LabSO 2025

Laboratorio Sistemi Operativi - A.A. 2024-2025

Michele Grisafi - michele.grisafi@unitn.it

Errori in C

Gestione errori in C

Durante l'esecuzione di un programma ci possono essere diversi tipi di errori: system calls che falliscono, divisioni per zero, problemi di memoria etc...

Alcuni di questi errori non fatali, come una system call che fallisce, possono essere indagati attraverso la variabile **`errno`**. Questa variabile globale contiene l'ultimo codice di errore generato dal sistema.

Per convertire il codice di errore in una stringa comprensibile si può usare la funzione `char *strerror(int errnum)`.

In alternativa, la funzione `void perror(const char *str)` che stampa su stderr la stringa passatagli come argomento concatenata, tramite ': ', con `strerror(errno)`.

Esempio: errore apertura file

```c
#include <stdio.h> <errno.h>  <string.h>                //errFile.c

extern int errno; // declare external global variable
int main(void){
    FILE * pf;
    pf = fopen("nonExistingFile.boh", "rb"); //Try to open file
    if (pf == NULL) { //something went wrong!
        fprintf(stderr, "errno = %d\n", errno);
        perror("Error printed by perror");
        fprintf(stderr,"Strerror: %s\n", strerror(errno));
    } else {
        fclose (pf);
    }
}
```

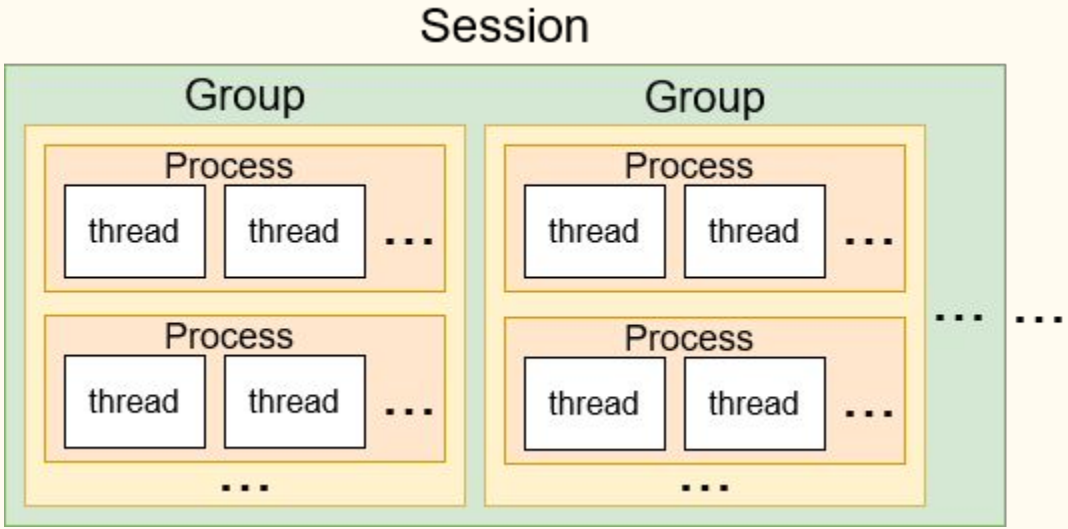
Esempio: errore processo non esistente

```c
#include <stdio.h> <errno.h>  <string.h> <signal.h>     //errSig.c
extern int errno; // declare external global variable
int main(void){
    int sys = kill(3443,SIGUSR1); //Send signal to non existing proc
    if (sys == -1) { //something went wrong!
        fprintf(stderr, "errno = %d\n", errno);
        perror("Error printed by perror");
        fprintf(stderr,"Strerror: %s\n", strerror(errno));
    } else {
        printf("Signal sent\n");
    }
}
```

Process groups

Gestione processi in Unix

All’interno di Unix i processi vengono raggruppati secondi vari criteri, dando vita a sessioni, gruppi e threads.



Perchè i gruppi

I process groups consentono una migliore gestione dei segnali e della comunicazione tra i processi.

Un processo, per l'appunto, può:

- Aspettare che tutti i processi figli appartenenti ad un **determinato gruppo** terminino;
- Mandare un segnale a tutti i processi appartenenti ad un **determinato gruppo.**

```
waitpid(-33,NULL,0); // Wait for a child in group 33 (|-33|)
kill(-45,SIGTERM); // Send SIGTERM to all children in group 45
```

Gruppi in Unix

Mentre, generalmente, una sessione è collegata ad un terminale, i processi vengono raggruppati nel seguente modo:

- In bash, processi concatenati tramite pipes appartengono allo stesso gruppo: `cat /tmp/ciao.txt | wc -l | grep ‘2’`
- Alla loro creazione, i figli di un processo ereditano il gruppo del padre. Se il padre cambia il proprio gruppo, i figli rimangono con il gruppo originario!
- Inizialmente, tutti i processi appartengono al gruppo di ‘init’, ed ogni processo può cambiare il suo gruppo in qualunque momento. Il nuovo gruppo deve:
  - Esistere e far parte della stessa sessione del processo
  - Oppure essere uguale al PID del processo chiamante

Il processo il cui PID è uguale al proprio GID è detto *process group leader*.

Group system calls

```
int setpgid(pid_t pid, pid_t pgid); //set GID of proc. (0=self)
pid_t getpgid(pid_t pid); // get GID of process (0=self)
```

```c
#include <stdio.h> <unistd.h> <sys/wait.h>                    //setpgid.c
int main(void){
    int isChild = !fork(); //new child
    printf("PID %d PPID: %d GID %d\n",getpid(),getppid(),getpgid(0));
    if(isChild){
        setpgid(0,0); // Become group leader
        printf("PID %d PPID: %d GID %d\n",getpid(),getppid(),getpgid(0));
    };
    while(wait(NULL)>0);
}
```

Ereditare un gruppo

```c
#include <stdio.h> <unistd.h> <sys/wait.h>                    //setpgid.c
int main(void){
    int isChild = !fork(); //new child
    printf("PID %d PPID: %d GID %d\n",getpid(),getppid(),getpgid(0));
    if(isChild){
        printf("PID %d setting my own pid\n",getpid());
        setpgid(0,0); // Become group leader
        printf("PID %d creating child\n",getpid());
        isChild = !fork();
        if(isChild) printf("PID %d inherited group %d\n",getpgid(0));
    };
    while(wait(NULL)>0);
}
```

Inviare i segnali: kill()

`int kill(pid_t pid, int sig);` NB: ordine degli argomenti!

Invia un segnale ad uno o più processi a seconda dell'argomento *pid:*

- `pid > 0`: segnale al processo con `PID=pid`
- `pid = 0`: segnale ad ogni processo dello stesso gruppo
- `pid = -1`: segnale ad ogni processo possibile (stesso UID/RUID)
- `pid < -1`: segnale ad ogni processo del gruppo `|pid|`

Mandare segnali ai gruppi

Nel prossimo esempio:

1. Processo ‘ancestor’ crea un figlio
   a. Il figlio cambia il proprio gruppo e genera 3 figli (4 proc. nel **Gruppo1**)
   b. I 4 processi aspettano fino all’arrivo di un segnale
2. Processo ‘ancestor’ crea un secondo figlio
   a. Il figlio cambia il proprio gruppo e genera 3 figli (4 proc. nel **Gruppo2**)
   b. I 4 processi aspettano fino all’arrivo di un segnale
3. Processo ‘ancestor’ manda due segnali diversi ai due gruppi

Mandare segnali ai gruppi

```
#include <stdio.h><unistd.h><sys/wait.h><signal.h><stdlib.h>//gsignal.c
void handler(int signo){
    printf("[%d,%d] sig%d received\n",getpid(),getpgid(0),signo);
    sleep(1); exit(0);
}
int main(void){
    signal(SIGUSR1,handler);
    signal(SIGUSR2,handler);
    int ancestor = getpid(); int group1 = fork(); int group2;
    if(getpid()!=ancestor ){ // First child
        setpgid(0,getpid()); // Become group leader
        fork();fork(); //Generated 3 children in new group
    }
...
```

```
    else{
        group2 = fork();
        if(getpid()!=ancestor){ // Second child
            setpgid(0,getpid()); // Become group leader
            fork();fork();}} //Generated 3 children in new group
    if(getpid()==ancestor){
        printf("[%d]Ancestor and I'll send signals\n",getpid());
        sleep(1);
        kill(-group2,SIGUSR2); //Send SIGUSR2 to group2
        kill(-group1,SIGUSR1); //Send SIGUSR1 to group1
    }else{
        printf("[%d,%d]chld waiting signal\n", getpid(),getpgid(0));
        while(1);
    }
    while(wait(NULL)>0);
    printf("All children terminated\n");
}
```

Aspettare un gruppo

`pid_t waitpid(pid_t pid, int *status, int options)`

Consente un'attesa selettiva basata su dei parametri. `pid` può essere:

- `-n` (aspetta un figlio qualsiasi nel gruppo `|-n|`)
- `-1` (aspetta un figlio qualsiasi)
- `0` (aspetta un figlio qualsiasi appartenente allo stesso gruppo)
- `n` (aspetta il figlio con `PID=n`)

`options` sono i seguenti parametri ORed:

- `WNOHANG`: ritorna immediatamente se nessun figlio è terminato → non si resta in attesa!
- `WUNTRACED`: ritorna anche se un figlio si è interrotto senza terminare.
- `WCONTINUED`: ritorna anche se un figlio ha ripreso l'esecuzione.

Wait figli in un gruppo

Nel prossimo esempio:

1. Processo ‘ancestor’ crea un figlio
   a. Il figlio cambia il proprio gruppo e genera 3 figli (Gruppo1)
   b. I 4 processi aspettano 2 secondi e terminano
2. Processo ‘ancestor’ crea un secondo figlio
   a. Il figlio cambia il proprio gruppo e genera 3 figli (Gruppo2)
   b. I 4 processi aspettano 4 secondi e terminano
3. Processo ‘ancestor’ aspetta la terminazione dei figli del gruppo1
4. Processo ‘ancestor’ aspetta la terminazione dei figli del gruppo2

Wait figli in un gruppo

```
#include <stdio.h><unistd.h><sys/wait.h>                    //waitgroup.c
int main(void){
    int group1 = fork(); int group2;
    if(group1 == 0){ // First child
        setpgid(0,getpid()); // Become group leader
        fork();fork(); //Generated 4 children in new group
        sleep(2); return; //Wait 2 sec and exit
    }else{
        group2 = fork();
        if(group2 == 0){
            setpgid(0,getpid()); // Become group leader
            fork();fork(); //Generated 4 children
            sleep(4); return; //Wait 4 sec and exit
        } ...
```

```
    }
    sleep(1); //make sure the children changed their group
    while(waitpid(-group1,NULL,0)>0);
    printf("Children in %d terminated\n",group1);
    while(waitpid(-group2,NULL,0)>0);
    printf("Children in %d terminated\n",group2);
}
```

CONCLUSIONI

L’organizzazione dei processi in gruppi consente di organizzare meglio le comunicazione e di coordinare le operazioni avendo in particolare la possibilità di inviare dei segnali ai gruppi nel loro complesso.