Esercitazione di Venerdì 20 Aprile 2018

1. (padreFiglio.c) Scrivere un programma C che riporti su standard output la stringa "Hello, I'm the father" e crei un processo figlio che riporti su standard output la stringa "Hello, I'm the child": ricordarsi che il padre deve controllare il valore di ritorno della fork per assicurarsi che la creazione sia andata a buon fine. Entrambi i processi, inoltre, stampano a video il valore ritornato dalla fork(). Ricordarsi infine che è buona norma che un padre aspetti sempre la terminazione dei figli che crea.

```
soELab@Lica02:~/processi/secondaEsercitazione$ cat padreFiglio.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char **argv)
{
        int pid; /* per valore di ritorno della fork */

        if ((pid = fork()) < 0)
        {
                printf("ERRORE nella fork\n");
                exit(1);
        }
        if (pid == 0)
        {
                 /* child */
                printf("Ciao, sono il processo figlio, il valore ritornato
dalla fork e' %d\n", pid);
                exit(0);
        }
        /* father */
        printf("Ciao, sono il processo padre, il valore ritornato dalla
fork e' %d\n", pid);
        wait((int *) 0);  /* il padre aspetta il figlio senza pero' ricavare
informazioni */
        exit(0);
}
```

2. (padreFiglioConStatus.c) Partendo dall'esercizio precedente, il padre deve usare nella wait il parametro di ingresso per recuperare il valore tornato dal figlio con la exit e deve riportarlo su standard output: decidere che valore fare tornare al figlio.

```
soELab@Lica02:~/processi/secondaEsercitazione$ cat padreFiglioConStatus.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char **argv)
{
        int pid; /* per valore di ritorno della fork */
        int pidFiglio; /* per valore di ritorno della wait */
        int status; /* per usarlo nella wait */
        int exit_s; /* per filtrare valore di uscita del figlio  */

        if ((pid = fork()) < 0)
        {
```

```
                printf("ERRORE nella fork\n");
                exit(1);
        }
        if (pid == 0)
        {
                 /* child */
                printf("Ciao, sono il processo figlio, il valore ritornato
dalla fork e' %d\n", pid);
                exit(0);
        }
        /* father */
        printf("Ciao, sono il processo padre, il valore ritornato dalla
fork e' %d\n", pid);
        /* il padre aspetta il figlio */
        if ((pidFiglio=wait(&status)) < 0)
        {
                printf("ERRORE nella wait %d\n", pidFiglio);
                exit(2);
        }
        if (pid == pidFiglio) printf("Terminato figlio con PID = %d\n",
pidFiglio);
        else exit(3);

        if ((status & 0xFF) != 0)
                printf("Figlio terminato in modo involontario\n");
        else
        {
                exit_s = status >> 8;
                /* selezione degli 8 bit piu' significativi */
                exit_s &= 0xFF;
                printf("Per il figlio %d lo stato di EXIT e` %d\n",
pidFiglio, exit_s);
        }
        exit(0);
}
```

3. (padreFigliMultipli.c) Scrivere un programma che generi un numero di figli uguale al numero N passato come argomento (strettamente maggiore di 0 e minore di 255). Ogni figlio deve riportare su standard output il proprio indice e lo deve anche ritornare al padre. Il processo padre deve attendere la terminazione di tutti i processi figli riportando su standard output i valori ritornati da ogni processo figlio insieme con il pid del processo terminato.

```
soELab@Lica02:~/processi/secondaEsercitazione$ cat padreFigliMultipli.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>


int main (int argc, char **argv)
{
int N;                  /* numero di figli */
int pid;                /* pid per fork */
int i;                  /* indice */
int pidFiglio, status, ritorno; /* per valore di ritorno figli */

/* controllo sul numero di parametri: essattamente un numero */
```

```
if (argc != 2)
{
        printf("Errore numero di parametri: %s vuole un numero\n",
argv[0]);
        exit(1);
}

/* convertiamo il parametro in numero */
N=atoi(argv[1]);
if (N <= 0 || N >= 255)
{
        printf("Errore l'unico parametro non e' un numero strettamente
positivo o non e' minore di 255: %d\n", N);
        exit(2);
}

/* creazione figli */
for (i=0;i<N;i++)
{
        if ((pid=fork())<0)
        {
                printf("Errore creazione figlio\n");
                exit(3);
        }
        else if (pid==0)
        { /* codice figlio */
                printf("Sono il figlio %d di indice %d\n", getpid(), i);
                exit(i);
        }
} /* fine for */

/* codice del padre */
/* Il padre aspetta i figli */
for (i=0; i < N; i++)
{
        pidFiglio = wait(&status);
        if (pidFiglio < 0)
        {
                printf("Errore in wait\n");
                exit(4);
        }
        if ((status & 0xFF) != 0)
                printf("Figlio con pid %d terminato in modo anomalo\n",
pidFiglio);
        else
        {
                ritorno=(int)((status >> 8) & 0xFF);
                printf("Il figlio con pid=%d ha ritornato %d\n",
pidFiglio, ritorno);
        }
}
exit(0);
}
```

4. (padreFigliMultipliConSalvataggioPID.c) Partendo dall'esercizio precedente (il valore massimo di N deve essere minore di 155), il padre deve salvare i pid di tutti i figli creati in un array dinamico. Ogni figlio (indice i) deve calcolare in modo random un numero compreso fra 0 e 100+i e lo deve ritornare al padre. Il processo padre deve attendere la terminazione di tutti i processi figli riportando su standard output i valori ritornati da ogni processo figlio insieme con il pid del processo terminato e il numero d'ordine derivante dalla creazione.
   OSSERVAZIONE: per generare numeri random usare
   Chiamata alla funzione di libreria per inizializzare il seme:
   #include <time.h>
   srand(time(NULL));
   Funzione che calcola un numero random compreso fra 0 e n-1:
   #include <stdlib.h>
   int mia_random(int n)
   {
       int casuale;
       casuale = rand() % n;
       return casuale;
   }

```
soELab@Lica02:~/processi/secondaEsercitazione$ cat padreFigliMultipliConSalvataggioPID.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>

int mia_random(int n)
{
int casuale;
casuale = rand() % n;
return casuale;
}

int main (int argc, char **argv)
{
int N;                  /* numero di figli */
int *pid;               /* iarray di pid per fork */
int i, j;               /* indici */
int pidFiglio, status, ritorno; /* per valore di ritorno figli */

/* controllo sul numero di parametri: essattamente un numero */
if (argc != 2)
{
        printf("Errore numero di parametri: %s vuole un numero\n",
argv[0]);
        exit(1);
}

/* convertiamo il parametro in numero */
N=atoi(argv[1]);
if (N <= 0 || N >= 155)
{
        printf("Errore l'unico parametro non e' un numero strettamente
positivo o non e' minore di 155: %d\n", N);
        exit(2);
}
```

```
/* allocazione pid */
if ((pid=(int *)malloc(N*sizeof(int))) == NULL)
{
        printf("Errore allocazione pid\n");
        exit(3);
}

/* creazione figli */
for (i=0;i<N;i++)
{
        if ((pid[i]=fork())<0)
        {
                printf("Errore creazione figlio\n");
                exit(4);
        }
        else if (pid[i]==0)
        { /* codice figlio */
                int r; /* per valore generato random */
                srand(time(NULL));  /*  inizializziamo  il  seme  per  la
generazione random di numeri  */
                printf("Sono il figlio %d di indice %d\n", getpid(), i);
                r=mia_random(100+i);
                exit(r);
        }
} /* fine for */

/* codice del padre */
/* stampa di debugging */
printf("Valore dei pid dei figli creati (per debugging)\n");
for (i=0; i < N; i++)
        printf("pid[%d]=%d\n", i, pid[i]);

/* Il padre aspetta i figli */

for (i=0; i < N; i++)
{
        pidFiglio = wait(&status);
        if (pidFiglio < 0)
        {
                printf("Errore in wait %d\n", pidFiglio);
                exit(5);
        }
        if ((status & 0xFF) != 0)
                printf("Figlio con pid %d terminato in modo anomalo\n",
pidFiglio);
        else
        {
                ritorno=(int)((status >> 8) & 0xFF);
                for (j=0; j < N; j++)
                        if (pidFiglio == pid[j])
                                {
                                        printf("Il  figlio  con  pid=%d  di
indice %d ha ritornato %d\n", pidFiglio, j, ritorno);
                                        break;
                                }
        }
```

```
}
exit(0);
}
```

5. (padreFigliMultipliConConteggioOccorrenze.c) Scrivere un programma che deve prevedere N+1 parametri: i primi N (con N >= 2) devono essere nomi di file e l'ultimo parametro deve essere considerato un singolo carattere Cx (si facciano tutti i controlli del caso). Il padre deve generare un numero di figli uguale al numero di file passati come argomento. Ogni figlio deve leggere dal file associato contando le occorrenze del carattere Cx. Ogni figlio deve ritornare al padre il numero di occorrenze (supposto minore di 255). Il processo padre deve attendere la terminazione di tutti i processi figli riportando su standard output i valori ritornati da ogni processo figlio insieme con il pid del processo terminato.

```
soELab@Lica02:~/processi/secondaEsercitazione$ cat padreFigliMultipliConConteggioOccorrenze.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>

int main (int argc, char **argv)
{
char Cx;                          /* carattere che i figli devono cercare
nel file a loro associato */
int N;                  /* numero di figli */
int pid;                /* pid per fork */
int i;                  /* indice */
int totale=0;           /* serve per calcolare il numero di occorrenze: in
questo caso abbiamo usato un semplice int perche' la specifica dice che si
può supporre minore di 255 */
int fd;                 /* per la open */
char c;                 /* per leggere i caratteri dal file */
int pidFiglio, status, ritorno; /* per valore di ritorno figli */

/* controllo sul numero di parametri: almeno due nomi file e un carattere
*/
if (argc < 4)
{
        printf("Errore numero di parametri: i parametri passati a %s sono
solo %d\n", argv[0], argc);
        exit(1);
}

/* controlliamo che l'ultimo paramentro sia un singolo carattere */
if (strlen( argv[argc-1]) != 1)
{
        printf("Errore ultimo parametro non singolo carattere\n");
        exit(2);
}

/* individuiamo il carattere da cercare */
Cx = argv[argc-1][0];

N=argc-2;
/* creazione figli */
for (i=0;i<N;i++)
```

```
{
        if ((pid=fork())<0)
        {
                printf("Errore creazione figlio\n");
                exit(3);
        }
        else if (pid==0)
        { /* codice figlio */
                printf("Sono il figlio %d di indice %d\n", getpid(), i);
                /* apriamo il file: notare che l'indice che dobbiamo usare
e' i+1 */
                /* in caso di errore decidiamo di ritornare -1 che saa'
interpretato dal padre come 255 e quindi un valore non valido! */
                if ((fd = open(argv[i+1], O_RDONLY)) < 0)
                {
                        printf("Errore: FILE NON ESISTE\n");
                        exit(-1);
                }
                /* leggiamo il file */
                while (read (fd, &c, 1) != 0)
                {          if (c == Cx) totale++;      /* se troviamo il
carattere increntiamo il conteggio */
                }
                printf  ("STAMPA  DI  DEBUGGING:  Il  numero  totale  di
occorrenze del carattere %c nel file %s e' %d\n", Cx, argv[i+1], totale);
                exit(totale);
        }
} /* fine for */

/* codice del padre */
/* Il padre aspetta i figli */
for (i=0; i < N; i++)
{
        pidFiglio = wait(&status);
        if (pidFiglio < 0)
        {
                printf("Errore in wait\n");
                exit(4);
        }
        if ((status & 0xFF) != 0)
                printf("Figlio con pid %d terminato in modo anomalo\n",
pidFiglio);
        else
        {
                ritorno=(int)((status >> 8) & 0xFF);
                printf("Il  figlio  con  pid=%d  ha  ritornato  %d\n",
pidFiglio, ritorno);
        }
}
exit(0);
}
```

NOTA BENE: Per esercitarsi con la generazione di processi, ispirarsi all'ultimo esercizio e sostituire, al posto del conteggio occorrenze, uno degli altri esercizi sui file della precedente esercitazione.