

# System call: esercitazione su gestione processi multipli

---

*Roberto De Virgilio*

*Sistemi operativi - 18 Dicembre 2017*

# Esercizio 1

- ◆ Scrivere **myshell.c**, un programma in Linguaggio C, per il controllo dei processi in cui si simuli **una shell**:
- ◆ Il processo **padre** invoca un **ciclo infinito** in cui stamperà il messaggio **"myshell#"** e poi attenderà la digitazione di un comando linux (senza argomenti); a seguire sarà generato un processo **figlio**
- ◆ il **processo figlio** generato eseguirà il comando e terminerà
- ◆ Il processo **padre** attende la terminazione del processo figlio; quindi itera il procedimento
- ◆ l'utente chiuderà la shell digitando **"CTRL+C"**.

# Esercizio 1

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

int main (void) {

    pid_t esito;
    int status;
    char comando[128];
    while(1) {
        printf("myshell# ");
        scanf("%s", comando); //lettura rudimentale: niente argomenti separati
        if ((esito=fork()) < 0)
            perror("fallimento fork");
        else
            if (esito == 0) {
                execlp(comando,comando,NULL); // NOTA: non gestisce argomenti
                perror("Errore esecuzione:");
                exit(0);
            }
            else{ // codice genitore
                while( wait(&status) != esito ); // aspetta completamento
            }
        // il processo genitore (shell) torna immediatamente
        // a leggere un altro comando
    }
}
```

# Esercizio 1

- **Output:**

```
$ gcc myshell.c -o myshell
$ ./myshell
myshell# ls
myshell.c myshell test
myshell# pwd
/Users/rdevirgilio
myshell# ^C
$
```

# Esercizio 2

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

`./listaTreni <treno1> <treno2> ... <trenoN>`

- ✿ tale programma prende da riga di comando varie stringhe **<treno>** costituite da 6 caratteri in cui
  - ✿ primi due caratteri indicano **<tipotreno>** (tipi ammessi “**ic**”, “**es**”, “**rg**”)
  - ✿ ultimi quattro caratteri indicano **<identificativotreno>** (es. **4567**)
- ✿ per ogni treno il processo **padre** genera un processo **figlio** che deve verificare la **validità** del treno che deve essere conforme al formato previsto (**lunghezza 6, primi due caratteri indicano tipo ammesso, ultimi 4 caratteri sono un numero**); il processo figlio stamperà su standard output un messaggio a riguardo (es. “**<trenoI> valido**” o “**<trenoI> non valido**”)
- ✿ Il processo **padre** deve attendere la terminazione del processo figlio prima di procedere al treno successivo
- ✿ Alla fine il processo **padre** stamperà su standard output gli **identificativi** dei soli treni **validi** (raggruppati per **tipo**)

# Esercizio 2 (simulazione)

- **Output:**

```
$ gcc listaTreni.c -o listaTreni
```

```
$ ./listaTreni ic7654 ic8977 es7899 rg56
```

Codice di treno valido: ic7654

Codice di treno valido: ic8977

Codice di treno valido: es7899

Codice di treno non valido: rg56

ic inseriti: 7654 8977

es inseriti: 7899

# Esercizio 2

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

int test_train(char* T){

    if(strlen(T) != 6 ) {
        return 0;
    }
    char tipo[3];
    strncpy(tipo,T,3);
    tipo[2] = '\0';
    if(strcmp(tipo,"ic") !=0 &&
       strcmp(tipo,"es") !=0 && strcmp(tipo,"rg") !=0) {
        return 0;
    }
    return 1;
}
```

## Esercizio 2

```
int main (int argc, char* argv[]){  
  
    if(argc <= 1) {  
        printf("usage: %s <treno1> ... <trenoN>\n", argv[0]);  
        exit(-1);  
    }  
  
    char tipi [3][3]= {"ic","es","rg"};  
    int ntipi[3] = {0,0,0};
```

# Esercizio 2

```
int i, status;
for ( i = 1; i < argc ; i++){

    int pid = fork();
    if (pid > 0) { /* processo padre */
        wait(&status);
        if (WEXITSTATUS(status)==1){
            printf("Codice di treno valido: %s\n",argv[i]);
            switch (argv[i][0]) {
                case 'i': ntipi[0]++; break;
                case 'e': ntipi[1]++; break;
                case 'r': ntipi[2]++; break;
                default: ;
            }
        }
        else {
            printf("Codice di treno non valido: %s\n",argv[i]);
        }
    }
    else {
        if (pid == 0) {
            exit(test_train(argv[i]));
        }
    }
}
```

# Esercizio 2

```
char treni [argc-1][7];
for ( i = 0; i<argc-1 ; i++){
    strcpy(treni[i], argv[i+1]);
}

int t;
for(t = 0; t < 3;t++){
    if (ntipi[t] > 0) {
        printf("%s inseriti: ",tipi[t]);
        for ( i = 0; i<argc-1 ; i++){
            char tipo [3];
            char codice [5];
            strncpy(tipo,treni[i],2);
            tipo[2]= '\0';
            int k;
            for( k= 2; k<strlen(treni[i]); k++){
                codice[k-2] = treni[i][k];
            } // fine for
            codice[strlen(treni[i])-2] = '\0';
            if(strcmp (tipo,tipi[t]) == 0){
                printf("\t %s ",codice);
            } // fine if
        } // fine for
    }// fine if
    printf("\n");
} // fine for

} // fine main
```

# Esercizio 2

- **Output:**

```
$ gcc listaTreni.c -o listaTreni
```

```
$ ./listaTreni ic7654 ic8977 es7899 rg56
```

Codice di treno valido: ic7654

Codice di treno valido: ic8977

Codice di treno valido: es7899

Codice di treno non valido: rg56

ic inseriti: 7654 8977

es inseriti: 7899

# Esercizio 3

- ◆ Scrivere **generapassword.c**, un programma in Linguaggio C che chiede da standard input una stringa (*password*) e restituisce su standard output la codifica criptata di tale stringa.
- ◆ Si assuma che la stringa sia criptata con l'algoritmo DES, utilizzato fino a pochi anni fa sui sistemi UNIX. In particolare la password si ottiene con la seguente istruzione:

*cryptedException = crypt(clearPassword , salt)*

- ◆ Dove *crypt* è il comando di sistema UNIX (v. *man crypt*) e *salt* è rappresentato dalla prime due lettere della password in chiaro *clearPassword*.

*crypt ( "zufolando" , "zu" ) => zuC6b2FGSuy7w*

*crypt ( "Carlo" , "Ca" ) => CaDHaJq99wurg*

# Esercizio 3

- ◆ Scrivere **password\_cracking.c**, un programma in Linguaggio C che tenti di individuare la **password** di un utente (password utente di un sistema Unix) sfruttando un **dizionario** di parole comuni (formato testo, in italiano); deve quindi generare una serie di processi in cui:
  - ◆ Esempio **dizionario**:

[...]  
primitivo  
primizia  
primiziale  
primo  
primogenito  
[...]

# Esercizio 3

- ◆ Scrivere **password\_cracking.c**, un programma in Linguaggio C che tenti di individuare la **password** di un utente (password utente di un sistema Unix) sfruttando un **dizionario** di parole comuni (formato testo, in italiano); deve quindi generare una serie di processi in cui:
  - ◆ Il processo **padre**:
    - ◆ chiede all'utente di fornire su standard input la **password cifrata**
    - ◆ invoca i processi **figli** assegnando loro un dizionario da caricare (*dictionary.txt*) e una parte del dizionario da testare (*prima lettera della password cifrata*)
    - ◆ attende la terminazione di tutti i processi **figli** e stampa l'esito della computazione:
      - ◆ "Tutti i processi figli sono terminati!"
      - ◆ Nel caso di password non trovata: "PID = %d – Valore di ritorno = %d \n"
      - ◆ Nel caso di password trovata: "Il processo con PID = %d ha trovato che la password inizia per '%c' \n"

# Esercizio 3

## ◆ I processi **figli**:

- ◆ invocano il programma “**password\_cracking\_crypt**” passando come parametri: la **password criptata**, il **filename del dizionario**, la **lettera di competenza**. Tale programma (da implementare) apre in lettura il dizionario passato, cerca le parole con l’iniziale data e tra queste cerca la decodifica della password criptata data.
- ◆ tutti i processi **figli** stampano l'esito della computazione:
  - ◆ "Figlio #**%d** (con PID = **%d**) incaricato delle parole che iniziano per '**%c**'\n"
  - ◆ Nel caso di password non trovata: "La password non è stata trovata con le parole che iniziano con la lettera '**%c**'\n"
  - ◆ Nel caso di password trovata: "Password TROVATA!: '**%s**' (criptata = **%s**)\n"

# Esercizio 3

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

#define MAX_CHARS 50
#define NUM_CHARS 52

int main (void) {

    pid_t pid[NUM_CHARS], term[NUM_CHARS];
    int exit_state[NUM_CHARS], n;
    char charList[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz",
        charToTry[2], cryptedPassword[MAX_CHARS], asciiFileHandler[MAX_CHARS],
        dictionaryFilename[] = "dictionary.txt";
```

# Esercizio 3

```
/* Inizializzazione di alcune variabili */
n = 0;
charToTry[1] = '\000';

/* Si chiede all'utente di inserire la password in forma criptata.
L'algoritmo utilizzato è il "crypt" di Unix */

printf ("Inserire la password criptata: ");
scanf ("%s", cryptedPassword);

while (n < NUM_CHARS) {
    pid[n] = fork ();

    if ( pid[n] == 0 ) {

        printf ("\nFiglio #%d (con PID = %d) incaricato delle parole
                che iniziano per '%c'\n" , n, getpid() , charList[n]);
        charToTry[0] = charList[n];
        execl("./password_cracking_crypt", "password_cracking_crypt",
               cryptedPassword, charToTry , dictionaryFilename, NULL);
        exit ( -1 );
    }
    else {
        // padre
        n++;
    }
}
```

# Esercizio 3

```
/* Si attende che tutti i processi figli creati portino a termine  
il compito assegnato; si utilizza la funzione "wait" e si memorizza  
il PID del processo terminato e il relativo codice di ritorno.  
*/  
for (n = 0; n < NUM_CHARS; n++) {  
    term [n] = wait (&exit_state [n]);  
}  
  
printf ("\nTutti i processi figli sono terminati!\n\n");  
for (n = 0; n < NUM_CHARS; n++) {  
    if ((exit_state [n] / 256) <= 3 )  
        printf ("PID = %d - Valore di ritorno = %d \n",  
                term [n], exit_state [n] / 256);  
    else  
        printf ("Il processo con PID = %d ha trovato che la password  
                inizia per '%c' \n", term [n], exit_state [n] / 256);  
}  
exit ( 0 );  
}
```

# Esercizio 3

```
/* password_cracking_crypt.c */

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

#define MAX_STRING 50

int main( int argc , char *argv[] ) {
    char passwordSalt[3],
        generatedPasswordToTest [MAX_STRING],
        dictionaryPasswordToTest [MAX_STRING],
        secredPass[MAX_STRING],
        initialChar;

    FILE *dictionaryFile;
```

# Esercizio 3

```
if ( argc == 4 ) {
    /* Ok, ci sono tutti i parametri necessari */
    /* si apre il file del dizionario */

    dictionaryFile = fopen ( argv[3] , "r" );
    if ( dictionaryFile != NULL ) {
        /* si inizializzano alcune variabili */
        strcpy ( sacredPass, argv[1] );
        initialChar = *argv[2];
        passwordSalt[2] = '\000';
```

# Esercizio 3

```
/* lettura dell'intero file delle parole del dizionario */
while ( ! feof ( dictionaryFile ) ) {
    /* lettura del file riga per riga */
    fgets ( dictionaryPasswordToTest , MAX_STRING , dictionaryFile );
    /* controllo se l'iniziale della parola è quella da gestire */
    if ( dictionaryPasswordToTest[0] == initialChar ) {
        /* si toglie il carattere di 'a capo' sovrascrivendolo con il
           carattere 'terminatore di stringa' */
        if (dictionaryPasswordToTest[strlen (dictionaryPasswordToTest)- 1] == '\n') {
            dictionaryPasswordToTest[strlen (dictionaryPasswordToTest)- 1] = '\000';
        }
        /* si crea il "salt" necessario per l'algoritmo di creazione
           della password criptata e la si genera */
        passwordSalt[0] = dictionaryPasswordToTest[0];
        passwordSalt[1] = dictionaryPasswordToTest[1];

        strcpy (generatedPasswordToTest , crypt(dictionaryPasswordToTest, passwordSalt));
    }
}
```

# Esercizio 3

```
/* si controlla se la password appena generata coincide con  
quella passata in input */  
  
if (!strcmp ( sacredPass, generatedPasswordToTest ) ) {  
    printf ("\n*****\n");  
    printf ("Password TROVATA!: '%s' (criptata = %s)\n",  
           dictionaryPasswordToTest , generatedPasswordToTest );  
    printf ("\n*****\n");  
    fclose (dictionaryFile);  
    /* il codice di ritorno è il codice ASCII della lettera iniziale */  
    exit ( initialChar );  
}  
}  
}
```

# Esercizio 3

```
/* il controllo è terminato e non è stata trovata nessuna password che combaci */
    printf ("\nLa password non è stata trovata con le parole
            che iniziano con la lettera '%c'\n" , initialChar);
    fclose (dictionaryFile);
    exit ( 1 );
}
else {
    printf ("\nErrore nella lettura del file! Exit.\n");
    exit(2);
}
else {
    /* Errore: parametri di input errati */
    printf ( "\nUtilizzo: %s password_criptata lettera_iniziale
              descrittore_file_dizionario\n\n" , argv[0] );
    exit(3);
}
```

# Esercizio 3

- **Output:**

```
$ gcc password_cracking_crypt.c -o password_cracking_crypt
```

```
$ gcc password_cracking.c -o password_cracking
```

```
$ ./password_cracking
```

```
Inserire la password criptata: CaDHaJq99wurg
```

```
Figlio #0 (con PID = 10343) incaricato delle parole che iniziano per 'A'
```

```
Figlio #1 (con PID = 10344) incaricato delle parole che iniziano per 'B'
```

```
...
```

```
La password non è stata trovata con le parole che iniziano con la lettera 'A'
```

```
La password non è stata trovata con le parole che iniziano con la lettera 'E'
```

```
Password TROVATA!: 'Carlo' (criptata = CaDHaJq99wurg)
```

```
...
```

```
Tutti i processi figli sono terminati!
```

```
PID = 10346 - Valore di ritorno = 1
```

```
Il processo con PID = 10345 ha trovato che la password inizia per '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 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
```

