

System call: esercitazione su creazione, terminazione, esecuzione

Roberto De Virgilio

Sistemi operativi - 6 Dicembre 2017

Esercizio I (warm-up)

- ✿ Scrivere **stampa.c**, un programma in Linguaggio C, per il controllo dei processi in cui un processo padre genera un processo figlio:
- ✿ Il processo **padre**, senza aspettare la conclusione del processo figlio, stamperà su output il messaggio “**Sono il processo padre**” e poi uscirà dal programma (**exit status 0**)
- ✿ Il processo **figlio** stamperà su output il messaggio “**Sono il processo figlio**” e poi uscirà dal programma (**exit status 0**)
- ✿ Nel caso di errore (nella creazione del processo figlio) catturare l'errore stampando un messaggio di errore (si utilizzi la funzione **perror**) e poi uscire dal programma (**exit status 2**)

Esercizio I (warm-up)

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

int main (void) {

    pid_t pid;
    pid = fork();

    if (pid < 0) {
        perror("fork");
        exit(2);
    }

    if (pid == 0) {
        printf("Sono il processo figlio\n");
        exit(0);
    }
    else {
        printf("Sono il processo padre\n");
        exit(0);
    }

}
```

Esercizio I (warm-up)

- **Output:**

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

```
$ ./test
```

Sono il processo padre

Sono il processo figlio

Esercizio 1 bis (warm-up)

- ✿ Scrivere **attesa.c**, un programma in Linguaggio C, per il controllo dei processi in cui un processo padre genera un processo figlio:
- ✿ Il processo **padre** attende la terminazione dei compiti assegnati al processo figlio (stampando in output un messaggio) e poi stampa in output il PID del figlio che ha terminato e il codice di ritorno
- ✿ Il processo **figlio** stampa a standard output il proprio PID e il PID del padre e poi termina (**exit status 24**)
- ✿ Nel caso di errore (nella creazione del processo figlio) catturare l'errore stampando un messaggio di errore (si utilizzi la funzione **perror**) e poi uscire dal programma (**exit status 2**)

Esercizio 1 bis (warm-up)

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

int main (void) {
    pid_t pid;
    int status;

    pid = fork ();

    if (pid < 0) {
        perror("fork");
        exit(2);
    }

    if ( pid == 0 ) {

        printf ("[figlio] Il mio PID è %d e quello del processo
                che mi ha invocato è %d\n" , getpid() , getppid() );
        exit ( 24 );
    }
}
```

Esercizio 1 bis (warm-up)

```
else {
    // segue il codice eseguito dal processo padre

    printf ("[padre] In attesa della terminazione del
            processo figlio...\n");

    // si attende che il processo figlio termini la computazione

    pid = wait ( &status );

    // OK: si sblocca il padre perchè il figlio ha terminato

    printf ("[padre] Il figlio, con PID %d, ha terminato il lavoro
            con il seguente codice di ritorno: %d.\n", pid , status/256 );
}

}
```

Esercizio 1 bis (warm-up)

- **Output:**

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

```
$ ./test
```

```
[padre] In attesa della terminazione del processo figlio...
```

```
[figlio] Il mio PID è 13570 e quello del processo che mi  
ha invocato è 13569
```

```
[padre] Il figlio, con PID 13570, ha terminato il lavoro  
con il seguente codice di ritorno: 24.
```

Esercizio 2

- ◆ Scrivere **dieci_figli.c**, un programma in Linguaggio C, per il controllo dei processi in cui un processo padre generi **dieci processi figli** ed **attenda** il loro termine:
- ◆ Il processo **padre**, stamperà il **codice di terminazione** per ogni processo **figlio** che termina
- ◆ Ogni processo generato, che svolge il compito del **processo figlio** sarà numerato da 1 a 10; dopo una breve attesa (**un secondo**) ogni figlio stamperà un messaggio di presentazione, contenete l'indice del figlio (“**Figlio: #indice**”). A seguire, di nuovo **un secondo** di attesa e poi il processo figlio terminerà dando come codice di terminazione **100 + l'indice**
- ◆ Nel caso di errore (nella creazione del processo figlio) catturare l'errore stampando un messaggio di errore (si utilizzi la funzione **perror**) e poi uscire dal programma (**exit status 2**)

Esercizio 2

- ◆ *Si ricorda:*
 - ◆ *Per fare un ritardo breve in maniera semplice possiamo usare la funzione **sleep**.*
 - ◆ *Per attendere la terminazione di un figlio, si può utilizzare la funzione **wait**.*
 - ◆ *Per conoscere il codice di terminazione potremo usare la macro **WEXITSTATUS(status)**.*

Esercizio 2

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#define NUMP 10          /* Numero di figli da generare */
```

Esercizio 2

```
int main(void) {
    int i,pid;
    // Genera i 10 processi
    for(i=0;i<NUMP;i++){
        pid = fork();

        if (pid < 0) {
            perror("fork");
            exit(2);
    }
```

Esercizio 2

```
if (pid==0) { // figlio
    sleep(1); // Ritardo iniziale
    printf("Figlio: %d\n",i+1); // Stampa messaggio del figlio
    sleep(1); // Ritardo finale
    exit(101+i); // Termina con codice di ritorno

}
else { // padre (il pid e' quello del figlio)
    printf("Ho generato il figlio %d con pid %d\n",i+1,pid);
}
} // fine del for

// Attende che dieci processi terminino

for(i=0;i<NUMP;i++){
    int status;
    wait(&status); // Attende termine di un figlio (uno qualunque)
    printf("Terminato processo %d\n",WEXITSTATUS(status));
}

}// fine del main
```

Esercizio 2

- **Output:**

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

```
$ ./test
```

```
Ho generato il figlio 1 con pid 13381
```

```
... . . .
```

```
Ho generato il figlio 10 con pid 13390
```

```
Figlio: 4
```

```
... . . .
```

```
Figlio: 10
```

```
Terminato processo 104
```

```
... . . .
```

```
Terminato processo 110
```

Esercizio 3

- ◆ Scrivere **scrivi_leggi_file.c**, un programma in Linguaggio C, per il controllo dei processi in cui un processo padre genera un processo figlio:
- ◆ Il processo **padre**
 1. stampa in output il messaggio “[padre] Inserimento stringa nel file”,
 2. poi scrive la stringa “**Sono il padre**” nel file di testo **test.txt** (aperto prima di generare il processo figlio)
 3. stampa in output il valore dell'offset del file (tramite la funzione **fseek**)
 4. stampa in output il messaggio “[padre] chiudo il file e lo cancello” e successivamente chiude e rimuove il file creato; in linguaggio c esiste la funzione **int remove (char* nomefile)**

Esercizio 3

◆ *Il processo figlio*

1. *stampa in output il messaggio “[figlio] attendo 4 secondi”*,
2. *attende 4 secondi e poi stampa in output il valore dell’offset del file (tramite la funzione **fseek**) creato dal processo padre*
3. *riporta il file all’inizio e stampa in output il contenuto del file*
4. *stampa in output il messaggio “[figlio] chiudo il file” e successivamente chiude il file*
5. *uscirà dal programma (**exit status 0**)*

Esercizio 3

- ✿ *Nel caso di errore (nella creazione del processo figlio) catturare l'errore stampando un messaggio di errore (si utilizzi la funzione **perror**) e poi uscire dal programma (**exit status 2**)*

Esercizio 3

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

int main (void) {
    pid_t pid;
    int i;
    char c;
    // si apre il file in scrittura
    FILE *textFile = fopen( "./test.txt" , "w+" );
```

Esercizio 3

```
if ( textFile != NULL ) {
    // si richiama la funzione fork
    // per creare un figlio
    pid = fork ();

    if (pid < 0) {
        perror("fork");
        exit(2);
    }
```

Esercizio 3

```
if ( pid == 0 ) {  
  
    printf ("[figlio] Attendo 4 secondi...\n");  
    sleep ( 4 );  
  
    printf ("\n[figlio] Posizione attuale  
        puntatore file: %ld\n", ftell(textFile) );  
  
    // si riporta all'inizio il cursore  
    fseek(textFile, 0, SEEK_SET);
```

Esercizio 3

```
// si stampa il contenuto testuale del file,  
// carattere per carattere  
  
printf ("[figlio] Stampo contenuto del file di testo: ");  
  
c = fgetc ( textFile );  
while ( ! feof ( textFile ) ) {  
    printf ("%c" , c);  
    c = fgetc ( textFile );  
}  
printf ("\n");
```

Esercizio 3

```
// si chiude il file aperto e si esce dal processo

printf ("[figlio] Chiudo il file ed esco.\n");
fclose ( textFile );
exit ( 0 );

// si nota che, nonostante il processo padre abbia
// già cancellato il file, il processo figlio è
// comunque in grado di leggerne il contenuto!
}
```

Esercizio 3

```
else {
    // sono il padre
    printf ("[padre] Inserimento stringa nel file\n");
    fprintf(textFile, "%s\n", "Sono il padre");

    // si stampa la posizione attuale del cursore sul file

    printf ("[padre] Posizione attuale
            puntatore file: %ld\n", ftell(textFile) );
    // si chiude il file aperto e lo si cancella dal filesystem
    printf ("[padre] Chiudo il file e lo cancello\n");
    fclose ( textFile );
    remove ( "test.txt" );
    } // fine else padre
} // fine else sul controllo del file
else {
    printf ("Errore! Non è stato possibile aprire il file.\n");
}

}
```

Esercizio 3

- **Output:**

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

```
$ ./test
```

```
[padre] Inserimento stringa nel file
```

```
[padre] Posizione attuale puntatore file: 14
```

```
[padre] Chiudo il file e lo cancello
```

```
[figlio] Attendo 4 secondi...
```

```
[figlio] Posizione attuale puntatore file: 14
```

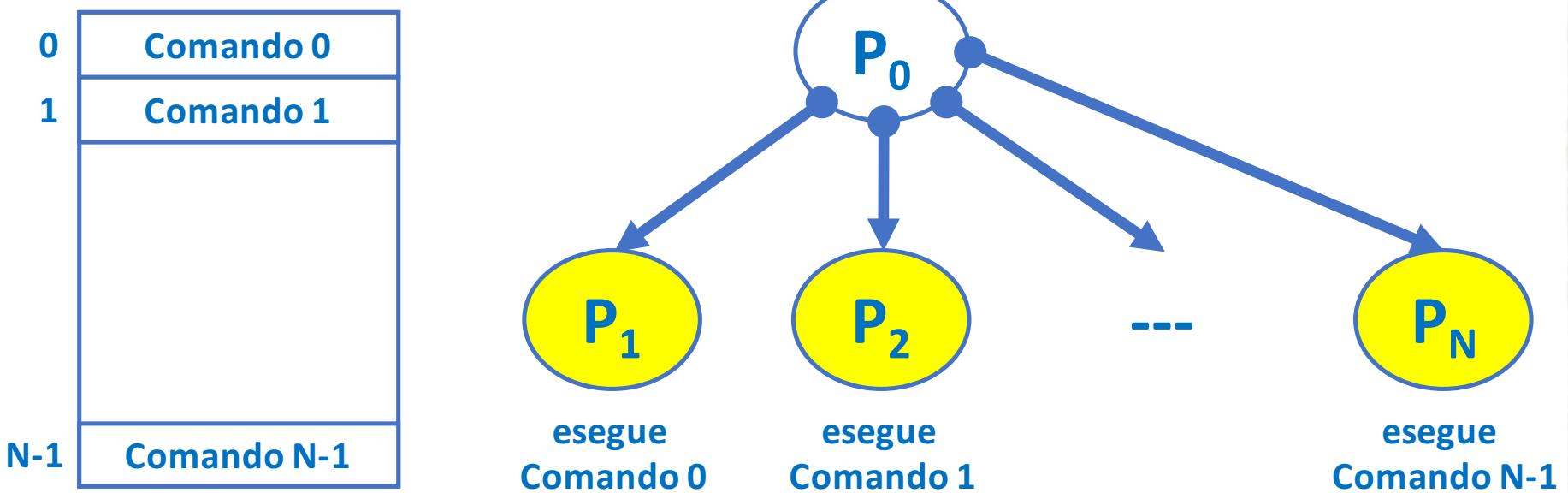
```
[figlio] Stampo contenuto del file di testo: Sono il padre
```

```
[figlio] Chiudo il file ed esco.
```

Esercizio 4

- ◆ Scrivere **sequenza_comandi.c**, un programma in Linguaggio C, in cui un processo padre controlla una sequenza di processi figlio:
- ◆ il processo **padre (Po)** deve acquisire da input una sequenza di **N** comandi (per semplicità, senza argomenti e senza opzioni), con **N** dato da input (**N** al massimo è **20**).
- ◆ una volta letti gli **N** comandi, **Po** deve dare origine ad una sequenza lineare di **N** processi **figlio** (**figlio P₁** - **figlio P₂** - **figlio P₃** - ... - **P_{N+1}**):
 - ◆ ogni processo **figlio** dovrà eseguire un diverso comando della sequenza data e poi terminare (**exit status 0**)
 - ◆ il processo **padre** deve attendere la fine dell'esecuzione di un processo **figlio** per generare il successivo.

Esercizio 4



Esercizio 4

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#define DIM 20
typedef char stringa[80];
typedef stringa strvett[DIM];
int gestoresequenza(int N, strvett vett);
```

Esercizio 4

```
int main(void) {
    int pid,ncom, stato, i;
    strvett vstr;
    printf("quanti comandi? ");
    scanf("%d", &ncom);
    for(i=0; i<ncom; i++) {
        printf("\ndammi il prossimo
               comando(senza argomenti)");
        scanf("%s", vstr[i]);
    }
    gestoresequenza(ncom,vstr);
    exit(0);
}
```

Esercizio 4

```
int gestoresequenza(int N, strvett vett) {
    int stato;
    int pid, i;
    for (i = 0; i < N; i++){
        printf("Comando %s con indice %d\n",vett[i],i);
        pid=fork();

        if (pid < 0) {
            perror("fork");
            exit(2);
        }

        if (pid == 0) { // figlio
            system(vett[i]);
            exit(0);
        }

        else { // padre
            pid = wait(&stato);
        }
    }
}
```

Esercizio 4

- **Output:**

```
$ gcc sequenza_comandi.c -o test
$ ./test
quanti comandi? 2
dammi il prossimo comando (senza argomenti) pwd
dammi il prossimo comando (senza argomenti) ls
Comando pwd con indice 0
/Users/rdevirgilio/SO_2017_2018/Lezioni
Comando ls con indice 1
attesa.c      dieci_figli.c      sequenza_comandi.c
```

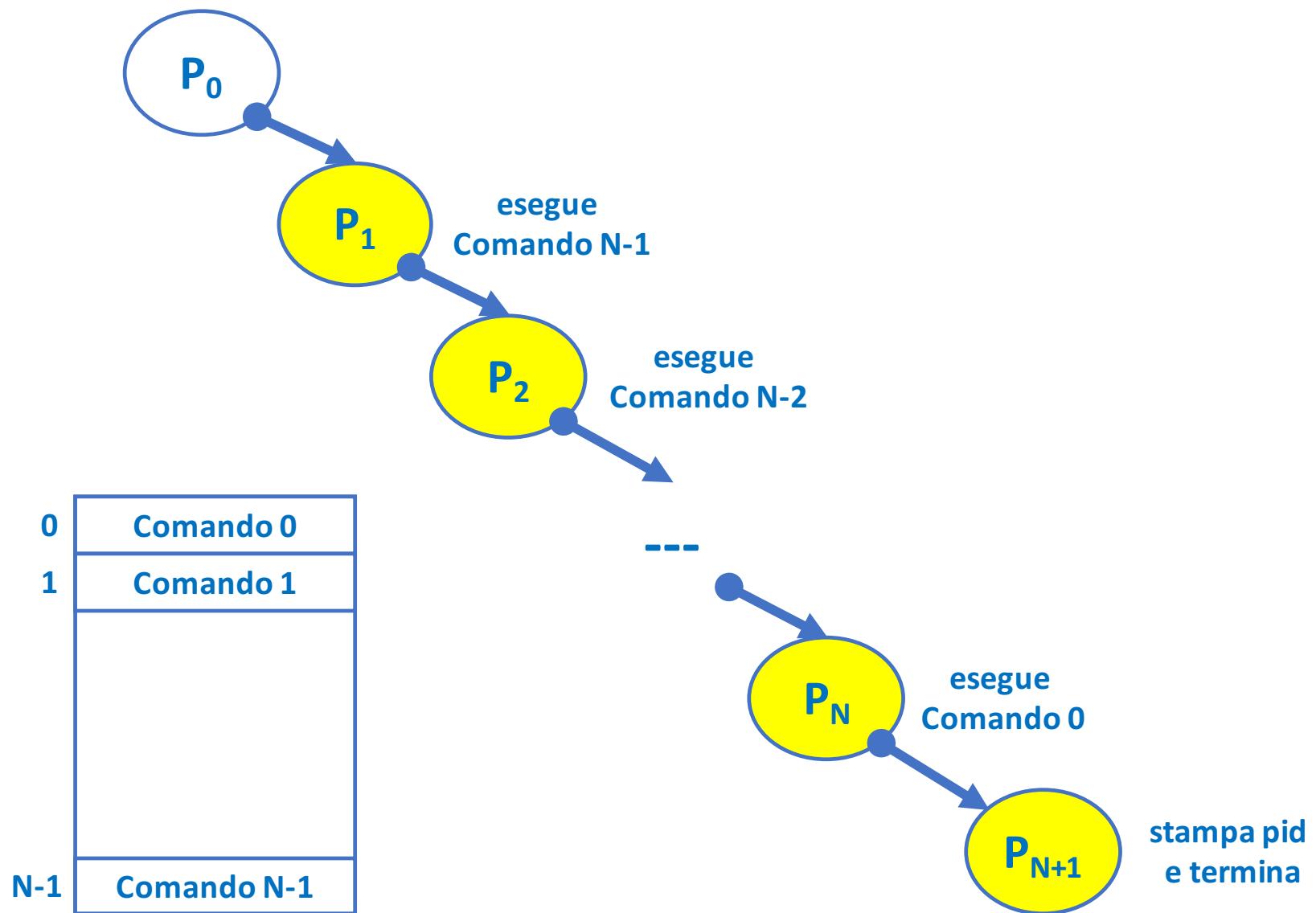
Esercizio 5

- ◆ Scrivere **gerarchia_comandi.c**, un programma in Linguaggio C, per il controllo di una sequenza di processi:
- ◆ il processo iniziale (**P₀**) deve acquisire da input una sequenza di **N** comandi (per semplicità, senza argomenti e senza opzioni), con **N** dato da input (**N** al massimo è **20**).

Esercizio 5

- ◆ una volta letti gli **N** comandi, il processo iniziale **P₀** deve dare origine ad una gerarchia lineare di processi di profondità **N+1** (figlio **P₁** - nipote **P₂** - bisnipote **P₃** - ... - **P_{N+1}**):
- ◆ ogni processo della gerarchia dovrà eseguire un diverso comando della sequenza data (si stampi su output informazioni riguardo il pid e il nome del comando che si sta per eseguire);
- ◆ l'ultimo processo(**P_{N+1}**) della gerarchia, invece, dovrà semplicemente stampare il suo pid e terminare (**exit status 0**)

Esercizio 5



Esercizio 5

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#define DIM 20
typedef char stringa[80];
typedef stringa strvett[DIM];
int gestoresequenza(int N, strvett vett);
void get_stato(int S, int pid);
```

Esercizio 5

```
int main(void) {
    int pid,ncom, stato, i;
    strvett vstr;
    printf("quanti comandi? ");
    scanf("%d", &ncom);
    for(i=0; i<ncom; i++) {
        printf("\ndammi il prossimo comando(senza
               argomenti)");
        scanf("%s", vstr[i]);
    }
    gestoresequenza(ncom-1,vstr);
    pid=wait(&stato);
    get_stato(stato,pid);
    exit(0);
}
```

Esercizio 5

```
void get_stato(int S, int pid) {
    printf("\nterminato processo n.%d", pid);
    if (S==0)
        printf(" terminazione volontaria
                con stato %d\n", S);
    else
        printf(" terminazione involontaria
                per segnale %d\n", S);
}
```

Esercizio 5

```
int gestoresequenza(int N, strvett vett) {
    int stato;
    int pid;
    pid=fork();

    if (pid < 0) {
        perror("fork");
        exit(2);
    }

    if (pid==0) /* figlio*/ {
        if (N==1) /* processo foglia */ {
            printf("\nfoglia %d \n", getpid());
            exit(0);
        }
        else /*attivazione di un nuovo comando*/ {
            pid=gestoresequenza(N-1, vett);
        }
    } // fine else figlio
```

Esercizio 5

```
else { /* padre */
    if (N != -1){
        printf("\nProcesso %d per comando %s", pid, vett[N]);
        pid=wait(&stato);
        get_stato(stato,pid);

        if (execlp(vett[N], vett[N], (char *)0) < 0) {
            perror("exec");
            exit(-1);
        }
        exit(0);
    }
} // fine else padre

}
```

Esercizio 5

- **Output:**

```
$ gcc gerarchia_comandi.c -o test
$ ./test
quanti comandi? 2
dammi il prossimo comando (senza argomenti) pwd
dammi il prossimo comando (senza argomenti) ls
foglia 888
terminato processo n.888 terminazione volontaria con stato 0
Processo 887 per comando pwd
terminato processo n.887 terminazione volontaria con stato 0
/Users/rdevirgilio/SO_2017_2018/Lezioni
Processo 886 per comando ls
terminato processo n.886 terminazione volontaria con stato 0
attesa.c      dieci_figli.c      sequenza_processi.c
```



```
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 29e 30 31 01 02 03 04   mar 04 05 06 07 08 09 10
  05 06 07 08 09 10e 11    11e 12 13 14 15 16 17
  12 13 14 15 16 17 18    18 19 20 21 22 23 24
  19 20 21e 22e 23 24 25   25 26 27 28 29 30 31
mar 26 27 28 29 01 02 03   apr 01e 02 03 04 05 06e 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
```

