Esempio Thread Local Storage

venerdì 7 aprile 2023 22:26

ESEMPIO

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

int* aux;//pointer used to update the stack of a different thread by the interfering_child_thread function

void* child_thread(void*p){

    int c = 1;
    aux = &c;

    while(1){
        printf("variable c has value: %d\n",c);
        sleep(2);
    }
}


void* interfering_child_thread(void*p){

    int c;

    while(1){
        scanf("%d",&c);
        *aux = c;
    }

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

    pthread_t tid;

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


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

    pause();
}

```

In questa applicazione abbiamo due thread e abbiamo una stack area per il thread 1 e una stack area per il thread 2, all'interno dell'address space.
Uno dei due va a comunicare all'interno delle informazioni (che è una variabile globale) l'indirizzo di memoria di una sua variabile locale, che è ovviamente all'interno della stack area, e l'altro thread può usare questa informazione per accedere alla stack area di questo.

Questo ci fa capire che questi thread vivono all'interno dello stesso address space.

Abbiamo la funzione child_thread dove vivrà un thread e la funzione interfering_child_thread dove ne vivrà un altro.

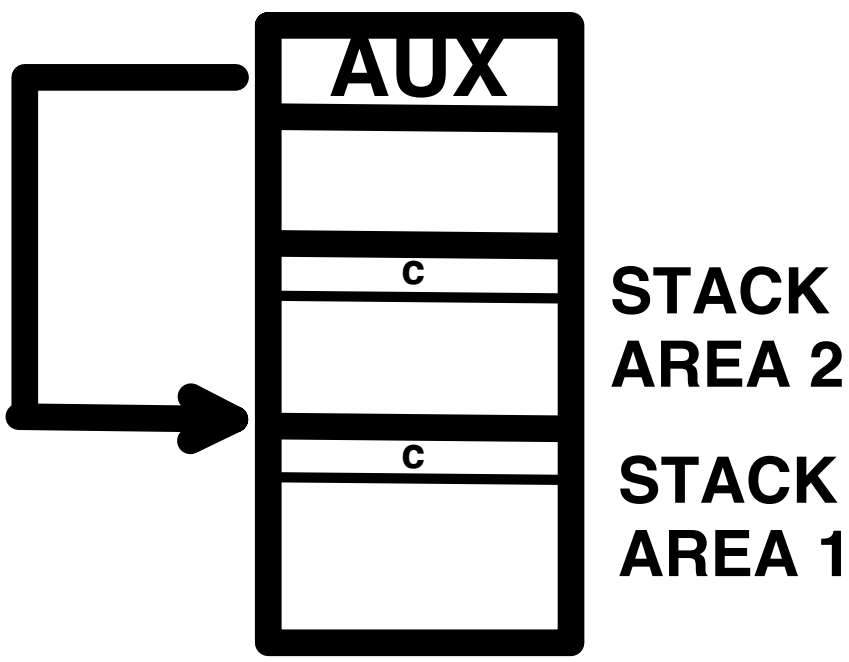
Nel main generiamo prima uno dei due thread e poi l'altro.
Il primo thread vivrà nella funzione child_thread() e il secondo in interfering_child_thread().

Passiamo null ad entrambe le funzioni come parametri, prendiamo il codice numerico del primo e nella stessa variabile sovrascriviamo con il codice numerico dell'altro, e andiamo in pausa.

Il child_thread non fa altro che avere una variabile locale che è c, inizializzata ad 1. E poi una variabile globale aut, che è un puntatore ad intero va a scrivere l'indirizzo di memoria di questa variabile.

Interfering_child_thread in un while 1 va a prendere un intero e lo scrive sulla variabile locale c e questa funzione sta lavorando in un altra stack area, così come la variabile c è in questa stack_area. Dopo la scanf andiamo ad accedere al contenuto di aux con *aux, che sta puntando alla variabile c = 1 della funzione precedente sull'altra stack area, e scriviamo lì la nostra nuova c.

Stiamo cambiando il valore di una variabile locale di un altro thread (child_thread) con il valore della variabile locale c in interferring_child_thread().




```
ND-THREADS/UNIX> gcc threads-interference.c -lpthread
 AND-THREADS/UNIX> ./a.out
 variable c has value: 1
 variable c has value: 1
 variable c has value: 1
 variable c has value: 1
 78variable c has value: 1
 variable c has value: 1

 variable c has value: 78
 variable c has value: 78
 variable c has value: 78
 variable c has value: 78
 variable c has value: 78
 variable c has value: 78
```