

System call: esercitazione su creazione, terminazione, esecuzione

Roberto De Virgilio

Sistemi operativi - 12 Dicembre 2017

Esercizio I (warm-up)

- ◆ Scrivere **fork_status.c**, un programma in Linguaggio C, per il controllo dei processi:
 - ◆ Il processo **padre**, genererà un processo **figlio**
 - ◆ il **processo figlio** stamperà su output vari messaggi (preceduti da CHILD:) riguardo al fatto di essere un processo figlio, riguardo il suo PID e il PID del processo padre; quindi chiederà da input di digitare un codice di uscita e una volta acquisito terminerà con tale codice
 - ◆ il **processo padre** attenderà 5 secondi per poi stampare su output vari messaggi (preceduti da PARENT:) riguardo al fatto di essere un processo padre, riguardo il suo PID e il PID del processo figlio; quindi stamperà il messaggio “attendo la fine del figlio”, attende la terminazione del figlio e alla fine stampa il codice di terminazione del figlio (si usi la macro **WEXITSTATUS**)

Esercizio 1 (warm-up)

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h> //per la perror()
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h> //per la WEXITSTATUS() macro

int main(void) {
    pid_t pid;
    int uscita;
```

Esercizio I (warm-up)

```
switch(pid=fork()) {  
    case -1:  
        perror("fork"); /* errore */  
        exit(1);          /* il padre esce con errore 1 */  
    case 0:  
        printf(" CHILD: processo figlio !\n");  
        printf(" CHILD: il mio PID : %d\n", getpid());  
        printf(" CHILD: il PID di mio padre: %d\n", getppid());  
        printf(" CHILD: scrivi il valore di uscita (< 100):\n ");  
        scanf(" %d", &uscita);  
        printf(" CHILD: termine !\n");  
        exit(uscita);
```

Esercizio I (warm-up)

```
default:  
    sleep(5); //serve per avere il tempo  
    //di scrivere il valore di exit()  
    printf("PARENT: processo padre !\n");  
    printf("PARENT: il mio PID : %d\n", getpid());  
    printf("PARENT: il PID di mio figlio : %d\n", pid);  
    printf("PARENT: Attendo la fine del figlio ...!\n");  
    wait(&uscita);  
    printf("PARENT: ecco lo stato di uscita del figlio :%d\n",  
          WEXITSTATUS(uscita));  
    printf("PARENT: termine !\n");  
}  
return 0;  
}
```

Esercizio I (warm-up)

- **Output:**

```
$ gcc fork_status -o test  
$ ./test  
  
CHILD: processo figlio !  
  
CHILD: il mio PID : 3032  
  
CHILD: il PID di mio padre: 3031  
  
CHILD: scrivi il valore di uscita (< 100):  
34  
  
CHILD: termino !  
  
PARENT: processo padre !  
  
PARENT: il mio PID : 3031  
  
PARENT: il PID di mio figlio : 3032  
  
PARENT: Attendo la fine del figlio ...  
  
PARENT: ecco lo stato di uscita del figlio :34  
  
PARENT: termino !
```

Esercizio 1

- ◆ Scrivere **stampa_info.c**, un programma in Linguaggio C a supporto di un programma per il controllo dei processi.
- ◆ tale programma prevede un argomento (una stringa) che può essere “*nipote*” o “*figlio*”
- ◆ una volta compilato *stampa_info.c* generando l'eseguibile **stampa_info**, l'esecuzione del programma
 - ◆ nel caso dell'argomento “*nipote*” (quindi *./stampa_info nipote*) stamperà su output “*Sono il nipote del primo padre*”
 - ◆ nel caso dell'argomento “*figlio*” (quindi *./stampa_info figlio*) stamperà su output “*Sono il figlio del secondo padre*”

Esercizio 1

- ◆ Scrivere **nipote.c**, un programma in Linguaggio C, per il controllo dei processi in cui ci siano due processi padre che generino **processi figli**:
- ◆ Il primo processo **padre**, stamperà il messaggio "**sono il primo processo padre**" e poi genererà un processo **figlio**
- ◆ il **processo figlio** generato dal primo padre stampare il messaggio "**sono il processo figlio del primo padre**". A seguire tale processo figlio genera un altro processo (il processo **nipote** del primo processo padre)
- ◆ il **processo nipote** richiamerà l'eseguibile **stampa_info** passando il parametro "**nipote**"; quindi poi termina
- ◆ il processo figlio deve attendere la terminazione del nipote per poi terminare lui stesso; il primo processo padre deve attendere la terminazione del figlio per terminare lui stesso.

Esercizio 1

- ◆ *il secondo processo **padre**, dopo la terminazione del primo processo padre, stamperà il messaggio “**sono il secondo processo padre**” e poi genererà un processo **figlio***
- ◆ *il **processo figlio** generato dal secondo padre richiamerà l'eseguibile **stamp_info** passando il parametro “**figlio**”; quindi poi termina*
- ◆ *il secondo processo padre deve attendere la terminazione del figlio per terminare poi lui stesso.*

Esercizio 1

```
/* stampa_info.c */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

int main (int argc, char* argv[]) {

    if (argc < 1) {
        printf("USAGE: ./stampa_info <tipo_parentela>\n");
    }

    if (strcmp(argv[1], "nipote")==0) {
        printf("Sono il nipote del primo padre\n");
    }
    else {
        printf("Sono il figlio del secondo padre\n");
    }

    exit(0);
}
```

Esercizio 1

```
/* nipote.c */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main (void) {

    int status;
    pid_t pid_f1, pid_f2, pid_n;
    pid_f1 = fork (); // creazione primo figlio
```

Esercizio 1

```
if (pid_f1 == 0) { // sono il primo figlio
    printf("Sono il processo figlio del primo padre\n");
    pid_n = fork ();
    if (pid_n == 0) { // sono il nipote
        execlp("./stampa_info","stampa_info",
                "nipote", (char *) 0);
        exit (0);
    }
    wait (&status); // il primo figlio attende il nipote
    exit (pid_n);
}
```

Esercizio 1

```
else { // sono il primo padre  
    printf("Sono il primo processo padre\n");  
    wait (&status); // il primo padre attende il  
                    // primo figlio  
}
```

Esercizio 1

```
pid_f2 = fork (); // il secondo padre genera un figlio

if (pid_f2 == 0) { // sono il secondo figlio
    execlp ("./stampa_info", "stampa_info",
             "figlio", (char *) 0);
    exit (0);
}
else {
    printf("Sono il secondo processo padre\n");
    wait (&status); // il secondo padre attende il
                     secondo figlio
}
}
```

Esercizio 1

- **Output:**

```
$ gcc stampa_info.c -o stampa_info
```

```
$ gcc nipote.c -o test
```

```
$ ./test
```

Sono il primo processo padre

Sono il processo figlio del primo padre

Sono il nipote del primo padre

Sono il secondo processo padre

Sono il figlio del secondo padre

Esercizio 2

- Scrivere **compila_esegui.c**, un programma in Linguaggio C, per il controllo dei processi con la seguente interfaccia:

```
./compila_esegui <file1.c> <file2.c> ... <fileN.c>
```

- Il processo **padre**, per ogni file sorgente C **<fileI.c>**, stamperà il messaggio “**procedo con il file: <fileI.c>**“ e genererà un processo **figlio**
- Ogni processo **figlio** generato deve generare un processo **nipote** :
 - il processo **nipote** deve invocare il comando **gcc** per compilare e produrre l'eseguibile del singolo file sorgente C (si usi il nome del file sorgente **fileI** come nome dell'eseguibile) e poi terminare,
 - mentre il processo **figlio** attesa la terminazione del processo **nipote** invocherà l'eseguibile e poi terminerà.
- Il processo **padre** deve attendere la terminazione del processo figlio prima di procedere al file successivo.

Esercizio 2

- ◆ *Si ricorda:*
 - ◆ *si utilizzi la funzione **excel** per l'esecuzione di un comando esterno*
 - ◆ *Per conoscere il codice di terminazione potremo usare la macro **WEXITSTATUS(status)**.*

Esercizio 2

- ◆ Si utilizzi il seguente codice come schema dei file sorgente C:

```
#include <stdio.h>
#include <string.h>

typedef char stringa[20];
stringa name;

void get_name(stringa file) {
    int i, cont = 0;
    for (i = 0; file[i] != '.'; i++)
        name[i] = file[i];
}

int main (void) {
    stringa file = "fileI.c";
    get_name(file);
    printf("Nome del file: %s\n",name);
    exit(0);
}
```

Esercizio 2 (simulazione)

- **Output:**

```
$ gcc compila_esegui -o compila_esegui
```

```
$ ./compila_esegui file1.c file2.c
```

procedo con il file: file1.c

Nome del file: file1

procedo con il file: file2.c

Nome del file: file2

Esercizio 2

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

typedef char stringa[20];
stringa name;

char* get_name(stringa file) {
    int i, cont = 0;
    for (i = 0; file[i] != '.'; i++)
        name[i] = file[i];
    return name;
}
```

Esercizio 2

```
int main (int argc, char* argv[]) {  
  
    int i;  
    /* controllo argomenti */  
    if (argc < 2) {  
        printf("[USAGE]: ./compila_esegui file1.c ... filen.c\n");  
        return 0;  
    }  
}
```

Esercizio 2

```
int status;  
for (i=1; i < argc; i++){  
    int pid = fork();  
    if (pid > 0) { /* processo padre */  
        printf("procedo con il file: %s\n", argv[i]);  
        wait(&status);  
    }  
}
```

Esercizio 2

```
else {
    if (pid == 0) { /* processo figlio */
        /* recupero nome sorgente ed eseguibile */
        pid = fork();
        if (pid > 0) { /* processo figlio */
            int terminated_pid = wait(&status);
            /* verifica condizioni di errore */
            if (WEXITSTATUS(status)==0){
                execl(get_name(argv[i]),
                      get_name(argv[i]),(char*)0);
                perror("esecuzione");
                exit(1);
            }
        }
    }
}
```

Esercizio 2

```
else { /* processo nipote */
    if (pid == 0) {
        execl ("/usr/bin/gcc", "/usr/bin/gcc",
               argv[i], "-o",
               get_name(argv[i]), (char*)0);
        perror("compilazione");
        exit(1);
    }
}
return 0;
}
```

Esercizio 2

- **Output:**

```
$ gcc compila_esegui -o compila_esegui
```

```
$ ./compila_esegui file1.c file2.c
```

procedo con il file: file1.c

Nome del file: file1

procedo con il file: file2.c

Nome del file: file2

Esercizio 3

- Scrivere **stampa_argv.c**, un programma in Linguaggio C, con la seguente interfaccia:

`./stampa_argv <parametro1> <parametro2> <parametro3>`

- tale programma stampa su output tutti gli argomenti della linea di comando come segue

Nome del programma: **stampa_argv**

Parametro n. 1: **parametro1**

Parametro n. 2: **parametro2**

Parametro n. 3: **parametro3**

Esercizio 3

- ◆ Scrivere **attesa_random.c**, un programma in Linguaggio C
- ◆ Il processo **padre**, crea due figli e poi si mette in attesa della terminazione di entrambi; prima di uscire stampa i PID dei processi terminati, nell'ordine in cui questi terminano
- ◆ il primo processo **figlio** stampa su output il suo PID ed esegue il programma “**stampa_argv**” a cui passa tre parametri fittizi.
- ◆ Il secondo processo **figlio** crea a sua volta un processo **figlio**, il terzo, e:
 - ◆ Il secondo processo **figlio** stampa su output il suo PID e termina subito, senza attendere la terminazione del processo generato
 - ◆ Il terzo processo **figlio** stampa su output il suo PID e aspetta un numero casuale di secondi (da 0 a 5) prima di terminare (per attendere K secondi si usa la funzione “**sleep(K)**” e per generare numeri casuali tra 0 e N-1 si utilizza “**rand()%N**”)

Esercizio 3

```
/* stampa_argv.c */

#include <stdio.h>

int main ( int argc , char *argv[] ) {
    if ( argc == 4 ) {
        printf ( "Nome del programma: %s\n" , argv[0] );
        printf ( "Parametro n. 1: %s\n" , argv[1] );
        printf ( "Parametro n. 2: %s\n" , argv[2] );
        printf ( "Parametro n. 3: %s\n" , argv[3] );
    }
    else {
        /* Errore: parametri di input errati */
        printf ( "\nUtilizzo: %s Par_1 Par_2 Par_3\n\n" , argv[0] );
    }
}
```

Esercizio 3

```
/* attesa_random.c */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main (void) {

    pid_t pid1, pid2, pid3, term1, term2;
    int stat01, stat02;

    // si richiama la funzione fork
    // per creare il primo figlio

    pid1 = fork ();
```

Esercizio 3

```
if ( pid1 == 0 ) {
    // quello che segue è il codice eseguito
    // dal primo figlio
    printf ("[Primo figlio - PID: %d] Lancio il
            programma 'stampa_argv'\n", getpid());
    execl ("./stampa_argv", "stampa_argv",
           "Parametro_#1",
           "Parametro_#2",
           "Parametro_#3", NULL);
    exit ( -1 );
}
```

Esercizio 3

```
// il padre procede subito senza aspettare la terminazione del figlio, si  
chiama ancora la funzione fork per creare il secondo figlio  
  
pid2 = fork ();  
if ( pid2 == 0 ) {  
// quello che segue è il codice eseguito dal secondo figlio  
    printf ("[Secondo figlio - PID: %d] Creo una altro figlio, il  
          terzo\n", getpid());  
    pid3 = fork ();  
    if (pid3 == 0 ) {  
        // quello che segue è il codice eseguito dal terzo figlio  
        printf ("[Terzo figlio - PID: %d] Vado a dormire...\n",  
               getpid());  
        sleep (rand()%5);  
        printf ("[Terzo figlio] Svegliato! Termino subito.\n");  
        exit ( 2 );  
    }  
    exit (1);  
}
```

Esercizio 3

```
// si attende la terminazione dei primi due figli, non si sa  
in quale ordine  
  
term1 = wait ( &stato1 );  
term2 = wait ( &stato2 );  
  
// probabilmente terminano i primi due figli, perchè il  
terzo dorme un po'...  
  
printf ("Ordine di terminazione dei primi due figli:  
        1: %d - 2: %d\n", term1, term2);  
exit ( 0 );  
}
```

Esercizio 3

- **Output:**

```
$ gcc stampa_argv.c -o stampa_argv  
$ gcc attesa_random.c -o test  
$ ./test  
  
[Primo figlio - PID: 3080] Lancio il programma 'stampa_argv'  
[Secondo figlio - PID: 3081] Creo una altro figlio, il terzo  
[Terzo figlio - PID: 3082] Vado a dormire...  
  
Nome del programma: stampa_argv  
  
Parametro n. 1: Parametro_#1  
Parametro n. 2: Parametro_#2  
Parametro n. 3: Parametro_#3  
  
[Terzo figlio] Svegliato! Termino subito.  
  
Ordine di terminazione dei primi due figli: 1: 3081 - 2: 3080
```



```
Current conditions at Pescara, Italy (IBP) 42.26N 014.12E 11M (IBP)
Last updated Feb 10, 2012 - 02:50 PM EST / 2012-02-10 1950 UTC
Temperature: 1 C
Relative Humidity: 80%
Wind: from the W (270 degrees) at 15 MPH (13 KT) gusting to 45 KPH
Weather: light snow grains
Sky conditions: overcast
Su Mo Tu We Th Fr Sa      Su Mo Tu We Th Fr Sa
feb 29 30 31 01 02 03 04    mar 04 05 06 07 08 09 10
  05 06 07 08 09 10 11    11 12 13 14 15 16 17
  12 13 14 15 16 17 18    18 19 20 21 22 23 24
  19 20 21 22 23 24 25    25 26 27 28 29 30 31
mar 26 27 28 29 01 02 03    apr 01 02 03 04 05 06 07
silvio@Stain ~ $ cd Video
silvio@Stain ~ /Video $ movgrab http://vimeo.com/27998081
Formats available for this Movie: flv
Selected format: flv
Progress: 61.47% 15.4M of 25.1M 693.6K/s
```

