Altre funzioni

sabato 8 aprile 2023 13:33

Terminazione in applicazioni multi-thread su UNIX

Ambienti a processi

`void exit(int)` ⟹ Terminazione dell' unico thread attivo, e quindi dell'intero processo

Ambienti multi-thread

`void exit(int)` ⟹ Terminazione del thread corente, non necessariamente dell'intero processo

`void exit_group(int)` ⟹ Terminazione dell'intero processo

Quindi chiamiamo syscall se vogliamo chiamare exit solamente e non exit_group. Perché?

`exit()` e' mappata su `exit_group()` in `stdlib`

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


void* child_thread(void*p){

    again:
    sleep(1);
    printf("I'm alive\n");
    goto again;
}

int main(int argc, char** argv){

   pthread_t tid;

   if(pthread_create(&tid,NULL,child_thread,NULL) != 0){
    printf("pthread_create error\n");
    fflush(stdout);
    exit(EXIT_FAILURE);
   }

   syscall(60,0);

}

```

```
OCESSES-AND-THREADS/UNIX> ./a.out
I'm alive
I'm alive
I'm alive
I'm alive
I'm alive
I'm alive
I'm alive
I'm alive
```

L'output è solo del thread_child, il main_thread ha completato la sua esecuzione.

Abbiamo un main che lancia un thread e poi abbiamo un syscall(60,0);
E il thread che viene ad essere lanciato ha una label "again", manda in print che è vivo, e poi ritorna con goto alla label.

Quindi child_thread non termina mai.
Ma chi è che sta chiamando syscall(60,0)?
Il thread main.
Bene, stiamo chiamando il servizio-la system call con codice 60.
E se noi andassimo sul WEB a cercare la tabella delle system_call che potremmo trovare su : https://filippo.io/linux-syscall-table/

50	listen	sys_listen	net/socket.c
51	getsockname	sys_getsockname	net/socket.c
52	getpeername	sys_getpeername	net/socket.c
53	socketpair	sys_socketpair	net/socket.c
54	setsockopt	sys_setsockopt	net/socket.c
55	getsockopt	sys_getsockopt	net/socket.c
56	clone	stub_clone	kernel/fork.c
57	fork	stub_fork	kernel/fork.c
58	vfork	stub_vfork	kernel/fork.c
59	execve	stub_execve	fs/exec.c
60	exit	sys_exit	kernel/exit.c
61	wait4	sys_wait4	kernel/exit.c
62	kill	sys_kill	kernel/signal.c
63	uname	sys_newuname	kernel/sys.c
64	semget	sys_semget	ipc/sem.c
65	semop	sys_semop	ipc/sem.c
66	semctl	sys_semctl	ipc/sem.c
67	shmdt	sys_shmdt	ipc/shm.c
68	msgget	sys_msgget	ipc/msg.c

Chiamata esplicita e selettiva delle system call UNIX

Supportata tramite il costrutto `syscall(…)`

Output:
Syscall return-value

Input:
Syscall num, Arg 0, Arg 1 ….

Codice numerico che identifica la specifica system call da chiamare

Se avessimo scritto davvero exit()

non saremmo mai usciti dal main_thread, perché la exit internamente chiama una exit_group() e quindi stiamo facendo uscire l'intero processo, ossia l'intera applicazione.

Librerie rientranti e non

- Un libreria si definisce rientrante se offre servizi **"thread safe"**
- Questo implica la possibilita' di usare la libreria in ambiente multi-thread senza problemi di consistenza sullo stato della libreria stessa (dovuti alla concorrenza)
- Consultare sempre il manuale per verificare se la funzione di libreria che si intende utilizzare e' rientrante o non

- Le funzioni della versione rientrante sono in genere identiche in specifica di interfaccia a quelle non rientranti, ma hanno il suffisso `_r` nella segnatura)

Spesso le librerie non rientrati costituiscono una grande sicurezza, perché magari chi attacca con più thread è anche peggio.