UNIVERSITÀ DEGLI STUDI DI BERGAMO

Corso di laurea in ingegneria informatica
Esame di sistemi operativi – 15 gennaio 2008

1.
Si considerino il seguente programma C:

```
1.  int main ( ) {
2.      int pid = -1;
3.      int cont = 1;
4.      while (cont <= 2) {
5.          pid = fork ( );
6.          if (pid == 0) {
                /* qui arriva(no) proc.: F1 e F2 _____________________ */
7.              if (cont == 1) {
8.                  pid = fork ( );
                    /* qui arriva(no) proc.: F1 e N _________________ */
9.                  if (pid != 0) {
10.                     pid = wait (NULL);
11.                 } /* if */
12.                 exit (0);
13.             } else {
                    /* qui arriva(no) proc.: F2 _____________________ */
14.                 printf ("%d\n", pid);
15.                 exit (0);
16.             } /* if */
17.         } else {
                /* qui arriva(no) proc.: P ______________________ */
18.             if (cont == 2) {
19.                 waitpid (pid, NULL);
20.             } else {
21.                 printf ("%d\n", pid);
22.             } /* if */
23.         } /* if */
24.         cont++;
25.     } /* while */
26.     exit (0);
27. } /* main */
```

Mettendo in esecuzione il programma "main", si avranno quattro processi: il padre P (all'attivazione di "main"), due figli F1 e F2 e un nipote N (il lettore stabilisca da sé di chi il nipote N sia figlio). Il figlio F1 è creato prima di F2. Il pid del processo P è 500, poi il S.O. prosegue assegnando pid consecutivi ai processi via via creati.

Ciò detto, si chiede di svolgere i punti seguenti, che è bene affrontare in successione:

1. Analizzando il codice C sopra riportato **si indichi** (indichino), nei punti dove sono già previsti i commenti "qui arriva(no) proc. ..", quale (quali) dei quattro processi P, F1, F2 e N arrivi (arrivino) in quel punto (si completino i commenti elencando i nomi dei processi).

2. Ciascuno dei due processi P e F2 (non si considerino F1 e N) esegue, prima o poi, una e una sola stampa a terminale mediante la funzione "printf", scrivendo il valore correntemente presente nella variabile "pid". (se più risposte sono possibili le si elenchino tutte) Simulando quanto basta l'andamento dei processi, **si scrivano** sotto tali valori stampati.

   - P pid = *501* _____
   - F2 pid = *0*

3. **Si compilino** le quattro tabelle A, B, C e D riportate sotto (per F1 F2 e N rispettivamente - P non è considerato), indicando i valori della variabile "pid". Attenzione alle convenzioni seguenti:
   - quando la variabile non esiste (in quanto non esiste il processo cui essa appartiene) si scriva NE
   - quando non si può dire con certezza se la variabile esista o quale ne sia il valore si scriva U
   - la frase "subito dopo l'istruzione X" indica l'istante immediatamente successivo all'istruzione X stessa
   - se il processo non ha mai avuto figli la chiamata di sistema "wait" non blocca e restituisce –1
   - e infine si suppone che tutte le chiamate ai servizi di sistema abbiano sempre successo

Tabella A. Valore delle variabili in **F1**	**pid**
subito dopo l'istruzione 10 *(a)*	*U*
subito dopo l'istruzione 21 *(b)*	*501*

Tabella B. Valore delle variabili in **F2**	**pid**
subito dopo l'istruzione 10 *(c)*	*U*
subito dopo l'istruzione 19 *(d)*	*NE*

Tabella C. Valore delle variabili in **N**	**pid**
subito dopo l'istruzione 10 *(e)*	*NE*
subito dopo l'istruzione 21 *(f)*	*U*

2.
Si consideri il frammento seguente di programma (gli #include necessari sono omessi):

```
/* programma main.c */
main ( ) {
    int pid1, pid2;

    char ACKMSG [40]= ".......";
    char c;
    ...
    pid1 = fork ( );
    if (pid1 == 0) {
        /* codice eseguito da Q, figlio di P          */
        read (stdin, &c, 1);
        if (c == 'q') {
            pid2 = fork ( );
            if (pid2 == 0) {
              /* codice eseguito da R, figlio di Q */
              exit (2);
            } else { /* codice eseguito da Q  */
              write (stdout, ACKMSG, 40);
              pid2 = waitpid (pid2, &status, 0);
              exit (1);
              } /* end if */
        } else { /* codice eseguito da Q */
            exit (-1);
            } /* end if */
    } else { /* codice eseguito da P */
      wait (pid1);
      write (stdout, status_c, 5);
      exit (0);
    } /* end if */
} /* main.c */
```

Un processo **P** crea un figlio **Q** che poi un figlio **R**.

Nella tabella a pagina seguente sono indicati (nella prima colonna) alcuni eventi verificatisi 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 completi** 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 è tornati al funzionamento in modo U.

Avvertenze per il riempimento della tabella:

1. non esistono altri processi nel sistema
2. il buffer dello standard output è di 30 caratteri
3. la notazione “1 interrupt” (2 o 3 interrupt) indica che si sono verificate 1 (2 o 3) interruzioni; nella risposta si faccia riferimento all'ultima di tali interruzioni
4. la chiamata di sistema “wait / waitpid” invoca “Sleep_on” su un evento opportuno
5. la terminazione di un processo (“exit”) invoca “Wake_up” per risvegliare il processo padre eventualmente in attesa
6. se un processo viene risvegliato da “Wake_up” e non ci sono altri processi pronti o in esecuzione, il processo viene immediatamente lanciato in esecuzione (in questo caso “Wake_up” invoca “Change”)
7. la chiamata di sistema “sleep (arg)” sospende l’esecuzione del processo che la invoca per un numero di secondi specificato dal parametro; pertanto “sleep” invoca “Sleep_on” su un evento opportuno e la gestione del tempo trascorso viene eseguita dalla routine di risposta all’interruzione da orologio
8. se viene invocata “Change” e nessun processo è pronto, “Change” non lancia in esecuzione alcun processo e il contesto è **non specificato** – **n.s.**
9. ssi indichino i moduli utilizzando la notazione seguente:

Notazione abbreviata	**Modulo (o frammento di modulo) di sistema**
G_SVC	Gestore SVC (chiamata a supervisore, supervisor call)
R_Int (Disp)	Routine di interruzione; Disp può valere: CK = orologio, RETE_acc = scheda rete per accept, S_out = Standard output, S_in = Standard input, DMA_in = disco_in_lettura, DMA_out = disco_in_scrittura
<nome routine di sistema>	può essere: fork, write, read, wait, exit, open, sleep, exec, Preempt, Change, accept ecc., ecc., ...
Sleep_on (E_n)	Sleep_on: per indicare l’evento su cui viene sospeso il processo si scriva convenzionalmente E_1, E_2, E_3, ..., E_n, ...
Wake_up (E_n)	Wake_up: il simbolo E_n indica l’evento, come in Sleep_on
Codice Utente	nessun modulo di sistema, il processo esegue codice utente

Evento (è 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. La fork è stata eseguita	*G_SVC* *fork* *Preempt* *Change* *fork*	*P* *P* *P* *P-Q* *Q*	*pronto*	*esecu*	*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* (E1)	*non esiste*
P: wait		*G_SVC* *wait* *Sleep_on* (E2) *Change*	*P* *P* *P* *P –n.s.*	*attesa* (E2)	*attesa* (E1)	*non esiste*
n.s.: interrupt da Standard input		*R_Int* (Std_in) *Wake_up* (E1) *Change* *Sleep_on* (E1) *read*	*n.s.* *n.s.* *n.s. – Q* *Q* *Q*	*attesa* (E2)	*esec U*	*non esiste*
Q: fork	Q NON ha esaurito il suo quanto di tempo	*G_SVC* *fork*	*Q* *Q*	*attesa* (E2)	*esec U*	*pronto*
Q: write	write ha trasferito 30 caratteri nel buffer del driver	*G_SVC* *write* *Sleep_on* (E3) *Change* *fork*	*Q* *Q* *Q* *Q-R* *R*	*attesa* (E2)	*attesa* (E3)	*esecU*

R:codice utente			*R*	*attesa* (E2)	*attesa* (E3)	*esecU*
R: 30 interrupt da standard output	L'ultimo è relativo all'ultimo carattere presente nel buffer. Vengono trasferiti nel buffer gli ultimi 10 caratteri da scrivere	*R_Int* (Std_out) *Wake_up* (E3) *Change* *Sleep_on* (E3) *write* *Sleep_on (E4)* *change* *R_Int* (Std_out)	*R* *R* *R-Q* *Q* *Q* *Q* *Q-R* *R*	*attesa* (E2)	*attesa* (E4)	*esecU*
R: exit		*G_SVC* *exit* *change*	*R* *R* *R-n.s.*	*attesa* (E2)	*attesa* (E4)	*non esiste*
n.s.: 10 interrupt da standard output	L'ultimo è relativo all'ultimo carattere presente nel buffer.	*R_Int* (Std_out) *Wake_up* (E4) *Change* *Sleep_on* (E4) *write*	*n.s.* *n.s.* *n.s.-Q* *Q* *Q*	*attesa* (E2)	*esecU*	*non esiste*
Q: waitpid		*G_SVC* *waitpid*	*Q* *Q*	*attesa* (E2)	*esecU*	*non esiste*

(fine esercizio 2)

3.
Illustrare le 4 condizioni necessarie perché si verifichi un deadlock e proporre delle soluzioni per prevenire ciascuna di esse.

Cfr. capitolo 8 del libro di testo.