SISTEMI OPERATIVI, CORSI A e B Prova del 4/9/2009 CORSO |A| |B| Fila…….. Posto ……

**COGNOME E NOME……MATRICOLA ……**

**ESERCIZIO A-1 (4 punti)**

Un sistema Unix controlla un impianto industriale tramite i Pthread `R`, `A1` e `A2`. Il Pthread `R` (rilevatore) effettua un ciclo nel quale rileva e analizza dei dati di temperatura dell'impianto e scrive il risultato dell'analisi in un buffer condiviso. Il dato prodotto dal rilevatore è acquisito dai due Pthread attuatori `A1` e `A2` che periodicamente e in modo indipendente tra loro e dal rilevatore leggono l'ultimo valore disponibile dal buffer e pilotano di conseguenza delle valvole.

Dato che i thread attuatori non modificano il contenuto del buffer possono accedere al buffer contemporaneamente. D'altra parte deve essere garantita la mutua esclusione nell'accesso al buffer tra il thread rilevatore e i thread attuatori.

La politica adottata dà la priorità nell'accesso al buffer al rilevatore e pertanto se il rivelatore è in attesa di accedere al buffer non è consentito l'accesso agli attuatori.

Per gestire il buffer sono impiegate le seguenti variabili condivise (è anche mostrato il valore col quale sono inizializzate).
`Attuatori_attivi=0, Attuatori_in_attesa=0; Rilevatore_in_Attesa=false; Rilevatore_attivo=false.`

Inoltre sono usati:

- La variabile `mutex` per gestire la mutua esclusione.
- La variabile di condizione `condRilevatore` per sospendere il thread rilevatore nel caso che uno o più thread attuatori stiano leggendo dal buffer.
- La variabile di condizione `condAttuatori` per sospendere i thread attuatori nel caso che il thread rilevatore sia in attesa di accedere al buffer o stia scrivendo nel buffer.

Dato il codice degli attuatori, si chiede di scrivere il codice del rilevatore.

**Pthread Attuatori:**

```
while (true) {
       Pthread_mutex_lock(&mutex);
         while (Rilevatore_attivo || Rivelatore_in_attesa) {
                Attuatori_in_attesa++;
                pthread_cond_wait(&condAttuatori,&mutex);
         }
         Attuatori_attivi++;
       Pthread_mutex_unlock(&mutex);

       <legge dal buffer>

       Pthread_mutex_lock(&mutex);
         Attuatori_attivi--;
         if (Attuatori_attivi==0)
                pthread_cond_signal(&condRilevatore);
       Pthread_mutex_unlock(&mutex);
       <controlla le valvole>
}
```

**SOLUZIONE**

**Pthread rilevatore:**

```
while (true) {
       <rileva e analizza la temperatura>
       Pthread_mutex_lock(&mutex);
         Rilevatore_in_Attesa=true;
         while (Attuatori_attivi>0)
                pthread_cond_wait(&condRilevatore,&mutex);
         Rilevatore_in_Attesa=false;
         Rilevatore_attivo=true;
       Pthread_mutex_unlock(&mutex);

       <scrive nel buffer>

       Pthread_mutex_lock(&mutex);
         Rilevatore_attivo=false;
         while (Attuatori_in_attesa>0) {
            pthread_cond_signal(&condAttuatori);
            Attuatori_in_attesa--;
            }
       Pthread_mutex_unlock(&mutex);
}
```

**ESERCIZIO A-2 (4 punti)**

Si consideri un processore che dispone dei seguenti registri:

- i registri speciali PC (program counter) e PS (program status)
- un banco di registri riservato allo stato utente, che comprende i registri generali R1, R2 e lo stack pointer SP,
- un ulteriore banco di registri riservato allo stato supervisore, che comprende i registri generali R'1, R'2 e lo stack pointer SP'.

Il sistema gestisce il processore con politica a priorità. Al tempo t sono presenti nel sistema il processo Pi (con priorità 5), in stato di esecuzione, il processo Pj (con priorità 4) in coda pronti e il processo Pk (con priorità 6) bloccato sul semaforo *mux*. Al tempo t il processo Pi invoca la chiamata di sistema *wait(mux)*. L'invocazione della chiamata di sistema viene effettuata tramite l'istruzione SVC che provoca un'interruzione. Immediatamente prima che l'interruzione venga riconosciuta, i registri del processore, i descrittori di Pi, Pj e Pk e lo stack del nucleo hanno i contenuti mostrati in tabella.

L'interruzione determina l'intervento del nucleo, che salva sullo stack del nucleo i registri PC e PS ed esegue la funzione di servizio dell'interruzione corrispondente alla chiamata di sistema. Supponendo che il vettore di interruzione associato all'interruzione generata da SVC sia 0600 e che la parola di stato del nucleo sia 275E, si chiede:

a) il contenuto dei descrittori, dei registri generali e speciali, dello stack del nucleo e lo stato del processore durante la fase di estrazione della prima istruzione della funzione di servizio;
b) il contenuto dei descrittori, dei registri generali e speciali, dello stack del nucleo e lo stato del processore durante la fase di estrazione dell'istruzione IRET con la quale termina la chiamata di sistema;
c) il contenuto dei descrittori, dei registri generali e speciali, dello stack del nucleo e lo stato del processore durante la fase di estrazione dell'istruzione eseguita subito dopo la IRET.

| Descrittore di $P_I$ | |
|---|---|
| Stato | Esec |
| Priorità | 5 |
| PC | 2FF0 |
| PS | 16F2 |
| SP | 4010 |
| R1 | 00AB |
| R2 | 00CD |

| Descrittore di $P_J$ | |
|---|---|
| Stato | Pronto |
| Priorità | 4 |
| PC | 7000 |
| PS | 16F2 |
| SP | 9000 |
| R1 | 1100 |
| R2 | 11AA |

| Descrittore di $P_k$ | |
|---|---|
| Stato | blocc. |
| Priorità | 6 |
| PC | A000 |
| PS | 16F2 |
| SP | C000 |
| R1 | 22CC |
| R2 | 22DD |

| Stack del nucleo | |
|---|---|
| …… | …… |
| 1016 | 0000 |
| 1015 | |
| 1014 | |
| 1013 | |
| 1012 | |
| 1011 | |

| Registri stato utente | |
|---|---|
| SP | 4000 |
| R1 | 0011 |
| R2 | 0012 |
| | |
| Registri stato superv. | |
| SP | 1016 |
| R'1 | AAAA |
| R'2 | BBBB |
| | |
| | |

| Processore: registri speciali e di stato | | | | | | |
|---|---|---|---|---|---|---|
| PC | 2F00 | PS | 16F2 | stato | utente | |

**SOLUZIONE**

a) contenuto dei descrittori, dei registri generali e speciali, dello stack del nucleo e lo stato del processore durante la fase di estrazione della prima istruzione della funzione di servizio:

| Descrittore di $P_I$ | |
|---|---|
| Stato | Esec |
| Priorità | 5 |
| PC | Invar. |
| PS | Invar. |
| SP | Invar. |
| R1 | Invar. |
| R2 | Invar. |

| Descrittore di $P_J$ | |
|---|---|
| Stato | Pronto |
| Priorità | 4 |
| PC | Invar. |
| PS | Invar. |
| SP | Invar. |
| R1 | Invar. |
| R2 | Invar. |

| Descrittore di $P_k$ | |
|---|---|
| Stato | Blocc. |
| Priorità | 6 |
| PC | Invar. |
| PS | Invar. |
| SP | Invar. |
| R1 | Invar. |
| R2 | Invar. |

| Stack del nucleo | |
|---|---|
| …… | …… |
| 1016 | 0000 |
| 1015 | 2F00 |
| 1014 | 16F2 |
| 1013 | |
| 1012 | |
| 1011 | |

| Registri stato utente | |
|---|---|
| SP | Invar. |
| R1 | Invar. |
| R2 | Invar. |
| | |
| Registri stato superv. | |
| SP | 1014 |
| R'1 | AAAA |
| R'2 | BBBB |
| | |
| | |

| Processore: registri speciali e di stato | | | | | | |
|---|---|---|---|---|---|---|
| PC | 0600 | PS | 275E | stato | Superv. | |

b) contenuto dei descrittori, dei registri generali e speciali, dello stack del nucleo e lo stato del processore durante la fase di estrazione dell'istruzione IRET con la quale termina la chiamata di sistema;

| Descrittore di $P_I$ | |
|---|---|
| Stato | Blocc |
| Priorità | 5 |
| PC | 2F00 |
| PS | 16F2 |
| SP | 4000 |
| R1 | 0011 |
| R2 | 0012 |

| Descrittore di $P_J$ | |
|---|---|
| Stato | Esec |
| Priorità | 4 |
| PC | Invar. |
| PS | Invar. |
| SP | Invar. |
| R1 | Invar. |
| R2 | Invar. |

| Descrittore di $P_k$ | |
|---|---|
| Stato | Blocc. |
| Priorità | 6 |
| PC | Invar. |
| PS | Invar. |
| SP | Invar. |
| R1 | Invar. |
| R2 | Invar. |

| Stack del nucleo | |
|---|---|
| …… | …… |
| 1016 | 0000 |
| 1015 | 7000 |
| 1014 | 16F2 |
| 1013 | |
| 1012 | |
| 1011 | |

| Registri stato utente | |
|---|---|
| SP | 9000 |
| R1 | 1100 |
| R2 | 11AA |
| | |
| Registri stato superv. | |
| SP | 1014 |
| R'1 | ? |
| R'2 | ? |
| | |
| | |

| Processore: registri speciali e di stato | | | | | | |
|---|---|---|---|---|---|---|
| PC | 0600+? | PS | 275E | stato | Superv. | |

c) contenuto dei descrittori, dei registri generali e speciali, dello stack del nucleo e lo stato del processore durante la fase di estrazione dell'istruzione eseguita subito dopo la IRET.

| Descrittore di $P_I$ | |
|---|---|
| Stato | Blocc |
| Priorità | 5 |
| PC | Invar. |
| PS | Invar. |
| SP | Invar. |
| R1 | Invar. |
| R2 | Invar. |

| Descrittore di $P_J$ | |
|---|---|
| Stato | Esec |
| Priorità | 4 |
| PC | Invar. |
| PS | Invar. |
| SP | Invar. |
| R1 | Invar. |
| R2 | Invar. |

| Descrittore di $P_k$ | |
|---|---|
| Stato | blocc. |
| Priorità | 6 |
| PC | Invar. |
| PS | Invar. |
| SP | Invar. |
| R1 | Invar. |
| R2 | Invar. |

| Stack del nucleo | |
|---|---|
| …… | …… |
| 1016 | 0000 |
| 1015 | |
| 1014 | |
| 1013 | |
| 1012 | |
| 1011 | |

| Registri stato utente | |
|---|---|
| SP | Invar. |
| R1 | Invar. |
| R2 | Invar. |
| | |
| Registri stato superv. | |
| SP | 1016 |
| R'1 | ? |
| R'2 | ? |
| | |
| | |

| Processore: registri speciali e di stato | | | | | | |
|---|---|---|---|---|---|---|
| PC | 7000 | PS | 16F2 | stato | utente | |

**ESERCIZIO A-3 (3 punti)**

In un processo P sono definiti il thread A e B, realizzati con scheduling *round robin* e quanto di tempo di 10 *msec.* I thread A e B cooperano scambiando messaggi attraverso un buffer condiviso che può contenere un solo messaggio. Le parti rilevanti del codice dei thread A e B sono le seguenti:

```
Thread A:
   a.1)      while true {
   a.2)       <produce  mess>
   a.3)       lock(&BufferVuoto);
   a.4)       <deposita mess in buffer>
   a.5)       unlock(&BufferPieno);
             }
```

```
Thread B:
   b.1)      while true {
   b.2)       lock(&BufferPieno);
   b.3)       <preleva mess dal buffer>
   b.4)       unlock(&BufferVuoto);
   b.5)       <consuma mess>
             }
```

Le funzioni `lock` e `unlock` sono realizzate nel modo elementare con l'istruzione *TSL*, senza incorporare la `ThreadYield`. Il valore 1 della chiave significa che la risorsa associata è libera.
Le righe di codice a1, a4, a5, b1, b3 e b4 hanno tempo di esecuzione trascurabile. Il tempo di esecuzione delle linee a3 e b2 è trascurabile se il valore iniziale della chiave è 1; altrimenti ha un valore positivo per ogni iterazione. Le linee a2 e b5 hanno tempo di esecuzione di 6 msec.
Al tempo *T* il buffer è pieno, e le chiavi `BufferPieno` e `BufferVuoto` hanno rispettivamente valore 1 e 0.
Il thread A è pronto, con il contatore di programma posizionato sull'inizio della linea a1), mentre il thread B termina l'esecuzione della riga b1 e contemporaneamente esaurisce il quanto di tempo.
Si chiede quanto tempo occorre al thread A per depositare due messaggi nel buffer, nell'ipotesi che non siano presenti altri thread nel sistema.

**SOLUZIONE**

1) Primo messaggio depositato al tempo T+ 20 Infatti:
   - Thread A esegue a1, a2 e poi a3 per 10 msec;
   - Thread B esegue b2, b3, b4, b5, b1, b2 per 10 msec; (`BufferPieno=0; BufferVuoto=1`)
   - Thread A esegue a3 e a4 per 0 msec e deposita il messaggio al tempo T+20; (`BufferPieno=0;BufferVuoto=0`)

2) Secondo messaggio depositato al tempo T+ 40 Infatti:
   - Thread A esegue a5, a1, a2, a3 per 10 msec; (`BufferPieno=1;BufferVuoto=0`)
   - Thread B esegue b2, b3, b4 per 6 msec, quindi b1, b2 per 4 msec; (`BufferPieno=0;BufferVuoto=1`)
   - Thread A a3, a4 per 0 msec e deposita il mesaggio al tempo T+40;

**ESERCIZIO A-4 (2 punti)**
In un sistema tipo Unix che adotta una politica di schedulazione a code multiple sono definiti il semaforo *sem* e i processi P1 e P2 con priorità 5 e 4, rispettivamente. Al tempo *t* il processo P1 è in esecuzione, la coda pronti contiene il processo P2 e il semaforo *sem* ha valore 0. Dopo il tempo *t* si verifica la seguente sequenza di eventi:

a) P1 esegue la chiamata di sistema *fork()* che genera il processo P3;
b) Scade il quanto di tempo;
c) Il processo in esecuzione esegue *wait(sem);*
d) Il processo in esecuzione esegue *signal(sem);*

Si chiede di specificare quale processo è in esecuzione dopo ogni evento e inoltre come si modificano le code dei thread pronti e il valore e la coda del semaforo *sem*.

**SOLUZIONE**

| Sequenza di eventi | In Esecuzione | Coda priorità 5 | Coda priorità 4 | Valore di sem | Coda di sem |
|---|---|---|---|---|---|
| 1) P1 esegue *fork()* | P1 (5) | P3 (5) | P2 (4) | 0 | - |
| 2) Scade il quanto di tempo; | P3 (5) | P1 (5) | P2 (4) | 0 | - |
| 3) il processo in esecuzione esegue *wait(sem)* | P1 (5) | | P2 (4) | 0 | P3 (5) |
| 4) Il processo in esecuzione esegue *signal(sem)* | P1 (5) | P3 (5) | P2 (4) | 0 | - |

**ESERCIZIO A-5 (2 punti)**
In un sistema con risorse R1, R2, R3 e R4, 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 possono intercalarsi 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 consideri la sequenza di richieste sotto riportate:

| Processo | Prima richiesta | Seconda richiesta | Terza richiesta | Quarta richiesta | Terminazione |
|---|---|---|---|---|---|
| P1 | 1 istanza di R5 | 1 istanza di R3 | 1 istanza di R4 | 1 istanza di R1 | Rilascia tutto |
| P2 | 1 istanza di R3 | 1 istanza di R4 | 1 istanza di R1 | 1 istanza di R2 | Rilascia tutto |
| P3 | 1 istanza di R5 | 1 istanza di R4 | 1 istanza di R2 | - | Rilascia tutto |

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

**SOLUZIONE**
I processi evitano lo stallo perché richiedono le risorse in modo ordinato.
L'ordinamento delle risorse ai fini della politica di prevenzione dello stallo è: R5, R3, R4, R1, R2

**ESERCIZIO B-1 (4 punti)**

In un file system UNIX i blocchi del disco hanno ampiezza di 1Kbyte e i puntatori ai blocchi sono a 32 bit. Gli i-node contengono, oltre agli altri attributi, 10 puntatori diretti e 3 puntatori indiretti.
Tenendo presente che il primo blocco del disco ha indice logico 0, si chiede:

1. il numero di puntatori che possono essere contenuti in un blocco indiretto;
2. l'indice logico del primo blocco e dell'ultimo blocco indirizzabili con puntatori diretti;
3. l'indice logico del primo blocco e dell'ultimo blocco indirizzabili con indirizzamento indiretto semplice;
4. l'indice logico del primo blocco e dell'ultimo blocco indirizzabili con indirizzamento indiretto doppio;
5. l'indice logico del primo blocco e dell'ultimo blocco indirizzabili con indirizzamento indiretto triplo;

Considerato il file (aperto) individuato dal file descriptor *fd,* la cui lunghezza corrente (in byte) è 278.538 e il cui *i-node* contiene i seguenti puntatori a blocchi:

| Puntatore | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Valore del puntatore | 100 | 101 | 102 | 120 | 121 | 122 | 300 | 301 | 302 | 303 | 500 | 700 | -- |

dove i blocchi indiretti 500, 700, e 800 hanno i seguenti contenuti parziali:

Blocco 500:

| Indice di elemento nel blocco | 0 | 1 | 2 | 3 | 4 | 5 | ……. |
|---|---|---|---|---|---|---|---|
| Valore del puntatore | 304 | 305 | 306 | 307 | 308 | 309 | ……. |

Blocco 700:

| Indice di elemento nel blocco | 0 | 1 | 2 | 3 | 4 | 5 | ……. |
|---|---|---|---|---|---|---|---|
| Valore del puntatore | 800 | 801 | 802 | 850 | 851 | 852 | ……. |

Blocco 800:

| Indice di elemento nel blocco | 0 | 1 | 2 | 3 | 4 | 5 | ……. |
|---|---|---|---|---|---|---|---|
| Valore del puntatore | 1200 | 1201 | 1202 | 1203 | 1204 | 1205 | ……. |

si chiede inoltre:

6. il numero di blocchi che compongono il file;
7. quali sono i blocchi indiretti che vengono letti per eseguire l'operazione *read(fd,&buf,1*) quando lo I/O pointer ha valore 12.298
8. quali sono i blocchi indiretti che vengono letti per eseguire l'operazione *read(fd,&buf,1)* quando lo I/O pointer ha valore 273.428.

**SOLUZIONE**

1. il numero di puntatori che possono essere contenuti in un blocco indiretto è 2^8= 256;
2. il primo e l'ultimo blocco indirizzabili con puntatori diretti hanno rispettivamente indici logici 0 e 9;
3. il primo e l'ultimo blocco indirizzabili con indirizzamento indiretto semplice hanno rispettivamente indici logici 10 e (10+ 2^8)- 1= 265;
4. il primo e l'ultimo blocco indirizzabili con indirizzamento indiretto doppio hanno rispettivamente indici logici 10+ 2^8= 266 e (10+ 2^8+ 2^16)- 1= 65.801;
5. il primo e l'ultimo blocco indirizzabili con indirizzamento indiretto triplo hanno rispettivamente indici logici 10+ 2^8+ 2^16= 65.802 e (10+ 2^8+ 2^16+ 2^24)- 1= 16.843.017;
6. l'ultimo carattere del file è contenuto nel blocco 278.538 ***div*** 2^10= 272; quindi il file è composto da 273 blocchi
7. il carattere *12.298,* sul quale è posizionato il puntatore di lettura, è contenuto nel blocco di indice logico *12.298* ***div*** 2^10= 12, indirizzato dal puntatore di indice 2 e valore 306 del blocco indiretto 500, individuato dal puntatore 10 dello *i-node*. Quindi per eseguire l'operazione *read(fd,&buf,1*) deve essere letto il blocco indiretto 500.
8. il carattere *273.428,* sul quale è posizionato il puntatore di lettura, è contenuto nel blocco di indice logico *273.428* ***div*** 2^10= 267, individuato dal puntatore di indice 1 e valore 1201 nel blocco indiretto 800, a sua volta individuato dal puntatore di indice 0 del blocco indiretto 700 corrispondente al puntatore 11 dello i-node. Quindi per eseguire l'operazione *read(fd,&buf,1*) devono essere letti il blocco indiretto di secondo livello700 e il blocco indiretto di primo livello 800.

**ESERCIZIO B-2 (4 punti)**
Un sistema RAID 4 è composto da 6 dischi ($D_0$,…,$D_5$) indipendenti (non sincronizzati) e le strip contenenti le parità sono contenute nel disco 5.
Al tempo t il contenuto delle strip di indice 69, 70 e 71 è il seguente:

| | $D_0$ | $D_1$ | $D_2$ | $D_3$ | $D_4$ | $D_5$ |
|---|---|---|---|---|---|---|
| Strip 69 | 110… | 101… | 111… | 001… | 100… | 001… |
| Strip 70 | 010… | 000… | 011… | 110… | 111… | 000… |
| Strip 71 | 011… | 001… | 100… | 100… | 010… | 000… |

A partire dal tempo t avvengono in sequenza i seguenti eventi:
t+1. Scrittura del valore 010… sulla strip 69 del disco $D_1$;
t+2. Guasto del disco $D_1$;
t+3. Scrittura del valore 000… sulla strip 70 del disco $D_0$;
t+4. Lettura del contenuto della strip 71 del disco $D_1$;
Si chiede:
1. Quali operazioni sui dischi sono necessarie per completare l'operazione al tempo t+1
2. Quali operazioni sui dischi sono necessarie per completare l'operazione al tempo t+3
3. Quali operazioni sui dischi sono necessarie per completare l'operazione al tempo t+4
4. Il contenuto delle strip 69, 70 e 71 dei dischi al tempo t+5

**SOLUZIONE**
1. Operazioni necessarie per scrivere 010… sulla strip 69 del disco $D_1$:
- **lettura** della strip 69 del disco $D_1$ per leggere il vecchio dato;
- EXOR tra il vecchio dato (101…) e il nuovo dato (010…) per calcolare la maschera dei bit modificati;
- **scrittura** del nuovo dato (010…) sulla strip 69 del disco $D_1$;
- **lettura** della strip 69 del disco $D_5$ (per leggere il vecchio valore di parità) ;
- EXOR tra la maschera dei bit modificati e la vecchia parità per calcolare la nuova parità (110…)
- **scrittura** della nuova parità sulla strip 69 del disco $D_5$;
Totale operazioni su disco: 4

2. Operazioni necessarie per scrivere 000… sulla strip 70 del disco $D_0$:
- **lettura** della strip 70 del disco $D_0$ per leggere il vecchio dato;
- EXOR tra il vecchio dato (010…) e il nuovo dato (000…) per calcolare la maschera dei bit modificati;
- **scrittura** del nuovo dato (000…) sulla strip 70 del disco $D_0$;
- **lettura** della strip 70 del disco $D_5$ (per leggere il vecchio valore di parità);
- EXOR tra la maschera dei bit modificati e la vecchia parità per calcolare la nuova parità (010…)
- **scrittura** della nuova parità sulla strip 70 del disco $D_5$;
Totale operazioni su disco: 4

3. Operazioni necessarie per leggere il contenuto della strip 71 del disco $D_1$
- **leggere** la strip 71 dei dischi $D_0$ , $D_2$, $D_3$, $D_4$, $D_5$;
- calcolo del contenuto della strip 70 del disco $D_1$ dato dallo EXOR tra tutti i dati letti e uguale a 000…
Totale operazioni su disco: 5

4. Contenuto delle strip 69, 70 e 71 dei dischi al tempo t+5:

| | $D_0$ | $D_1$ | $D_2$ | $D_3$ | $D_4$ | $D_5$ |
|---|---|---|---|---|---|---|
| Strip 69 | 110… | | 111… | 001… | 100… | 110… |
| Strip 70 | 000… | | 011… | 110… | 111… | 010… |
| Strip 71 | 011… | | 100… | 100… | 010… | 000… |

**ESERCIZIO B-3 (3 punti)**
In un sistema operativo che gestisce la memoria con paginazione e segmentazione, il processo P utilizza uno spazio di memoria virtuale composto da tre segmenti S1, S2, S3, rispettivamente di lunghezza 100.000, 68.000 e 91.000.
Ogni pagina ha dimensione di 2 Kbyte e ogni descrittore di pagina contiene, oltre al blocco associato alla pagina, un indicatore binario che indica se la pagina è presente in memoria (1 indica che è presente). La tabella seguente mostra un frammento delle tabelle delle pagine associate ai segmenti S1, S2, e S3:

| S1: | |
|---|---|
| Pagina | Descrittore |
| 78 | 1980, 1 |
| 79 | -----, 0 |
| … | … |
| 101 | 1888, 1 |
| 102 | -----, 0 |

| S2: | |
|---|---|
| Pagina | Descrittore |
| 78 | -----, 0 |
| 79 | 2619, 1 |
| … | … |
| 101 | 2222, 1 |
| 102 | -----, 0 |

| S3: | |
|---|---|
| Pagina | Descrittore |
| 78 | -----, 0 |
| 79 | 3099, 1 |
| … | … |
| 101 | 3398, 1 |
| 102 | 3987, 1 |

Il processo genera in sequenza i seguenti indirizzi:

1. (S1, 161.900)
2. (S2, 207.099)
3. (S3, 209.678)
4. (S3, 162.100)
5. (S2, 163.600)
6. (S3, 159.781)

Si chiede di tradurre questi indirizzi in indirizzi fisici oppure dire se originano un fault di pagina.

**SOLUZIONE**

1. (S1, 161.900) corrisponde alla pagina 79 di S1 che non è presente in memoria, quindi genera un fault di pagina
2. (S2, 207.099) corrisponde alla pagina 101 di S2 che è presente in memoria. L'indirizzo fisico si ottiene sommando l'offset 251 all'indirizzo iniziale della pagina fisica dato da 2222*2048: 4.550.907
3. (S3, 209.678) corrisponde alla pagina 102 di S3 che è presente in memoria. L'indirizzo fisico si ottiene sommando l'offset 782 all'indirizzo iniziale della pagina fisica dato da 3987*2048: 8.166.158
4. (S3, 162.100) corrisponde alla pagina 79 di S3 che è presente in memoria. L'indirizzo fisico si ottiene sommando l'offset 308 all'indirizzo iniziale della pagina fisica dato da 3099*2048: 6.347.060
5. (S2, 163.600) corrisponde alla pagina 79 di S2 che è presente in memoria. L'indirizzo fisico si ottiene sommando l'offset 1808 all'indirizzo iniziale della pagina fisica dato da 2619*2048: 5.365.520
6. (S3, 159.781) corrisponde alla pagina 78 di S3 che non è presente in memoria, quindi genera un page fault

**ESERCIZIO B.4 (2 punti)**

In un file system che rappresenta i file mediante la tecnica della lista concatenata, ogni descrittore di file (contenuto nella cartella nella quale è definito il file), contiene il nome del file e il puntatore al primo blocco del file. I blocchi hanno dimensione pari ad 1 Kbyte.
Il file system permette sia l'accesso sequenziale che l'accesso diretto ai file, e offre tra le altre le seguenti chiamate di sistema:

- `fd=open(nomefile)` che restituisce il descrittore `fd` necessario per accedere al file. La open posiziona il puntatore di lettura sul carattere di indice 0 del file.
- `read(fd, &buf, lung)` che legge dal file il cui descrittore è `fd` un numero di caratteri pari a `lung` e li copia nel buffer `buf`.
- `seek(fd, pos)` che posiziona il puntatore di lettura del file `fd` sul carattere di indice `pos` del file.

Il file system contiene (tra gli altri) il file A allocato sui blocchi: 44->51->61->39->38->37.
Dire quanti accessi a blocchi del file sul disco sono richiesti dalle seguenti sequenze di operazioni (da considerare in alternativa):
1. `fd=open("A"); seek(fd,4600); read(fd, &buf, 10);`
2. `fd=open("A"); read(fd, &buf, 3000); seek(fd,0);`

**SOLUZIONE**

1. `fd=open("A"); seek(fd,4600); read(fd, &buf, 10)`
Il primo carattere da leggere (che è il 4600) e l'ultimo carattere da leggere (che è il 4610) risiedono entrambi nel blocco 4, infatti la parte intera di 4600/1024 e di 4610/1024 è 4. Pertanto è necessario accedere ai blocchi 44,51,61,39,38 e sono quindi necessari 5 accessi al disco.

2. `fd=open("A"); read(fd, &buf, 3000); seek(fd,0);`
Il primo carattere da leggere è nel blocco 0 e l'ultimo carattere da leggere, il 2999, risiede nel blocco 2. Pertanto è necessario accedere ai blocchi 44,51,61 e sono quindi necessari 3 accessi al disco. L'ultima chiamata di seek non ha effetto sul numero di accessi.

**ESERCIZIO B.5 (2 punti)**

Data la seguente lista di capabilities (C-list):

Utente A: <File 1: RWX> → <File2: RW -> → <File3: R - -> →<DirA: <RWX> →< DirB: <RWX>
Utente B: <File2: R - X -> → <File3: RW ->
Utente C: <File 1: - - X> → <File2: RWX> → <DirA: <R - X>
Utente D: <File 1: RW -> → <File3: R-- -> →<DirA: <- - X> →< DirB: < R - X >

Convertirla nella corrispondente lista di controllo degli accessi (ACL).

**SOLUZIONE**

File1: <Utente A: RWX> → <Utente C: - - X> → <Utente D: RW ->
File2: <Utente A: RW -> → <Utente B: R - X> → <Utente C: RWX>
File3: <Utente A: R - -> → <Utente B: RW - > → <Utente D: R - ->
DirA: <Utente A: RWX> → <Utente C: R - X> → <Utente D: - - X>
DirB: <Utente A: RWX> → <Utente D: R - X>