ESERCIZIO A-4 (2 punti)

Si consideri un sistema nel quale sono definiti il semaforo *sem* e i processi P1 (con priorità 1), P2 (con priorità 1) e P3 (con priorità 2). Lo scheduling avviene con una politica a priorità, che prevede il prerilascio e assegna il processore al processo pronto di priorità più elevata (a pari priorità applica la politica FIFO). La politica applicata al semaforo è la FIFO. Al tempo *t* il semaforo *sem* ha valore 0, e la sua coda contiene P3.

Allo stesso tempo, il processo P1 è in esecuzione il processo P2 è pronto.

Come si modificano il semaforo *sem* e la CodaPronti e quale processo è in esecuzione se si verificano (in alternativa) le seguenti sequenze di eventi:

- a) P1 esegue *wait(sem)* e successivamente il processo in esecuzione esegue *signal(sem)*;
- b) P1 esegue *signal(sem)* e successivamente il processo in esecuzione esegue *signal(sem)*;

## SOLUZIONE

	Sequenze di eventi	In Esecuzione	Coda Pronti	Valore di <i>sem</i>	Coda di <i>sem</i>
a-1	P1 esegue <i>wait(sem)</i>	P2	∅	0	P3 → P1
a-2	Il processo in esecuzione esegue <i>signal(sem)</i>	P3	P2	0	P1
b-1	P1 esegue <i>signal(sem)</i>	P3	P2 → P1	0	∅
b-2	Il processo in esecuzione esegue <i>signal(sem)</i>	P3	P2 → P1	1	∅

# SISTEMI OPERATIVI, CORSI A e B - SESTO APPELLO 2007- 6/9/2007

## ESERCIZIO A-5 (2 punti)

Dire che cosa viene stampato dal processo che esegue il seguente frammento di codice, considerando sia il caso in cui la *fork* ha successo, sia quello in cui la *fork* fallisce.

```
...
printf("uno");
a = fork();
if (a>0) {
    printf("due");
    exec("prova",NULL);
    printf("tre");
}
else {
    printf("quattro");
    exec("altro",NULL);
    if (a<0) exit();
}
printf("cinque");
...
}
```

## SOLUZIONE

Il processo che esegue l'intero frammento è il "padre", quello che esegue la fork.

Il padre stampa inizialmente "uno", quindi:

- se la fork ha successo stampa "due", esegue la exec di "prova" e:
  - se la exec fallisce, stampa "tre" e "cinque"; altrimenti non stampa nient'altro;
- se la fork fallisce, stampa "quattro", esegue la exec di altro e:
  - se la exec fallisce termina, altrimenti non stampa nient'altro;

Il figlio (se viene effettivamente generato per il buon esito della fork) stampa "quattro", esegue la exec di "altro" e:

- se la exec fallisce, stampa "cinque"; altrimenti non stampa nient'altro;

# SISTEMI OPERATIVI, CORSI A e B - SESTO APPELLO 2007- 6/9/2007

## ESERCIZIO B-1 (4 punti)

Un sistema simile a Unix gestisce la memoria con segmentazione e caricamento in partizioni variabili con politica *best fit*. Per ogni processo sono definiti i segmenti codice, dati e stack. I processi controllati dallo stesso codice condividono il segmento codice. L'ampiezza della memoria è di 35 Mbyte e il sistema operativo occupa una partizione con origine 0 e lunghezza 3.

Al tempo  $t$  sono caricati in memoria i seguenti processi (le lunghezze delle partizioni sono espresse in Mbyte):

- il processo A, che occupa la partizione A1 (con origine 9 e lunghezza 2) per il codice, la partizione A2 (con origine 22 e lunghezza 3) per i dati e la partizione A3 (con origine 17 e lunghezza 2) per lo stack;
- il processo B, che occupa la partizione B1 (con origine 30 e lunghezza 4) per il codice, la partizione B2 (con origine 26 e lunghezza 2) per i dati e la partizione B3 (con origine 3 e lunghezza 2) per lo stack.

Successivamente:

- al tempo  $t+x$  il processo A esegue una *fork*, con la quale viene generato il processo C.
- al tempo  $t+y$  il processo B esegue una *fork*, con la quale viene generato il processo D.
- al tempo  $t+w$  il processo C esegue una *exec*. Le lunghezze dei nuovi segmenti codice, dati e stack sono rispettivamente di 4 Mbyte, 3 Mbyte e 1 Mbyte;
- al tempo  $t+z$  il processo D esegue una *exec*. Le lunghezze dei nuovi segmenti codice, dati e stack sono rispettivamente di 3 Mbyte, 3 Mbyte e 2 Mbyte.

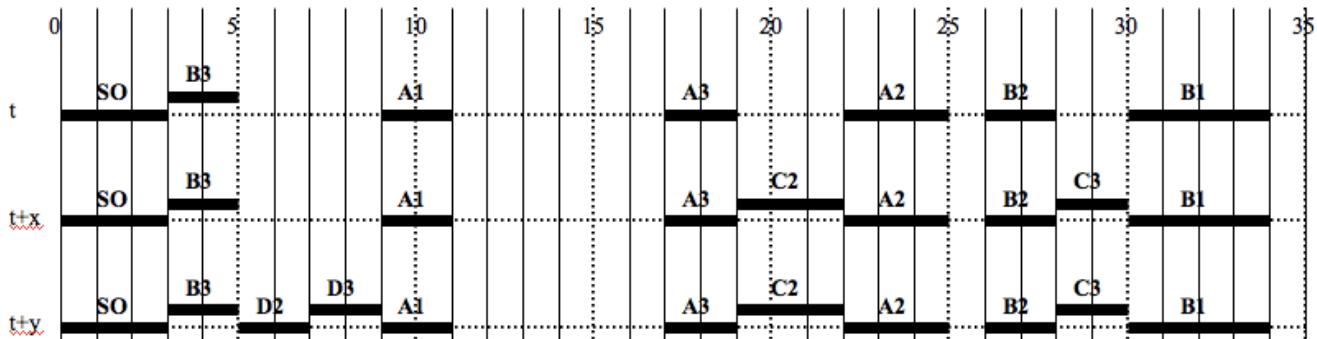
L'operazione *exec* è realizzata rilasciando prima le partizioni precedentemente assegnate ai segmenti del processo e assegnando quindi le partizioni ai nuovi segmenti.

Si chiede:

1. la configurazione della memoria ai tempi  $t+x$  e  $t+y$ ;
2. la configurazione della memoria ai tempi  $t+w$  e  $t+z$ .

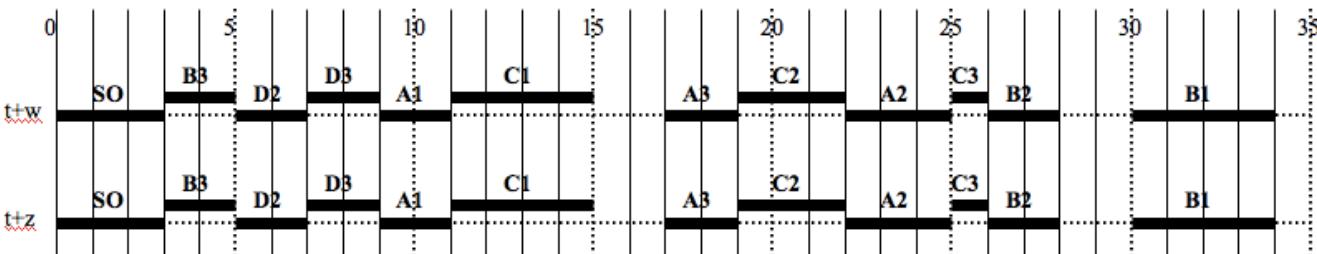
## SOLUZIONE

1. Configurazione della memoria ai tempi  $t+x$  e  $t+y$ .



2. Configurazione della memoria ai tempi  $t+w$  e  $t+z$ .

La *exec* eseguita dal processo D al tempo  $t+w$  fallisce perché dopo aver caricato il segmento codice nella partizione liberata rilasciando i precedenti segmenti D2 e D3, non si trova una partizione di lunghezza sufficiente per caricare il segmento dati. Pertanto la sostituzione di codice non avviene.



# SISTEMI OPERATIVI, CORSI A e B - SESTO APPELLO 2007- 6/9/2007

## ESERCIZIO B.2 (4 punti)

In un file system UNIX i blocchi del disco hanno ampiezza di 1Kbyte e i puntatori ai blocchi sono a 16 bit. Gli i-node contengono, oltre agli altri attributi, 10 indirizzi diretti e 3 indirizzi indiretti.

Si consideri il file rappresentato dal seguente i-node:

ind	0	1	2	3	4	5	6	7	8	9	10	11	12
Blocco fisico	70	340	412	44	609	610	611	612	981	172	179	198	199

Alcuni frammenti dei blocchi 179, 198, 199, 7000, 7001, 7002 e 7003 sono riportati nel seguito.

Blocco fisico 179:

Indice nel blocco	0	1	2	3	4	5	6	7	8	9	...
Blocco fisico	9100	9101	9122	9271	8987	9765	9810	8897	9456	9500	...

Blocco fisico 198:

Indice nel blocco	0	1	2	3	4	5	6	7	8	9	...
Blocco fisico	7000	7001	7002	7003	7101	7102	7103	7107	7200	7210	...

Blocco fisico 199:

Indice nel blocco	0	1	2	3	4	5	6	7	8	9	...
Blocco fisico	6121	6122	6123	6124	6125	6126	6100	6101	6102	6103	...

Blocco fisico 7000:

Indice nel blocco	0	1	2	3	4	5	6	7	8	9	...
Blocco fisico	211	210	205	203	201	208	207	212	204	202	...

Blocco fisico 7001:

Indice nel blocco	0	1	2	3	4	5	6	7	8	9	...
Blocco fisico	5001	5002	5003	5004	5005	5006	5007	5008	5009	5010	...

Blocco fisico 7002:

Indice nel blocco	0	1	2	3	4	5	6	7	8	9	...
Blocco fisico	5804	5800	5801	5802	5803	5987	5988	5989	5990	5991	...

Blocco fisico 7003:

Indice nel blocco	0	1	2	3	4	5	6	7	8	9	...
Blocco fisico	6780	6787	6788	6789	6700	6568	6569	6570	6571	6572	...

Si chiede:

1. il numero di indirizzi contenuti in ogni blocco indiretto;
2. la massima dimensione di un file (in caratteri)
3. in quali blocchi fisici sono allocati i seguenti blocchi logici del file: 5, 10, 15, 530, 2065

## SOLUZIONE

1. Ogni blocco indiretto contiene  $2^{10}/2 = 512$  indirizzi
2. la massima dimensione di un file è di  $10 + 512 + 512^2 + 512^3$  blocchi =  $(10 + 512 + 512^2 + 512^3) * 1024$  caratteri
3. Per  $0 \leq ind \leq 9$  al blocco logico di indice  $ind$  corrisponde il blocco fisico individuato dall'indirizzo diretto  $ind$   
Per  $10 \leq ind < 512 + 10 = 522$ , al blocco logico di indice  $ind$  corrisponde il blocco fisico individuato dall'indirizzo  $ind-10$  del blocco indiretto semplice.  
Per  $522 \leq ind < 512^2 + 10 = 2621544$ , con al blocco logico di indice  $ind$  corrisponde il blocco fisico individuato:
  - dall'indirizzo  $(ind-522) \ mod \ 512$  del blocco indiretto di secondo livello,
  - a sua volta individuato dall'indirizzo  $(ind-522) \ div \ 512$  del blocco indiretto di primo livello,
  - a sua volta individuato dall'indirizzo indiretto doppio.

BLOCCO LOGICO	BLOCCO FISICO	Raggiunto attraverso:		
		INDIRIZZO DIRETTO	BLOCCO INDIRETTO PRIMO LIVELLO	BLOCCO INDIRETTO SECONDO LIVELLO
5	610	5	Blocco ..... Ind. nel blocco.....	Blocco ..... Ind. nel blocco.....
10	9100		Blocco 179 Ind. nel blocco 0	Blocco ..... Ind. nel blocco.....
15	9765		Blocco 179 Ind. nel blocco 5	Blocco ..... Ind. nel blocco.....
530	204		Blocco 198 Ind. nel blocco 0	Blocco 7000 Ind. nel blocco 8
2065	6570		Blocco 198 Ind. nel blocco 3	Blocco 7003 Ind. nel blocco 7

# SISTEMI OPERATIVI, CORSI A e B - SESTO APPELLO 2007- 6/9/2007

## ESERCIZIO B-3 (3 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  $2^8$  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 del processo in esecuzione è caricata permanentemente nel blocco 0 (codice binario 00000000) della memoria principale, che pertanto non è disponibile per il caricamento di pagine del processo o di tabelle delle pagine di secondo livello. Quando risiede in memoria, Ogni tabella di secondo livello occupa (seppure incompletamente) 1 blocco di memoria.

Gli elementi delle tabelle (di primo o di secondo livello) sono codici binari di 8 bit. Il codice 00000000 denota una pagina non presente in memoria, mentre ogni altro codice individua il blocco fisico che contiene la pagina.

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. Il processo riferisce in sequenza i seguenti indirizzi logici:

1. 0010 0111 1100 1001
2. 0111 0111 1000 0001
3. 0111 1011 1010 1001
4. 1010 0001 0000 1001
5. 1100 0110 1000 1001
6. 1110 1110 0100 1111

Per ogni riferimento si chiede:

- se si verifica errore di pagina,
- in caso contrario, l'indirizzo fisico corrispondente.

	Blocco		Blocco		Blocco		Blocco		Blocco		Blocco
0000	11110000	0000	01110000	0000	10010000	0000	00000000	0000	10100000	0000	00000000
0001	11110001	0001	01110001	0001	10010001	0001	10000001	0001	10100001	0001	00000000
0010	11110010	0010	00000000	0010	00000000	0010	10000010	0010	00000000	0010	00000000
0011	11110011	0011	01110011	0011	00000000	0011	00000000	0011	00000000	0011	01100011
0100	11110100	0100	01110100	0100	00000000	0100	10000100	0100	10100100	0100	00000000
0101	11110101	0101	00000000	0101	10010101	0101	00000000	0101	00000000	0101	00000000
0110	11110110	0110	01110110	0110	00000000	0110	00000000	0110	00000000	0110	01100110
0111	11110111	0111	01110111	0111	00000000	0111	10000111	0111	00000000	0111	00000000
1000	11111000	1000	01111000	1000	10011000	1000	00000000	1000	10101000	1000	01101000
1001	11111001	1001	00000000	1001	00000000	1001	10001001	1001	10101001	1001	00000000
1010	11111010	1010	00000000	1010	10011010	1010	00000000	1010	00000000	1010	01101010
1011	11111011	1011	01111011	1011	10011011	1011	00000000	1011	00000000	1011	00000000
1100	11111100	1100	01111100	1100	10011100	1100	10001100	1100	10101100	1100	01101100
1101	11111101	1101	00000000	1101	00000000	1101	10001101	1101	10101101	1101	01101101
1110	11111110	1110	01111110	1110	00000000	1110	00000000	1110	00000000	1110	01101110
1111	11111111	1111	00000000	1111	10011111	1111	10001111	1111	00000000	1111	00000000
Tab. 1° livello		Blocco 11110010 (Tab. 2° livello di indice 0010)		Blocco 11110111 (Tab. 2° livello di indice 0111)		Blocco 11110111 (Tab. 2° livello di indice 0111)		Blocco 11111010 (Tab. 2° livello di indice 1010)		Blocco 11111100 (Tab. 2° livello di indice 1110)	

## SOLUZIONE

- |  |                                      |
|--|--------------------------------------|
| 1. Indirizzo logico 0010 0111 1100 1001: Errore di pagina NO | Indirizzo fisico 0111 0111 1100 1001 |
| 2. Indirizzo logico 0111 0111 1000 0001: Errore di pagina SI |                                      |
| 3. Indirizzo logico 0111 1011 1010 1001: Errore di pagina NO | Indirizzo fisico 1001 1011 1010 1001 |
| 4. Indirizzo logico 1010 0001 0000 1001: Errore di pagina NO | Indirizzo fisico 1000 0001 0000 1001 |
| 5. Indirizzo logico 1100 0110 1000 1001: Errore di pagina SI |                                      |
| 6. Indirizzo logico 1110 1110 0100 1111: Errore di pagina NO | Indirizzo fisico 0110 1110 0100 1111 |

# SISTEMI OPERATIVI, CORSI A e B - SESTO APPELLO 2007- 6/9/2007

## ESERCIZIO B-4 (2 punti)

In un sistema UNIX, a un certo tempo sono presenti i processi P e Q, appartenenti a gruppi diversi.

Al tempo  $t_1$  il processo P apre il file *pippo*, la cui lunghezza corrente è di 280 caratteri, e legge 100 caratteri da questo file. Al tempo  $t_2$ , P esegue con successo una fork, generando il processo F.

Al tempo  $t_3$  il processo F comanda la lettura di 200 caratteri dal file *pippo* e al tempo  $t_4$  scrive 100 caratteri su questo file.

Al tempo  $t_5$  anche il processo Q apre il file *pippo* e comanda la lettura di 300 caratteri da questo file.

Si chiede:

1. la posizione del puntatore di lettura del processo F prima della lettura eseguita al tempo  $t_3$ ;
2. la posizione del puntatore di lettura del processo F dopo questa lettura e il numero di caratteri letti;
3. la posizione del puntatore di lettura del processo F dopo la scrittura eseguita al tempo  $t_4$ ;
4. la posizione del puntatore di lettura del processo Q dopo la lettura eseguita al tempo  $t_5$

## SOLUZIONE

1. Posizione del puntatore di lettura del processo F prima della lettura eseguita al tempo  $t_3$ : 100
2. Posizione del puntatore di lettura del processo F dopo questa lettura e numero di caratteri letti: 280; letti 180 caratteri
3. Posizione del puntatore di lettura del processo F dopo la scrittura eseguita al tempo  $t_4$ : 280
4. Posizione del puntatore di lettura del processo Q dopo la lettura eseguita al tempo  $t_5$ : 300.

## ESERCIZIO B-5 (2 punti)

In un file system UNIX si considerino i file */usr/luigi/giochi/scacchi.exe* e */usr/luigi/giochi/manualescacchi.pdf* appartenenti all'utente *luigi*, e si consideri anche l'utente *anna*.

I diritti associati alle directory *usr*, *luigi*, *manuali* e ai file *scacchi.exe* e *manualescacchi.pdf* sono i seguenti

	<i>owner (r w x)</i>	<i>group (r w x)</i>	<i>others (r w x)</i>
<i>usr</i>	<i>1 0 1</i>	<i>1 0 1</i>	<i>1 0 1</i>
<i>luigi</i>	<i>1 1 1</i>	<i>1 0 1</i>	<i>1 0 1</i>
<i>giochi</i>	<i>1 1 1</i>	<i>1 0 1</i>	<i>1 0 0</i>
<i>scacchi.exe</i>	<i>1 1 1</i>	<i>1 0 1</i>	<i>1 0 1</i>
<i>manualescacchi.pdf</i>	<i>1 1 1</i>	<i>1 0 0</i>	<i>1 0 0</i>

Quali tra le operazioni di lettura, scrittura ed esecuzione possono essere eseguite sui file *scacchi.exe* e *manualescacchi.pdf* dall'utente *anna* nelle seguenti ipotesi:

1. *luigi* e *anna* appartengono allo stesso gruppo;
2. *luigi* e *anna* appartengono a gruppi diversi;

## SOLUZIONE

IPOTESI	LETTURA	SCRITTURA	ESECUZIONE
<i>luigi</i> e <i>anna</i> appartengono allo stesso gruppo	<i>scacchi.exe</i> : SI <i>manualescacchi.pdf</i> : SI	<i>scacchi.exe</i> : NO <i>manualescacchi.pdf</i> : NO	<i>scacchi.exe</i> : SI <i>manualescacchi.pdf</i> : NO
<i>luigi</i> e <i>anna</i> appartengono a gruppi diversi.	<i>scacchi.exe</i> : NO <i>manualescacchi.pdf</i> : NO	<i>scacchi.exe</i> : NO <i>manualescacchi.pdf</i> : NO	<i>scacchi.exe</i> : NO <i>manualescacchi.pdf</i> : NO