UNIVERSITÀ DEGLI STUDI DI BERGAMO

Corso di laurea in ingegneria informatica
Esame di sistemi operativi – 1 luglio 2008

Cognome_____SOLUZIONI_____Nome_____Matricola_____

1.
Si consideri il seguente programma:

```
01 int main() {
02  int i, j, k, pid, status;
03  i = 3;
04  k = 5;
05  while (i > 0) {
06    i--;
07    pid = fork();
08    if (pid == 0) {
09      j = k+6;
10      if (i == 2) {
11        k = 5;
12      exit(0);
13      }
14      else {
15      j = -1;
16      exit(1);
17      }
18    }
19    k = i+5;
20    wait(&status);
21  }
22  return j;
23 }
```

Completare le tabelle 1 e 2 sotto riportate, indicando, negli istanti di tempo specificati, il valore delle variabili i, k e pid. Attenzione:

1. nel caso in cui al momento indicato la variabile non esista (in quanto non esiste il processo) riportare NE;
2. quando non si può dire con certezza se la variabile esista e/o quale ne sia il valore, riportare U;
3. si suppone che tutte le chiamate ai servizi di sistema abbiano sempre successo e che il sistema operativo assegni ai processi creati dei pid consecutivi a partire da 560;
4. la frase “prima dell’istruzione X” si riferisce all’istante in cui si inizia l’esecuzione dell’istruzione X stessa.

N.B. Le tabelle sottoriportate si riferiscono solo ad alcuni dei processi generati dal programma.

Tabella 1. Valore delle variabili nel processo padre.	i	j	k	pid
Prima dell’istruzione 7 (la prima volta in cui viene eseguita)	2	U	5	U
Prima dell’istruzione 12	2	U	U	560
Prima dell’istruzione 22	0	U	5	562

Tabella 2. Valore delle variabili nel processo 560.	i	j	k	pid
Prima dell’istruzione 7 (la prima volta in cui viene eseguita)	NE	NE	NE	NE
Prima dell’istruzione 12	2	11	5	0
Prima dell’istruzione 22	NE	NE	NE	NE

2.

Si consideri il seguente programma:

```
main()
{  int pid, pid1, d;
   char c[65];
   ...
   read(stdin, &d, 1);
   pid=fork();
   if (pid == 0)
     {  /* codice eseguito da Q figlio di P */
        …
        read(stdin, &d, 1);
        exit(1);
     }
    else
     {pid1=fork();
      if (pid1 == 0)
        {  /* codice eseguito da R figlio di P */
           …
           exit(2);
        }
       else
        { /* codice eseguito dal padre P  */

          write(stdout, c, 65);
          pid = wait(&status);
          exit(0);
        }
     }

} /* end main */
```

Un processo P esegue tale programma, creando un processo figlio Q e un processo figlio R.

Nella tabella a pagina seguente sono indicati (nella prima colonna) alcuni eventi che si sono verificati durante l'esecuzione del programma 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:

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
<nome routine di sistema>	puo' essere: fork, write, read, wait, exit, 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

- non esistono altri processi nel sistema
- il buffer del driver di standard output ha dimensione di 70 caratteri
- la notazione 35 interrupt (15 interrupt) indica che si sono verificati 35 (15) interrupt. Nella risposta fare riferimento all'ultimo di tali interrupt
- la terminazione di un processo (exit) invoca wakeup per risvegliare il processo padre eventualmente in attesa. 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)

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: read	Lo standard input è pronto; P non ha ancora esaurito il suo quanto di tempo	G_SVC read	P P	Esec U	non esiste	non esiste
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*	*Esec U*	*non esiste*
Q: read	Lo standard input non è pronto	*G_SVC* *read* *Sleep-on(E1)* *Change* *fork*	*Q* *Q* *Q* *Q-P* *P*	*Esec U*	*Attesa*	*non esiste*
P: fork	P non ha ancora esaurito il suo quanto di tempo	*G_SVC* *fork*	*P* *P*	*Esec U*	*Attesa*	*pronto*
P: write	write ha trasferito i 65 caratteri nel buffer	*G_SVC* *write* *Sleep_on(E2)* *Change* *Preempt*	*P* *P* *P* *P-R* *R*	*Attesa*	*Attesa*	*Esec U*
R: interruput da standard input	Lo standard input è pronto	*R_int(Sin)* *Wakeup(E1)*	*R* *R*	*Attesa*	*pronto*	*Esec U*
R: 50 interrupt da standard output	R ha esaurito il suo quanto di tempo	*R_int(Sout)* *Preempt* *Change* *Sleep-on(E1)* *read*	*R* *R* *R-Q* *Q* *Q*	*Attesa*	*Esec U*	*Pronto*
Q: exit		*G_SVC* *exit* *Change*	*Q* *Q* *Q-R*	*Attesa*	*non esiste*	*Esec U*
R: 15 interrupt da standard output	l'ultimo è relativo all'ultimo carattere da trasferire; R ha esaurito il suo quanto di tempo	*R_int(Sout)* *Wakeup(E2)* *Preempt* *Change* *Sleep-on(E2)* *write*	*R* *R* *R* *R-P* *P* *P*	*Esec U*	*non esiste*	*Pronto*
P: wait		*G_SVC* *wait*	*P* *P*	*Attesa*	*non esiste*	*Esec U*
P: exit		*G_SVC* *exit*	*P* *P*	*non esiste*	*non esiste*	*Esec U*
R: exit		*G_SVC* *exit*	*R* *R*	*non esiste*	*non esiste*	*non esiste*

3.

Illustrare gli i-node, spiegandone in maniera dettagliata lo scopo, la struttura, e il loro funzionamento.

Cfr. capp. 11 e 12 del libro di testo.