UNIVERSITÀ DEGLI STUDI DI BERGAMO

Corso di laurea in ingegneria informatica
Esame di sistemi operativi – appello straordinario – 8 aprile 2008
soluzioni

Cognome_____Nome_____Matricola_____

Esercizio n. 1

Si considerino i seguenti frammenti di programmi:

```
/* programma main.c */
main ( ) {
   int pid;
   char msg_padre [50] = "...";
     ...

   pid = fork ( );
   if (pid != 0) {
      /* codice eseguito da P */
        write (stdout, msg_padre, 50);
      pid = wait (&status);
       exit(0); }
   else { /* codice eseguito dal Q figlio di P  */
      pid = fork ( );
       if (pid == 0) {/* codice eseguito da R figlio di Q */
         execl (   );
         exit(1); }
       else { /* codice eseguito da Q */
         pid = wait (&status);
          exit(1); }
       }
   }/* main.c */
```

```
/* programma muta.c */
main ( ) {
      int fd;
      char l_da_file [1500];
      …
      fd = open ("/utenti/anna/testo", O_RDWR);
      read (fd, l_da_file, 1024);
      exit (2);}
} /* end muta.c */
```

Un processo P esegue il programma **main**, creando un processo figlio Q che, a sua volta, crea un figlio R. (che esegue una mutazione di codice).

- Dopo avere analizzato il codice, **rispondere alla seguente domanda**: qual è il numero massimo di propcessi che possono esistere contemporaneamente? 3 (P, Q, R oppure P, S, R)Motivare in modo esaustivo ma sintetico la risposta.

il numero massimo di processi è 3 (P,Q,R). Infatti P è sincronizzato con Q e Q è sincronizzato con R.

Nella tabella a pagina seguente sono indicati (nella prima colonna) alcuni eventi che si sono verificati durante l'esecuzione dei programmi da parte di P, Q e R; nella seconda colonna è aggiunta un'indicazione supplementare relativa a tali eventi. Si deve completare tale tabella indicando ordinatamente nella terza colonna tutti i moduli del S.O. che vengono eseguiti (completamente o in parte) in seguito all'evento, nella quarta colonna il contesto nel quale ciascun modulo è eseguito e, nelle ultime tre colonne, lo stato dei processi P, Q, e R dopo che tutti i moduli hanno svolto la funzione e si ritorna al funzionamento in modo U.

Avvertenze per il riempimento della tabella:

1. non esistono altri processi nel sistema
2. per la lettura e scrittura su file:
   a) la dimensione di un blocco tresferito in DMA da o su file è di 512 byte
   b) le operazioni di lettura e scrittura su file accedono sempre a disco, cioè è sempre necessario effettuare trasferimenti in DMA
   c) l'interrupt di fine DMA è associato al trasferimento del singolo blocco di file (evento DMAin per lettura di un blocco da file, e evento DMAout per scrittura di un blocco su file)
3. il buffer del driver di standard output ha dimensione di 40 caratteri
4. la notazione 1 interrupt (2, 3 interrupt) indica che si sono verificati 1 (2 o 3) interrupt; nella risposta fare riferimento all'ultimo di tali interrupt
5. la chiamata di sistema wait invoca sleep_on su un evento opportuno
6. la terminazione di un processo (exit) invoca wakeup per risvegliare il processo padre eventualmente in attesa
7. se un processo viene risvegliato da wakeup e non ci sono altri processi pronti o in esecuzione, il processo viene immediatamente lanciato in esecuzione (in questo caso wakeup invoca change)
8. la chiamata di sistema sleep () sospende l'esecuzione del processo che la invoca per un numero di secondi specificato dal parametro. Quindi sleep invoca sleep_on su un evento opportuno e la gestione del tempo trascorso viene eseguita dalla routine di risposta all'interrupt da orologio
9. se viene invocata change e nessun processo è pronto, change non lancia in esecuzione alcun processo e il contesto è **non specificato** – **n.s.**
10. se, al momento di un cambiamento di contesto, più di un processo è pronto all'esecuzione viene scelto quello di maggior priorità. I processi considerati hanno priorità: P>Q>R
11. indicare i moduli utilizzando la notazione seguente:

Notazione abbreviata	Modulo (o frammento di modulo) di sistema
G_SVC	Gestore SVC
R_Int(Disp)	Routine di interrupt; Disp può valere CK=orologio, Sout=Standard output, Sin=Standard input, DMAin=disco_in_lettura, DMAout=disco_in_scrittura
<nome routine di sistema>	puo' essere: fork, write, read, wait, exit, open, sleep, Preempt, Change
Sleep_on(E_n)	Sleep_on. Per indicare l'evento su cui viene sospeso il processo usare convenzionalmente E_1, E_2, E_3, ...
Wakeup(E_n)	Wakeup. E_n indica l'evento, come in Sleep_on
Codice Utente	nessun modulo di sistema, il processo esegue codice utente

Evento (preceduto dal processo nel cui contesto l'evento si verifica)	**Informazioni aggiuntive**	**Moduli eseguiti per gestire l'evento**	**Processo/i nel cui contesto è eseguito ogni modulo**	**Stato dei processi dopo la gestione dell'evento**		
				P	**Q**	**R**
P: fork	P ha esaurito il suo quanto di tempo (n.b. la fork è stata eseguita)	*G_SVC* *fork* *Preempt* *Change* *fork*	*P* *P* *P* *P-Q* *Q*	*Pronto*	*EsecU*	*Non esiste*
Q: fork	Q non ha esaurito il suo quanto di tempo (n.b. la fork è stata eseguita)	*G_SVC* *fork*	*Q* *Q*	*Pronto*	*EsecU*	*pronto*
Q: wait		*G_SVC* *wait* *Sleep_on (E1)* *change* *fork*	*Q* *Q* *Q* *Q-P* *P*	*EsecU*	*Attesa(E1)*	*pronto*
P: write	Write ha trasferito 40 caratteri nel buffer del driver	*G_SVC* *write* *Sleep_on(E2)* *Chang* *fork*	*P* *P* *P* *P-R* *R*	*Attesa(E2)*	*Attesa(E1)*	*EsecU*
R:execl	la execl va a buon fine	*G_SVC* *execl*	*R* *R*	*Attesa(E2)*	*Attesa(E1)*	*EsecU*
R: 40 interrupt da standard output	L'ultimo è relativo all'ultimo carattere presente nel buffer del driver	*R_int(Sout)* *Wake_up(E2)*	*R* *R*	*Pronto*	*Attesa(E1)*	*EsecU*
R: interrupt da orologio	R ha esaurito il quanto di tempo In questo evento viene anche gestito il trasferimento degli ultimi 10 caratteri nel buffer del driver	*R_int(CK)* *Preempt* *Change* *Sleep_on(E2)* *Write* *Sleep_on(E3)* *Change* *preempt*	*R* *R* *R-P* *P* *P* *P* *P-R* *R*	*Attesa(E3)*	*Attesa(E1)*	*EsecU*
R:open	la open va a buon fine e il quanto di tempo di R è scaduto	*G_SVC* *open* *preempt*	*R* *R* *R*	*Attesa(E3)*	*Attesa(E1)*	*EsecU*

Evento (preceduto dal processo nel cui contesto l'evento si verifica)	**Informazioni aggiuntive**	**Moduli eseguiti per gestire l'evento**	**Processo/i nel cui contesto è eseguito ogni modulo**	**Stato dei processi dopo la gestione dell'evento**		
R: 10 interrupt da standard output	L'ultimo è relativo all'ultimo carattere presente nel buffer del driver	*R_int(Sout)* *Wake_up(E3)*	*R* *R*	*pronto*	*Attesa(E1)*	*EsecU*
R: read	read ha inizializzato il DMA in lettura	*G_SVC* *read* *Sleep_on(E4)* *Change* *Sleep_on(E3)* *write*	*R* *R* *R* *R-P* *P* *P*	*EsecU*	*Attesa(E1)*	*Attesa(E4)*
P: wait		*G_SVC* *wait* *sleep_on(E5)* *change*	*P* *P* *P* *P-n.s.*	*Attesa(E5)*	*Attesa(E1)*	*Attesa(E4)*
n.s.: 2 interrupt da disco (lettura)	L'ultimo è relativo all'ultimo blocco da trasferire	*R_int(DMAin)* *Wake_up(E4)* *Change* *Sleep_on(E4)* *read*	*n.s.* *n.s.* *n.s.- R* *R* *R*	*Attesa(E5)*	*Attesa(E1)*	*EsecU*
R: exit		*G_SVC* *Exit* *Wake_up(E1)* *Change* *Sleep_on(E1)* *wait*	*R* *R* *R* *R-Q* *Q* *Q*	*Attesa(E5)*	*EsecU*	*non esiste*
Q: exit		*G_SVC* *Exit* *Wake_up(E5)* *Change* *Sleep_on(E5)* *wait*	*R* *R* *R* *R-Q* *Q* *Q*	*EsecU*	*non esiste*	*non esiste*
P: exit		*G_SVC* *Exit*	*P* *P*	*non esiste*	*non esiste*	*non esiste*

Esercizio n. 2

Un sistema dotato di memoria virtuale con paginazione e segmentazione tipo UNIX è caratterizzato dai seguenti parametri: l'indirizzo logico è di 14 bit; l'indirizzo fisico è di 14 bit. La dimensione delle pagine è di 1024 byte.

- Definire la struttura dell'indirizzo logico e di quello fisico indicando la lunghezza dei campi che li costituiscono:

  NPV: *4* _____ Spiazzamento logico: *10* _____

  NPF: *4* _____ Spiazzamento fisico: *10*_____

Nel sistema saranno attivati i processi P, Q e R, che eseguono i programmi PGP, PGQ e PGR. PGP e PGQ condividono un segmento dati. La dimensione iniziale dei segmenti dei tre programmi è la seguente:

CP:	2K;	DP:	1K;	PP:	1K;	COND:	1K;
CQ:	1K;	DQ:	1K;	PQ:	4K;	COND:	0K;
CR:	2K;	DR:	1K;	PR:	1K;	COND:	1K;

La dimensione complessiva di ognuno dei 3 processi è di 16K e quindi il segmento pila di ogni processo inizia all'indirizzo corrispondente agli 16K; nel processo P il segmento COND è allocato lasciando 3 pagine libere dopo il segmento dati (per permettere una crescita dello Heap, servizio BRK) nel processo R COND è allocato lasciando 1 pagina libere dopo DR.

- Inserire in tabella 1a il significato delle varie pagine di memoria logica (notazione: CP0, CP1, DP0, PP0,...CQ0, ..., COND0,...).

Indirizzo di pagina virtuale	**Processo P**	**Processo Q**	**Processo R**
0	*CP0*	*CQ0*	*CR0*
1	*CP1*	*DQ0*	*CR1*
2	*DP0*		*DR0*
3			
4			*COND0*
5			
6	*COND0*		
7			
8			
9			
A			
B			
C		*PQ3*	
D	*PP2*	*PQ2*	
E	*PP1*	*PQ1*	
F	*PP0*	*PQ0*	*PR0*

1A) Struttura della Memoria Logica

Indirizzo fisico	**Pagine allocate t_0**	**Indirizzo fisico**	**Pagine allocate t_1**
0	**Occupata**	0	**Occupata**
1	*CP0*	1	*PP2*
2	*CP1*	2	
3	*DP0*	3	*DP0*
4	*COND*	4	*COND*
5	*PP0*	5	*PP0*
6	*CR0*	6	*PP1*
7	*CR1*	7	*Occupata*
8	*DR0*	8	*Occupata*
9	*PR0*	9	
A		A	*CQ0*
B		B	*DQ0*
C		C	*PQ0*
D		D	*PQ1*
E		E	*PQ2*
F		F	*PQ3*

1B) Memoria Fisica agli istanti t_0 e t_1

- Indicare quanto spazio (in pagine) è disponibile per la crescita dello stack nei processi

  P: *8*_____ R: *10* _____

Ad un certo istante t_0 sono terminati i seguenti eventi:

1. allocazione della pagina fisica di indirizzo 0 a un processo diverso da P,Q,R
2. lancio di P (fork di P ed exec di PGP)
3. lancio di R (fork di R ed exec di PGR)

- Sapendo che il numero di pagine residenti **R** è 6, che viene utilizzato l'algoritmo LRU e che le pagine meno utilizzate in ogni processo sono la prima pagina del segmento codice, quindi la prima pagina del segmento dati (le prime caricate sono le meno utilizzate) e ipotizzando che l'allocazione delle pagine virtuali nelle pagine fisiche sia avvenuta in sequenza senza buchi a partire dalla pagina fisica 0, indicare, completando tabella 1B (sinistra), l'allocazione fisica delle pagine dei tre processi all'istante t_0. Si indica con **occupata** una pagina utilizzata da altro processo.

Ad un certo istante $t_1 > t_0$ sono terminati i seguenti eventi:

4. lancio di Q (fork di Q ed exec di PGQ)
5. terminazione del processo R
6. allocazione delle pagine di indirizzo fisico 7 e 8 a un processo diverso da P e Q.
7. crescita della pila di P di 2 pagine

- Completare la tabella 1B (destra) nelle medesime ipotesi delineate nel precedente punto (d) e assumendo che, dovendo utilizzare una pagina fisica libera, venga sempre utilizzata la pagina fisica libera avente indirizzo di pagina minore.

- Indicare il contenuto della tabella delle pagine della MMU all'istante t_1 completando la seguente tabella (usare P, Q e R come pid dei corrispondenti processi, oppure NS se nella corrispondente riga della tabella non è allocato alcun processo e quindi è non significativa, **-** se la pagina è occupata da un processo diverso da P,Q e R)); ipotizzare che le righe della tabella siano state allocate ordinatamente man mano che venivano allocate le pagine di memoria virtuale e che gli eventi di cui al punto (e) influenzanti la MMU abbiano utilizzato le righe lasciate libere e che **se richiesta una nuova riga si utilizzi la prima riga libera**. Indicare anche il valore assunto dal bit di validità di pagina (1 significa che la pagina è caricata).

PID	Num. Pag. Virt.	Num. Pag. Fis.	Bit Validità
-	-	*0*	*1*
P	*D (PP2)*	*1*	*1*
P	*1 (CP1)*	*2*	*1*
P	*2 (DP0)*	*3*	*1*
P	*6 (COND)*	*4*	*1*
P	*F (PP0)*	*5*	*1*
-	-	*7*	*1*
-	-	*8*	*1*
P	*E (PP1)*	*6*	*1*
NS	*NS (COND)*	*NS*	*0*
NS	*NS (PR0)*	*NS*	*0*
Q	*0 (CQ0)*	*A*	*1*
Q	*1 (DQ0)*	*B*	*1*
Q	*F (PQ0)*	*C*	*1*
Q	*E (PQ 1)*	*D*	*1*
Q	*D (PQ2)*	*E*	*1*
Q	*C (PQ3)*	*F*	*1*
NS	-	-	*0*

Esercizio n. 3

Definire i concetti di **deadlock** e **starvation**, sottolineandone la differenza, e illustrare un caso in cui un rimedio contro il deadlock causa starvation.

Risposta:

Deadlock, o stallo, è una situazione in cui due (o più) processi cercano di accedere a risorse condivise ad uso esclusivo in maniera tale da creare un'attesa circolare.
Ad esempio, siano A e B due risorse a cui processi P1 e P2 cercano di accedere in maniera esclusiva. Se le richieste di accesso avvengono nel seguente ordine:

1. P1: richiesta accesso ad A
2. P2: richiesta accesso a B
3. P1: richiesta accesso a B
4. P2: richiesta accesso ad A

allora si ha un deadlock in quanto con la richiesta 3 P1 rimane in attesa di P2, mentre subito dopo con la richiesta 4 P2 si pone in attesa di P1.

Si ha una situazione di starvation per un processo quando ad esso viene negato in continuazione l'accesso alle risorse necessarie al suo completamento.
La starvation può essere vista come un caso più generale di deadlock, in quanto si tratta comunque di una situazione in cui i processi non riescono a terminare, ma può anche riguardare un solo processo. Ci sono meno condizioni necessarie per una starvation che per un deadlock: ad esempio si può avere starvation anche senza attesa circolare (come nel caso di singolo processo lasciato senza risorse).

Per evitare il deadlock, si può assegnare ai processi un livello di priorità, e ogni volta che due processi entrano in competizione per la stessa risorsa, il sistema operativo può decidere a favore del processo con priorità maggiore. In questo modo non si ha mai la condizione di attesa circolare e quindi il deadlock è evitato. Il rischio di starvation permane, in quanto un processo a priorità particolarmente bassa potrebbe vedersi negato l'accesso alle risorse necessarie a causa di nuovi processi a priorità superiore che continuamente fanno ingresso nel sistema.