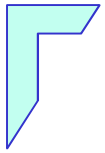
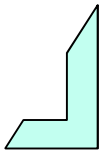


Esercitazione di Lab. di Sistemi Operativi

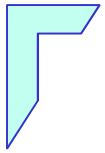
- I processi Unix -



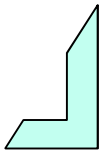
Sommario

- Gestione dei processi:
  - Creazione di un nuovo processo: fork
  - Sincronizzazione tra processi: wait

- Esercizi:
  - fork
  - fork + wait



- Gestione dei processi: fork -



- Gestione dei processi: fork -

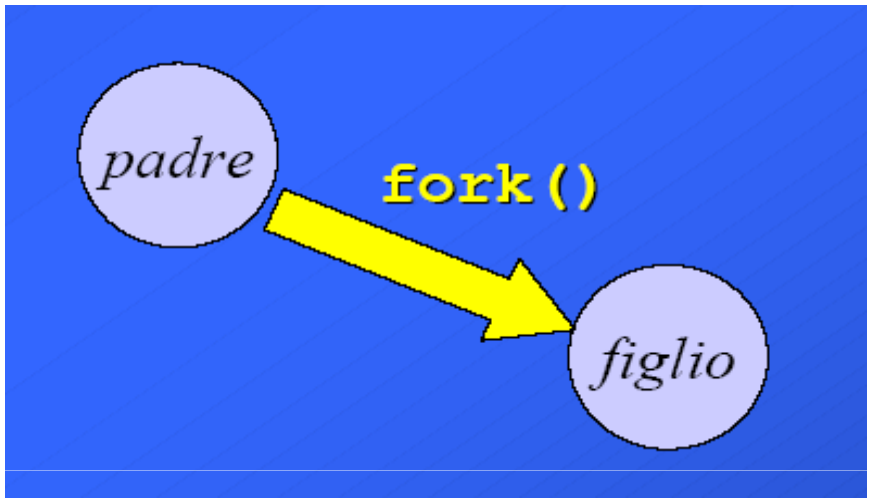
- Creazione processi: "la funzione **pid_t fork(void)**"

```
#include <sys/types.h>
#include <unistd.h>
                          (vfork da usare con exec, lavora nello
pid_t fork(void);         spazio del genitore e aspetta il figlio)
pid_t vfork(void);

                          ritornano il PID se OK, -1 se EAGAIN, ENOMEM o EPERM
```

- La fork non richiede parametri
- Istruisce il kernel Unix a creare un nuovo processo chiamato **figlio** da un processo esistente chiamato **padre**

- Gestione dei processi: fork -

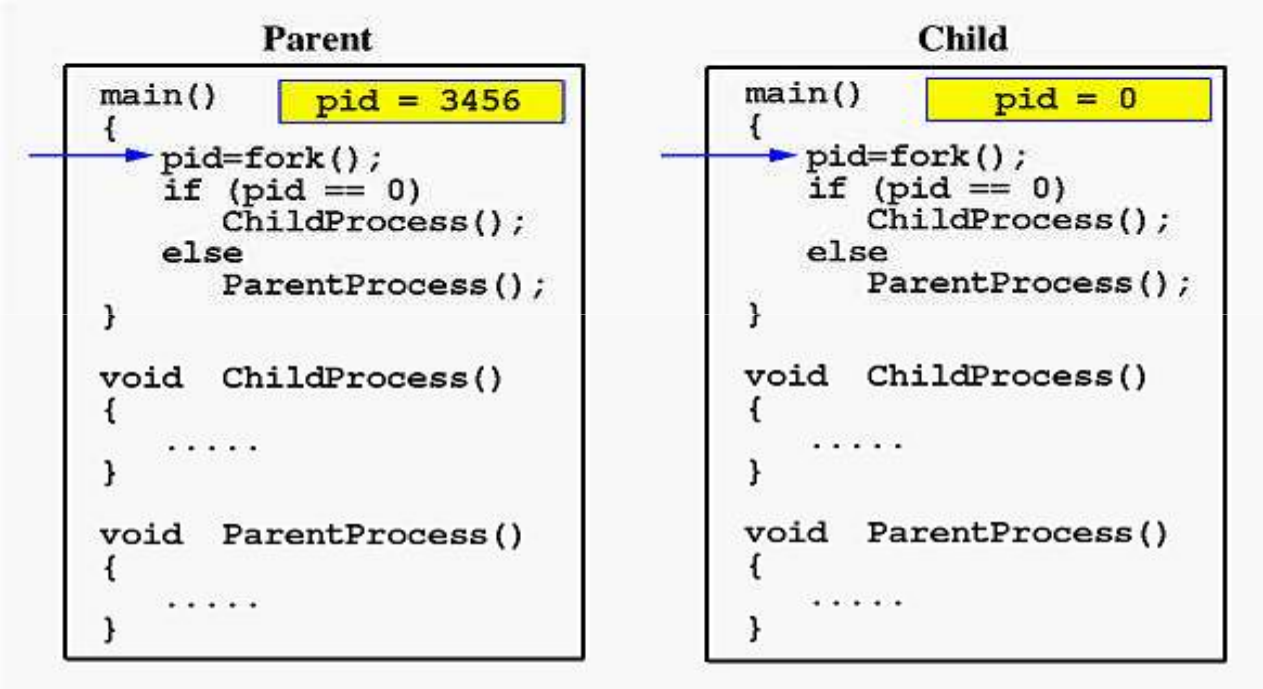


Ogni chiamata a **fork** (**vfork**) ha due ritorni:

1. Al genitore viene restituito l' **ID del figlio**
2. Al figlio viene restituito **l' ID 0.**

- Esempio: fork -

Esecuzione di una **fork()** da parte del programma main:



Relazione padre - figlio

Quando il programma **main**, esegue una **fork()**:

1. Padre e figlio procedono in parallelo

2. Ogni variabile del figlio è inizializzata con il valore assegnatole dal padre prima della fork()

3. Le risorse allocate al padre (ad esempio, i file aperti) prima della generazione sono condivise con i figli

4. Le informazioni per la gestione dei segnali sono le stesse per padre e figlio (associazioni segnali-handler)

5. Il figlio nasce con lo stesso Program Counter del padre, la prima istruzione eseguita dal figlio è quella che segue immediatamente la fork().

Esercizio n° 1 - fork -

- Si realizzi un programma C "**fork1.c**" che stampi a video il pid del processo che lancerà un **fork** e dopo l'esecuzione (fork), sia in grado di identificare i processi genitore e figlio con i relativi pid testando i valori di ritorno da fork.

Esecuzione: $ ./**fork1.out**

Output

$ Unico processo con (PID: numPid) che lancia la fork
Sono il **padre** con (PID: numPid). con un proc. figlio (PID: numPid)
Sono il **figlio** con (PID: numPid)

Suggerimento:
Per ottenere il pid del processo padre utilizzare la funzione:

```
pid_t getpid(void)
```

Soluzione Esercizio N° 1

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
int main()
{   int pid;
    printf("Unico processo con (PID: %d).\n",getpid());
    pid = fork(); /*Esecuzione fork*/
    if (pid == 0)
    {/* codice eseguito dal figlio */
     printf("Sono il figlio con (PID : %d).\n",getpid());
    }else if (pid > 0)
    {/* codice eseguito dal padre */
    printf("Sono il padre con (PID: %d). con un proc. figlio
    (PID: %d)\n",getpid(),pid);
    }else
    {/* codice eseguito dal padre in caso di errore */
      printf("Si e' verificato un errore nella chiamata a
fork.\n");
     exit(1);
    }
    exit(0);
}
```

Esercizio n° 2 - fork -

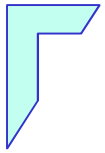
- Si realizzi un programma "**fork2.c**" che possa ricevere come input un numero intero immesso da tastiera. A tale numero, il processo figlio creato somma 15, mentre il processo padre somma 10

Esecuzione:
$ ./fork2.out <valore>
Output:
Processo padre Pid: numPid valore = <valore> + 10
Processo figlio Pid: numPid valore = <valore> + 15

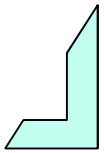
```
                    ESEMPIO CON VALORE = 8:
Unico processo con PID 28827.
padre [pid: 28827]> generato un figlio; il suo pid e' 28828
padre [pid: 28827]> valore iniziale= 8
padre [pid: 28827]> valore finale= 18
figlio [pid: 28828]> valore iniziale= 8
figlio [pid: 28828]> valore finale= 23
```

Soluzione Esercizio N° 2

```
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
int main(int argc,char **argv)
{
     int valore = 0;
     if ( 1 < argc )
     valore = atoi( argv[ 1 ] ); /*trasforma una stringa in un intero*/
     printf("Unico processo con PID %d.\n",getpid());
     int pid = fork(); /*Creazione processo figlio*/
     if (pid == 0) /* codice eseguito dal figlio */
     {   printf( "figlio [pid: %d]> valore iniziale= %d\n", getpid(), valore );
          valore += 15;
          printf( "figlio [pid: %d]> valore finale= %d\n", getpid(), valore );
     }
     else if (pid > 0) /* codice eseguito dal padre */
     {printf("padre [pid: %d]> generato un figlio; il suo pid e' %d\n",getpid(),
pid);
          printf( "padre [pid: %d]> valore iniziale= %d\n", getpid(), valore );
          valore += 10;
          printf( "padre [pid: %d]> valore finale= %d\n", getpid(), valore );
     }
     else
     {printf("padre [pid: %d]> problemi durante creazione del
     figlio.\n",getpid());
     exit(1);}
     exit(0);}
```



- Sincronizzazione tra processi: wait -



- Sincronizzazione tra processi: wait -

- La funzione **wait**:

```
#include<sys/types.h>
#include<sys/wait.h>
#include<sys/unistd.h>

pid_t wait(int *status)
```

- Grazie alla funzione **wait**, il padre attende la terminazione del processo figlio.
- Restituisce:
  - il pid del processo terminato
  - -1 in caso di errore: ad esempio se un processo non ha figli
- "status " contiene il valore di uscita del figlio e letto con le macro:
  - **WIFEXITED**(status):
    - è vero se il figlio termina normalmente
  - **WEXITSTATUS**(status)
    - restituisce il valore di status in caso che il figlio termina in maniera regolare

- Esempio: wait -

- **wait** con macro:

Esempio tipico:

```
int status;

wait(&status);
if ( WIFEXITED(status) )
     printf("valore di uscita: %d\n", WEXITSTATUS(status));
else
     printf("terminazione anomala\n");
```

True se il processo termina normalmente

Restituisce lo stato del processo

Esercizio n° 3 – fork + wait -

- Scrivere un programma C che prenda come parametro un file **inputFile** (controllare che il numero di argomenti passati sia corretto). Il programma dovrà aprire tale file, creare un processo figlio, **PF** che scriverà nel file **inputFile** una stringa inserita da tastiera. Il padre attenderà il figlio e visualizzerà sul video il contenuto del file **inputFile**

Esecuzione: $ ./**forkWait1.out** <nome_file>
Introduci una stringa e premi [Enter]

Output:

Il padre ha letto la stringa <valore_stringa>

Soluzione Esercizio N° 3 1/2

```
#include <stdio.h> /* perror */
#include <errno.h> /* perror */
#include <unistd.h> /* write, lseek, close*/
#include <sys/types.h> /*open, lseek */
#include <sys/stat.h> /*open */
#include <fcntl.h> /*open*/
#include <stdlib.h> /*exit*/
#include <string.h> /*strlen*/
int main (int argc, char **argv)
{
    int fd,pid, pid_processo;
    int nread,nwrite,status;
    char st1[256]; /*Stringa lunga 256 caratteri*/
    char st2[256]; /*Stringa lunga 256 caratteri*/
    if (argc != 2)
    {
      fprintf(stderr,"Uso: %s nomefile\n",argv[0]);
      exit(1);
    }/* Apertura del file in lettura e scrittura */
    if ((fd=open(argv[1], O_RDWR | O_CREAT, S_IRUSR | S_IWUSR))<0)
    {  perror("opening argv[1]");
       exit(1);
     }/*CONTINUA NELLA SLIDE SUCCESSIVA*/
```

Soluzione Esercizio N° 3 2/2

```
if((pid=fork())<0)  /*Creazione processo figlio*/
{   perror("fork");
    exit(-1);}
else if (pid==0) /* Processo figlio */
{   printf("Introduci una stringa e premi [Enter]\n");
    scanf("%s",st1);
    /* Il figlio eredita il descrittore fd dal padre */
    nwrite= write(fd,st1,sizeof(st1));
    /* L' I/O pointer si e' spostato alla fine del file */
    exit(0);
}
else { /* pid > 0 : Processo padre */
    /* Attesa della terminazione del figlio */
    pid_processo = wait(&status);
    if ( WIFEXITED(status) ) printf("Padre: figlio ha terminato con status
%d.\n",WEXITSTATUS(status));
    else printf("Padre: figlio ha terminato in modo anomalo.\n");
    /* Riposizionamento all'inizio del file */
    lseek(fd,0,SEEK_SET);
    if( (nread = read(fd,st2,sizeof(st2))) <0)
    {perror("read ");
     exit(-1);}
    printf("Il padre ha letto la stringa %s\n",st2);/*Scrive a video*/
    close(fd);
}exit(0);}
```

Esercizio n° 4 – fork + wait -

- Scrivere un programma C che prenda come parametro due nomi di file **inputFile** ed **outputFile** (controllare che il numero di argomenti passati sia corretto) il programma dovrà generare un processo figlio, **PF** che aprirà il file **inputFile** copiando i primi 40 caratteri di **inputFile** in **outputFile** aggiungendo alla fine il proprio PID. Il padre attenderà il figlio ed aggiungerà ad **outputF** il proprio PID.

Esempio **inputFile**
1111111111222222222233333333334444444444555555555566666666666

Esempio di esecuzione:
./**forkWait2.out** <**inputFile**> <**outputFile**>

Contenuto di **outputFile**
1111111111222222222233333333334444444444<PidF><PidP>

Soluzione Esercizio N° 4 1/3

```
#include <stdio.h> /* perror */
#include <errno.h> /* perror */
#include <unistd.h> /* write, lseek, close */
#include <sys/types.h> /*open, lseek */
#include <sys/stat.h> /*open */
#include <fcntl.h> /*open*/
#include <stdlib.h> /*exit*/
#include <string.h> /*strlen*/
#define READLENGTH 40
#define WRITELENGTH 40
main(int argc, char **argv)
{
  int infile, status, oufile, pid, nread;
  int pid_processo;
  char buf[40];
  char id[10];
  if (argc != 3)
  {printf(" uso: nomeprogramma <inputFile> <outputFile> \n");
   exit (1);}
  /* Apertura file  outputFile*/
    if ((oufile=open(argv[2], O_RDWR | O_CREAT, S_IRUSR | S_IWUSR))
<0)
      {
         printf("Errore apertura outputFile %s\n:",argv[2]);
          exit(1);
      }
```

Soluzione Esercizio N° 4 2/3

```
if ((pid = fork()) == -1) /*Esecuzione fork*/
    {perror("fork");
    exit(1);}
else if (pid == 0) /*Processo figlio*/
{   /* Apertura file inputFile*/
    if ((infile=open(argv[1], O_RDONLY)) <0)
     {printf("Errore apertura inputFile da leggere
      %s\n:",argv[1]);
      exit(1); }
    /*legge i primi 40 caratteri*/
    if ( (nread = read(infile, buf, READLENGTH)) <= 0) {
            close(infile);
            exit(1);}
     /*scrive i primi 40 caratteri*/
    if ( (write(oufile, buf, WRITELENGTH )) <= 0) {
            close(oufile);
            exit(1);}
    sprintf(id, "(PID:%d) PF ", getpid());
    if ( (write(oufile, id, strlen(id))) <= 0) {
       close(oufile);
       exit(1);}
     close(infile); /* chiude il proprio file descriptor */
     exit(0);
}
```

Soluzione Esercizio N° 4 3/3

```
else {   /*Processo Padre*/
   /*Attende che il figlio finisca di scrivere*/
   pid_processo = wait(&status);
   if ( WIFEXITED(status) ) printf("Padre: figlio ha
   terminato con  status %d.\n",WEXITSTATUS(status));
   else printf("Padre: figlio ha terminato in modo
         anomalo.\n");
   printf("Inizio processo padre\n");
   sprintf(id, "(PID:%d) Padre", getpid());
   /*scrive i PID caratteri*/
      if ( (write(oufile, id, strlen(id))) <= 0) {
         close(oufile);
         exit(1);
      }
    printf("Fine processo padre\n");
   }
   close(oufile);
   exit(0); /*Esce dal programma */
}
```

Esercizio n° 5 – fork + wait -

- Scrivere un programma C che prenda come parametri due nomi di file **inputFile** ed **outputFile** (controllare che il numero di argomenti passati sia corretto) il programma dovrà creare un processo figlio, **PF** che aprirà il file **inputFile** e scriverà i primi 10 caratteri e gli ultimi 10 nel file **outputFile**. Il padre attenderà il figlio e visualizzerà sul video il contenuto del file **outputFile**

Esempio **inputFile**

1111111111222222222233333333334444444444555555555566666666666

Esempio di esecuzione:

./**forkWait3.out** <**inputFile**> <**outputFile**>

Outuput: Padre: <11111111116666666666>

Soluzione Esercizio N° 5 1/4

```
#include <stdio.h> /* perror */
#include <errno.h> /* perror */
#include <unistd.h> /* write, lseek, close*/
#include <sys/types.h> /*open, lseek */
#include <sys/stat.h> /*open */
#include <fcntl.h> /*open*/
#include <stdlib.h> /*exit*/
#include <string.h> /*strlen*/
#define READLENGTH 10
#define WRITELENGTH 20
main(int argc, char **argv)
{
    int infile, status, oufile, pid, nread;
    int pid_processo;
    char buf[10];
    if (argc != 3)
    { printf ("uso: nomeprogramma <inputFile> <outputFile> \n");
      exit(1);}
    /* Apertura file  outputFile*/
    if ((oufile=open(argv[2], O_RDWR | O_CREAT, S_IRUSR | S_IWUSR))
    <0)
      { printf("Errore apertura outputFile %s\n:",argv[2]);
            exit(1);
      } /*CONTINUA NELLA SLIDE SUCCESSIVA*/
```

Soluzione Esercizio N° 5 2/4

```
/*Esecuzione fork*/
if ((pid = fork()) == -1)
    {perror("fork");
    exit(1);
}
else if (pid == 0) /*Processo figlio*/
{   printf("Inizio processo figlio\n");
        /* Apertura file inputFile*/
    if ((infile=open(argv[1], O_RDONLY)) <0)
        { printf("Errore apertura inputFile da leggere
%s\n:",argv[1]);
          exit(1);
        }
        /*legge i primi 10 caratteri*/
        if ( (nread = read(infile, buf, READLENGTH)) <= 0) {
            close(infile);
            exit(1);
         }
         /*scrive i primi 10 caratteri*/
        if ( (write(oufile, buf, READLENGTH)) <= 0) {
            close(oufile);
            exit(1);
         } /*CONTINUA NELLA SLIDE SUCCESSIVA*/
```

```
/*Si Posiziona alla fine e torna indietro di 10*/
if (lseek(infile, lseek(infile, 0, SEEK_END)-READLENGTH,
    SEEK_SET) == -1)
        {perror( "lseek errore" );
         exit( -1 ); }

        /*legge gli ultimi 10 caratteri*/
        if ( (nread = read(infile, buf, READLENGTH)) <= 0) {
            close(infile);
            exit(1);
                }
                /*scrive gli ultimi 10 caratteri*/
        if ( (write(oufile, buf, READLENGTH)) <= 0) {
            close(oufile);
            exit(1);
                }
        /* chiude il proprio file descriptor */
        close(infile);
        printf("Fine processo figlio\n");
        exit(0);
    } /*CONTINUA NELLA SLIDE SUCCESSIVA*/
```

Soluzione Esercizio N° 5 3/4

Soluzione Esercizio N° 5 4/4

```
else {/*Processo Padre*/
       pid_processo = wait(&status);
       if ( WIFEXITED(status) ) printf("Padre: figlio ha
    terminato con status %d.\n",WEXITSTATUS(status));
       else printf("Padre: figlio ha terminato in modo
            anomalo.\n");
       printf("Inizio processo padre\n");
       /*Si posiziona all'inizio del file da leggere*/
       if (lseek(oufile, 0, SEEK_SET) == -1)
       {perror( "lseek errore" );
          exit( -1 ); }
       /*legge i primi 20 caratteri*/
       if ( (nread = read(oufile, buf, WRITELENGTH)) <= 0) {
          close(oufile);
          exit(1);}
       /*scrive i primi 20 caratteri*/
       if ( (write(STDOUT_FILENO, buf, WRITELENGTH)) <= 0) {
          close(oufile);
          exit(1);}
       printf("Fine processo padre\n");
    }
    close(oufile);
    exit(0);
}
```

- Fine Esercitazione -