Processi e Sistema Operativo

Federico Zotti

2021-11-25

Indice

Processo

All'esecuzione di un programma, vengono allocati 3 spazi:

- il code segment: Istruzioni (quasi sempre *read only*);
- il data segment: Variabili globali, statiche, dinamiche (*RW*);
- lo stack segment: Variabili locali, informazioni di stato, tutti i registri durante il *context switch* (*RW*).

Il code segment è fisso e per cambiare il processo deve chiedere al sistema operativo di sostituire il proprio codice con il contenuto di un altro file. Il sistema opertivo può rifiutare al chiamata di sistema e terminare il processo.

Stati di un processo

- **Ready**: aspettando l'esecuzione (gestito dallo *scheduler*);
- **Exec**: in esecuzione;
- **Locked**: aspettando la risposta ad una chiamata di sistema;
- **Zombie**: se rimane dopo che è stato terminato il suo genitore.

Handler

Un'*handler* è una funzione che il processo associa ad un evento particolare.

Wait

La funzione *wait* permette di aspettare il cambio di stato di un processo processo figlio. Con *waitpid* è possibile specificare il PID del processo.

RAM

Memoria virtuale: RAM + Swap.

Tutta la memoria disponibile per il sistema è la memoria virtuale. È la somma della RAM fisica e dello *swap* sul disco. Lo swap è una porzione di disco usata come se fosse RAM.

La memoria virtuale viene gestita dalla MMU (parte interna al processore), che trasforma gli indirizzi virtuali in indirizzi fisici. Se un processo tenta di scrivere in un code segment, la MMU avvia un interrupt che informa l'OS dell'accaduto, il quale poi lo uccide.

Per allocare memoria, si utilizza la funzione `malloc`, per liberarla `free`.

Malloc

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

int main()
{
    int dim = 5;    //dimensione del buffer allocato in byte
    char * buffer;  //indirizzo del puntatore
    buffer = (char * ) malloc(dim); //alloca il buffer
    printf("%i\n", buffer);
}
```

Se `dim` è maggiore della quantità di memoria disponibile non viene allocato nessun buffer e la variabile `buffer` conterra 0.

Fork

Per generare un processo figlio si utilizza la chiamata di sistema `clone`. Per facilitarne l'utilizzo si può usare la libreria `unistd.h` con la funzione `fork`.

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

int main () {
    int pid;
    pid = fork();
    printf("pid = %d", pid);
}
```

Il valore di pid equivale al pid del processo per il padre e 0 per il processo figlio.

È più efficiente creare solo 4 processi e utilizzare solo quelli.

```
ncpu => nfigli
100k / nfigli -> blocco per figlio
```

Intra-Process Communication

Due thread appartenenti allo stesso processo possono comunicare tra di loro usando l'*intra-process communication*.

Threads (di un processo)

hpc-tutorials.llnl.gov/posix/

Segmenti in comune:

- Code segment;
- Data segment;
- Stack segment

  .

Ogni processo è composto da almeno un thread.

La non condivisione dello stack segment permette di avere variabili locali diverse.

La funzione che viene gestita dal thread si chiama *watchdog* o *callback*.

Posix threads (pthreads)

Questi sono l'implementazione su sistemi posix (linux, unix).

È necessario linkare esplicitamente la libreria con gcc:

```
1 gcc -pthread codice.c
```

Chiamata di sistema

Si dividiono in due categorie:

- Bloccanti (*read*);
- Non bloccanti (*write*).

Inter-Process Communication

Comunicazione tra thread appartenenti a processi diversi tramite **Socket**.