Esercizio #1 – *gestione dei processi, fork(), wait()*

Si supponga di avere a disposizione un file di testo `pippo.text` contenente una stringa di 60 caratteri, come segue

111111111122222222223333333333444444444455555555556666666666

Si vuole sincronizzare un processo P1 (padre) con il corrispondente processo P2 (figlio) come segue: il padre legge 20 bytes dal file e successivamente modifica i primi 10 caratteri, il figlio legge i primi 20 caratteri dal file, il padre ora legge ancora 10 caratteri. Si utilzzando un solo descrittore di file senza dupicarlo. Utilizzare una sincronizzazione esplicita basata su `fork()` e `wait()`. Cosa se ne conclude circa l'utilizzo della sincronizzazione tramite `wait()`? La sincronizzazione tramite `wait()` è flessibile? È facilmente modificabile? È totalmente sicura? A che byte punta il descrittore del file al termine delle operazioni richieste?

Esercizio #2 – *gestione dei processi, valore di ritorno del figlio*

Si vuole sviluppare una semplice toy-application in cui l'utente specifica da terminale il valore che il figlio dvorà restituire. Il processo padre, una volta che il figlio ha terminato, deve leggere il valore di ritorno specificato e stamparlo a video. Documentarsi circa l'utilizzo del valore di ritorno nelle man pages della `wait()`.

Esercizio #3 – *gestione dei processi, segnali, terminazione processi*

Proporre una soluzione alternativa a quella data per l'Esercizio #1, in modo tale da utilizzare i segnali per la comunicazione dell'avvenuta terminazione da parte del padre della scrittura su file. Cosa cambia rispetto alla soluzione proposta nell'Esercizio #1? Cosa cambia dal punto di vista della comunicazione? Riportare il diagramma temporale dell'evoluzione dei due processi. Cosa succede se il processo figlio termina (inaspettatamente) prima di inviare il segnale SIGUSR1 al padre? Cosa succede al processo figlio se il padre termina prima del dovuto?

Esercizio #4 – *gestione dei processi, segnali (svolto durante le esercitazioni)*

Un processo P1 crea un processo P2; il processo P2 rimane in attesa sui segnali SIGUSR1 e SIGUSR2 per tutta la durata del suo ciclo di vita, mentre da terminale l'utente invia una serie di segnali SIGUSR1 eun unico SIGUSR2 al processo P2. Al termine della sua esecuzione (definito dall'invio del segnale SIGUSR2) il processo P2 stampa a video il numero di interrupt software ricevuti tramite SIGUSR1, i.e. quante volte il segnale è stato effettivamente recapitato a P2 il segnale SIGUSR1.
Si supponga di avere la seguente evoluzione temporale, in cui P1 crea P2 al tempo t1; l'utente invia SIGUSR1 a P2 al tempo t2 e un segnale di SIGSTOP a P2 al tempo t3 (non importa il valore assoluto dei tempi, ma il loro ordinamento all'interno dell'intervallo considerato); due segnali SIGUSR1 sono inviati a P2 tra t4 e t5, mentre SIGCONT è inviato al tempo t6. P1 rimane in attesa della terminazione di P2 durante tutta l'esecuzione. Cosa stampa a video P2 a t=t7 se l'utente invia SIGUSR2? Si rifletta sul perchè di questo risultato. Proporre un'implementazione dello scenario dato.

t	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**...**
P1	Fork( )	Wait()						
Utente		SIGUSR1 a P2	SIGSTOP a P2	SIGUSR1 a P2	SIGUSR1 a P2	SIGCONT a P2	SIGUSR2 a P2	
P2							Print()	

Esercizio #5 – *gestione dei processi, condivisione dati*

Cosa stampa il seguente programma? Perchè? Ovviamente, rispondere senza compilare il codice...

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <sys/wait.h>
 4 #include <sys/types.h>
 5
 6 int j;
 7
 8 int main()
 9 {
10     pid_t pid;
11     int *k;
12
13     /* Initialization */
14     k = (int *)malloc(sizeof(int));
15     j = 1;
16     *k = 2;
17
18     /* Create child process */
19     pid = fork();
20     if(pid == 0)
21     {
22         j = 45;
23         (*k)++;
24         printf("CHILD : \t"
25             "j = %d\t"
26             "k = %d\n", j, *k);
27
28         exit(0);
29     }
30     else
31     {
32         wait(NULL);
33         printf("PARENT :\t"
34             "j = %d\t"
35             "k = %d\n", j, *k);
36
37         free(k);
38         return 0;
39     }
40 }
```