

Fondamenti di Informatica 2013-2014



Processi - da completare

Paola Mussida
Area Servizi ICT

Stampare il pid del processo
e del processo che l'ha
generato.

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

```
int main ()  
{
```

```
    sleep(60);  
    return 0;  
}
```

Generare un processo figlio
e mostrare i pid sia del
padre che del figlio.

Esercizio Pr_2 - Genera

5

```
#include <stdio.h>
#include <stdlib.h> //exit
#include <sys/types.h>
#include <unistd.h> //fork

int main ()
{
    pid_t child_pid;
    printf ("M the main program process id is %d\n",
           (int) getpid ());
```

```
else {
```

```
}  
return 0;
```

}

Generare un processo figlio
che lancia i due differenti
programmi esterni

✓ LS

✓ PWD

tramite la funzione execl.

```
#include <stdio.h>
#include <stdlib.h> //necessario per exit
#include <sys/types.h>
#include <unistd.h>
```

```
int spawnLS (char* path, char* nome,
             char* par1, char* par2);
```

```
int spawnPWD (char* path, char* nome );
```



```
int main ()
{
    int scelta;

    /* La lista di argomenti da passare al
       comando "ls". */

    char * path = "/bin/ls";
    char * nome = "ls";
    char * par1 = "-l";
    char * par2 = "/";
```

```
do{
    printf ("\nPremere: \n- 1 per ls; \n - 2 per pwd; \n");
    scanf("%d",&scelta);
}while (scelta != 1 && scelta!=2);

switch (scelta) {
    case 1:

        break;

    case 2:

        break;
}

printf ("done with main program\n");

return 0;
}
```

```
int spawnLS (char* path, char* nome, char* par1, char* par2)
{
    pid_t child_pid;

    /* Duplica questo processo. */

}
```

```
int spawnPWD (char* path, char* nome)
{
    pid_t child_pid;
```

```
/* Duplica questo processo. */
```

```
}
```

Scrivere un programma in cui il padre stampi 100000 volte una stringa differente da quella stampata dal figlio.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>

int main ()
{
    pid_t pid;
    int i=0;
    printf("I'm the original process with PID %d and
           PPID %d.\n", getpid(),getppid());

    /* Replicazione. Padre e figlio continuano da qui.*/
```

Esercizio Pr_4 - Concorrenza

15

```
else {
```

```
}
```

```
}
```

Generare un numero prefissato di processi, ognuno caratterizzato da una differente sigla composta da un carattere (A, B, C, ...) assegnata durante la creazione dal padre. Tale sigla deve essere memorizzata in ogni processo figlio generato.

Il processo padre deve visualizzare il PID di ogni processo figlio generato e la sigla assegnatagli.

Ogni processo generato deve visualizzare il proprio PID e la stringa assegnatagli dal padre.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#define MAX_FIGLI 5

int main ()
{
    pid_t pid;
    char sigla;
    int i;
    printf("I'm the original process with PID %d.\n",
                                                getpid());

    /* Ciclo per generare i figli. */
    for (i = 0; i<MAX_FIGLI; i++) {
        /* ... */
    }
}
```

```
} //Termina il ciclo for
```

```
}
```

Implementare un programma che generi un numero prefissato di processi figli. Il processo padre deve memorizzare, per ogni figlio generato, il PID del nuovo processo e il valore da esso restituito al termine della propria esecuzione. Ogni processo figlio generato deve richiedere all'utente di inserire un carattere. Tale carattere costituisce il valore da restituire al padre.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#define MAX_FIGLI 3

void main ()
{
    /*array di strutture per contenere le info sui figli*/

    pid_t pidx;
    int status, i, j, uscita=0;
    char c;

    printf("I'm the original process with PID %d.\n", getpid());
```

22

}

Esercizio Pr_6 - Acquisizione e Ritorno

23

```
/*Attesa della fine di ogni processo figlio e memorizzazione del valore restituito*/  
/* codice padre: attesa termine di tutti i figli */  
for (j=0; j< MAX_FIGLI; j++)  
{  
    /*memorizzazione valore restituito dal figlio terminato*/  
    /*attesa termine figli*/  
  
    i=0;                /*ricerca del figlio terminato nell'array*/  
    uscita=0;  
    while (  
    {  
  
    }  
}
```

```
/*stampa dei risultati*/
```

```
} // chiusura main
```


Implementare un programma che, lanciati due programmi, attenda la fine di entrambi per stabilire quale dei due è terminato per primo.

Il primo comando da invocare è:
“/bin/ls” con i parametri “-l” e “/”.

Il secondo comando è semplicemente:
“/bin/pwd”.

```
#include <stdio.h>
#include <stdlib.h> //necessario per exit
#include <sys/types.h>
#include <unistd.h>
```

```
void main ()
{
```

```
    /*Generazione dei due figli*/
```

```
/*Generazione primo figlio*/
```

```
if (
```

```
}
```

```
else {
```

```
    if (
```

```
{
```

```
/* Questo e' il primo figlio. */
```

```
}
```

```
else
{
    /*generazione del secondo figlio da parte del padre*/

    if (

    }
    else {
        if (                /*Questo e' il secondo figlio*/
        {

        }

    }
}
}
```

```
/*codice padre */
```

```
/*attesa del figlio più veloce*/
```

```
}
```

Provare a creare un processo orfano.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>

int main ()
{
    pid_t pid;
    printf("I'm the original process with PID %d and PPID %d.\n",
        getpid(),getppid());

    /* Replicazione. Padre e figlio continuano da qui.*/

}
```

Esercizio Pr_8 - Orfano

32

```
else {
```

```
}
```

```
}
```


Provare a creare un processo zombie.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
```

```
int main ()
{
```

```
    pid_t pid;
```

```
    printf("I'm the original process with PID %d and PPID %d.\n",
           getpid(),getppid());
```

```
    /* Replicazione. Padre e figlio continuano da qui.*/
```

```
else {
```

```
}
```

```
}
```

Analisi dell'esecuzione concorrente di un programma:

- ✓ albero dei processi;
- ✓ esposizione delle parti di codice di ogni processo;
- ✓ diagrammi di flusso temporale;

Esercizio Pr_10 - Analisi concorrenza

37

```
void main ()
{
    int v1,v2=14;
    pid_t pid;
    pid=fork();
    if (pid ==-1) <----- // T1 subito dopo la fork
    {
        printf("\nan error occurred\n");
        exit(-1);
    }
    else {
        if (pid == 0)
        {
            /* Processo figlio. */
            foo(&v1);
            exit(0); <----- // v1 = 65 in foo()
                                // T2 subito prima della exit
        }
        else
        {
            /* Processo padre*/
            fun(&v2);
            wait(...);
            v1=455; <----- // v2 = 321 in fun()
                                // T3 subito prima della wait
                                // T4 subito dopo l'assegnamento
        }
    }
}
```

Albero dei processi

Serve per evidenziare le relazioni padre-figlio


Esporre le parti di codice eseguite
da ciascun processo: **P padre**

Esercizio Pr_10 - Analisi concorrenza

40

Esporre le parti di codice eseguite
da ciascun processo: **PF figlio**

Diagramma di flusso temporale dei processi



tempo

Tabella da compilare per ogni processo P e PF

	T1	T2	T3	T4
PID				
V1				
V2				

Legenda:

- ✓ NE: il contesto non esiste;
- ✓ n: la variabile esiste e ha valore n;
- ✓ P o PF: indicano i PID dei rispettivi processi;
- ✓ X: la variabile esiste ma non è stata ancora inizializzata;
- ✓ ?: la variabile può non esistere o essere caratterizzata da diversi valori.

Tabella da compilare per il processo P: padre

	T1	T2	T3	T4
PID				
V1				
V2				

Legenda:

- ✓ NE: il contesto non esiste;
- ✓ n: la variabile esiste e ha valore n;
- ✓ P o PF: indicano i PID dei rispettivi processi;
- ✓ X: la variabile esiste ma non è stata ancora inizializzata;
- ✓ ?: la variabile può non esistere o essere caratterizzata da diversi valori.

Tabella da compilare per il processo PF: figlio

	T1	T2	T3	T4
PID				
V1				
V2				

Legenda:

- ✓ NE: il contesto non esiste;
- ✓ n: la variabile esiste e ha valore n;
- ✓ P o PF: indicano i PID dei rispettivi processi;
- ✓ X: la variabile esiste ma non è stata ancora inizializzata;
- ✓ ?: la variabile può non esistere o essere caratterizzata da diversi valori.

Analisi dell'esecuzione concorrente di un programma.

- ✓ Un processo padre P crea nell'ordine i tre processi figli F_1 , F_2 e F_3 e, dopo averli creati, si mette in attesa della loro terminazione.

- ✓ I tre figli evolvono in modo autonomo, eseguendo tre programmi diversi, il cui comportamento è peraltro sconosciuto.
- ✓ Quando i processi figli sono tutti terminati, anche il processo padre termina, visualizzando i PID dei tre processi figli, in ordine di terminazione.
- ✓ A questo scopo, il processo padre P memorizza l'elenco dei PID dei processi figli in un array di tipo `pid_t pid[3]`, nel quale scrive, uno dopo l'altro, i PID dei tre processi figli creati tramite la primitiva `fork()`.

Esercizio Pr_11 - Analisi concorrenza

47

```
void main ()
{
    pid_t pid[3], term[3];
    int n, stato;
        <-----

    /*Generazione primo figlio*/
    pid[0] = fork(); <-----
    //...
    if (pid[0] == 0) {
        /*corpo primo figlio*/
        exit(0);
    }
    /*Generazione secondo figlio*/
    pid[1] = fork(); <-----
    //...
    if (pid[1] == 0) {
        /*corpo secondo figlio*/
        exit(0);
    }

    /*Generazione terzo figlio*/
    pid[2] = fork(); <-----
    //...
    if (pid[2] == 0) {
        /*corpo terzo figlio*/
        exit(0);
    }
}
```

// creazione P

// creazione F1

// creazione F2

// creazione F3

```
/*Ciclo di attesa terminazione figli*/  
for(n=0; n<3; n++)  
    term[n] = wait(&stato);  
  
/*stampa i PID dei figli in ordine di terminazione*/  
for(n=0; n<3; n++)  
    printf("Terminato figlio con PID %d ",term[n]);  
}
```


Tabella da compilare per ogni processo

	Creaz P	Creaz F1	Creaz F2	Creaz F3
pid[0]				
pid[1]				
pid[2]				

Legenda:

- ✓ NE: il contesto non esiste;
- ✓ n: la variabile esiste e ha valore n;
- ✓ P o PF: indicano i PID dei rispettivi processi;
- ✓ X: la variabile esiste ma non è stata ancora inizializzata;
- ✓ ?: la variabile può non esistere o essere caratterizzata da diversi valori.

Albero dei processi

Serve per evidenziare le relazioni padre-figlio

Tabella per il processo P

	Creaz P	Creaz F1	Creaz F2	Creaz F3
pid[0]				
pid[1]				
pid[2]				

Legenda:

- ✓ NE: il contesto non esiste;
- ✓ n: la variabile esiste e ha valore n;
- ✓ P o PF: indicano i PID dei rispettivi processi;
- ✓ X: la variabile esiste ma non è stata ancora inizializzata;
- ✓ ?: la variabile può non esistere o essere caratterizzata da diversi valori.

Tabella per il processo F1

	Creaz P	Creaz F1	Creaz F2	Creaz F3
pid[0]				
pid[1]				
pid[2]				

Legenda:

- ✓ NE: il contesto non esiste;
- ✓ n: la variabile esiste e ha valore n;
- ✓ P o PF: indicano i PID dei rispettivi processi;
- ✓ X: la variabile esiste ma non è stata ancora inizializzata;
- ✓ ?: la variabile può non esistere o essere caratterizzata da diversi valori.

Tabella per il processo F2

	Creaz P	Creaz F1	Creaz F2	Creaz F3
pid[0]				
pid[1]				
pid[2]				

Legenda:

- ✓ NE: il contesto non esiste;
- ✓ n: la variabile esiste e ha valore n;
- ✓ P o PF: indicano i PID dei rispettivi processi;
- ✓ X: la variabile esiste ma non è stata ancora inizializzata;
- ✓ ?: la variabile può non esistere o essere caratterizzata da diversi valori.

Tabella per il processo F3

	Creaz P	Creaz F1	Creaz F2	Creaz F3
pid[0]				
pid[1]				
pid[2]				

Legenda:

- ✓ NE: il contesto non esiste;
- ✓ n: la variabile esiste e ha valore n;
- ✓ P o PF: indicano i PID dei rispettivi processi;
- ✓ X: la variabile esiste ma non è stata ancora inizializzata;
- ✓ ?: la variabile può non esistere o essere caratterizzata da diversi valori.

Analisi dell'esecuzione concorrente di un programma:

- ✓ albero dei processi;
- ✓ esposizione delle parti di codice di ogni processo;
- ✓ diagrammi di flusso temporale;

- ✓ Il programma seguente viene eseguito inizialmente da un processo P, che crea due processi figli PF₁ e PF₂ (i cui identificatori vengono assegnati alle variabili f₁ e f₂).
- ✓ A sua volta il processo PF₁ crea due processi figli PN₁ e PN₂ (i cui identificatori vengono assegnati alle variabili n₁ e n₂).
- ✓ Tali processi PN₁ e PN₂ sono immaginabili come “nipoti” del processo P.

Esercizio Pr_12 - Analisi concorrenza

57

```
void main () {
    pid_t f1, f2, n1, n2;
    /*Generazione primo figlio*/
    f1 = fork();<-----
    //...
    if (f1 == 0) {
        /*corpo primo figlio*/
        n1 = fork();
        //...
        if (n1 == 0) {
            /* corpo primo nipote */
            fun();
            exit(0);
        } else {
            /*Codice del primo figlio*/
            n2 = fork();
            //...
            if (n2 == 0) {
                /* corpo secondo nipote */
                fun();
                exit(0);
            } else {
                /*fine primo figlio*/
                exit(0);<-----
            }
        }
    }
}
```

// T1 dopo la fork

// T2 prima della exit

```
else
{
    /*Codice del padre*/
    wait(...);
    /*Generazione secondo figlio*/
    f2= fork();
    //...
    if (f2 == 0) {
        /*corpo secondo figlio*/
        fun();
        exit(0);
    }
    else
    {
        /*Codice del padre*/
        wait(...);
        exit();
    }
}
```

// T3 prima della exit

Tabella da compilare per ogni processo

	T ₁	T ₂	T ₃
f ₁			
f ₂			
n ₁			
n ₂			

Legenda:

- ✓ NE: il contesto non esiste;
- ✓ n: la variabile esiste e ha valore n;
- ✓ P o PF: indicano i PID dei rispettivi processi;
- ✓ X: la variabile esiste ma non è stata ancora inizializzata;
- ✓ ?: la variabile può non esistere o essere caratterizzata da diversi valori.

Albero dei processi

Serve per evidenziare le relazioni padre-figlio

Esporre le parti di codice eseguite
da ciascun processo: **P padre**

Esporre le parti di codice eseguite
da ciascun processo: **f1**

Esporre le parti di codice eseguite
da ciascun processo: **f2**

Esporre le parti di codice eseguite
da ciascun processo: **n1**

Diagramma di flusso temporale dei processi

caso 1: PN₁ termina dopo T₁ ma prima di T₂

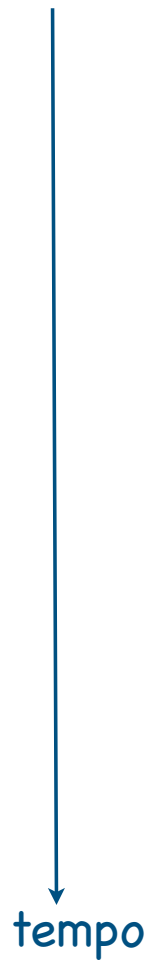




Diagramma di flusso temporale dei processi

caso 2: PN₁ termina dopo T₂ ma prima di T₃



tempo

Diagramma di flusso temporale dei processi caso 3: PN1 termina dopo T3



tempo

Tabella da compilare per il processo P

	T ₁	T ₂	T ₃
f ₁			
f ₂			
n ₁			
n ₂			

Legenda:

- ✓ NE: il contesto non esiste;
- ✓ n: la variabile esiste e ha valore n;
- ✓ P o PF: indicano i PID dei rispettivi processi;
- ✓ X: la variabile esiste ma non è stata ancora inizializzata;
- ✓ ?: la variabile può non esistere o essere caratterizzata da diversi valori.

Tabella da compilare per il processo f1

	T1	T2	T3
f1			
f2			
n1			
n2			

Legenda:

- ✓ NE: il contesto non esiste;
- ✓ n: la variabile esiste e ha valore n;
- ✓ P o PF: indicano i PID dei rispettivi processi;
- ✓ X: la variabile esiste ma non è stata ancora inizializzata;
- ✓ ?: la variabile può non esistere o essere caratterizzata da diversi valori.

Tabella da compilare per il processo f2

	T1	T2	T3
f1			
f2			
n1			
n2			

Legenda:

- ✓ NE: il contesto non esiste;
- ✓ n: la variabile esiste e ha valore n;
- ✓ P o PF: indicano i PID dei rispettivi processi;
- ✓ X: la variabile esiste ma non è stata ancora inizializzata;
- ✓ ?: la variabile può non esistere o essere caratterizzata da diversi valori.

Tabella da compilare per il processo n1

	T ₁	T ₂	T ₃
f ₁			
f ₂			
n1			
n2			

Legenda:

- ✓ NE: il contesto non esiste;
- ✓ n: la variabile esiste e ha valore n;
- ✓ P o PF: indicano i PID dei rispettivi processi;
- ✓ X: la variabile esiste ma non è stata ancora inizializzata;
- ✓ ?: la variabile può non esistere o essere caratterizzata da diversi valori.

Analisi dell'esecuzione concorrente di un programma.

- ✓ Si supponga che tutte le chiamate ai servizi di sistema abbiano sempre successo e che il S.O. assegni ai processi creati dei pid consecutivi a partire da 111.


```
void main () {
    pid_t pid;
    int i, j, dati[2], status;
    i=0;
    dati[0]=dati[1]=-1;
    for (j=0; j<2; j++) {
        pid=fork();
        if (pid==0){
            dati[i]=j;
            if (j == 0) {
                execl("/bin/pwd", "pwd", NULL);
                exit(1);
            }
            exit(1); //istr. eseguita solo dal 2o figlio
        }
        if (j == 1) pid = waitpid(pid, &status, 0);
        i++;
    }
    exit(0);
}
```

Tabella per il processo P

	i	pid	dati[0]	dati[1]
pre istr. 6				
pre istr. 14				
pre istr. 19				

Legenda:

- ✓ NE: il contesto non esiste;
- ✓ n: la variabile esiste e ha valore n;
- ✓ P o PF: indicano i PID dei rispettivi processi;
- ✓ X: la variabile esiste ma non è stata ancora inizializzata;
- ✓ ?: la variabile può non esistere o essere caratterizzata da diversi valori.

Tabella per il processo 111

	i	pid	dati[0]	dati[1]
pre istr. 6				
pre istr. 14				
pre istr. 19				

Legenda:

- ✓ NE: il contesto non esiste;
- ✓ n: la variabile esiste e ha valore n;
- ✓ P o PF: indicano i PID dei rispettivi processi;
- ✓ X: la variabile esiste ma non è stata ancora inizializzata;
- ✓ ?: la variabile può non esistere o essere caratterizzata da diversi valori.

Tabella per il processo 112

	i	pid	dati[0]	dati[1]
pre istr. 6				
pre istr. 14				
pre istr. 19				

Legenda:

- ✓ NE: il contesto non esiste;
- ✓ n: la variabile esiste e ha valore n;
- ✓ P o PF: indicano i PID dei rispettivi processi;
- ✓ X: la variabile esiste ma non è stata ancora inizializzata;
- ✓ ?: la variabile può non esistere o essere caratterizzata da diversi valori.