4' Esercitazione

https://politecnicomilano.webex.com/meet/gianenrico.conti

Gian Enrico Conti

Threads

Esercizi:

- creazione, exit e join
- passaggio si argomenti generici mediante void*
- ritorno di valori mediante void*
- problemi legati al ritorno di valori per puntatore a variabili automatiche
- sincronizzazione con semafori binari
- sezioni critiche e mutex

Outline

- **Threads**
  - Creazione – *create*
  - Attesa della terminazione – *join*
  - Sincronizzazione
    - Semafori
    - Mutex

Threads - Creazione (1)

- **In LINUX/UNIX/POSIX**

```
include <pthread.h>
```

- **Il comportamento del thread è determinato da una funzione, ("body")**
- **Molto generica"**
  - **Riceve puntatore a void**
  - **Ritorna un puntatore a void**
  - **Cast necessari (sia nella f. che nel main())**

```
void* thread_function(void* param){
                        …
}
```

Threads - Creazione (2)

- **Ogni thread ha un thread ID di tipo pthread_t:**

```
pthread_t thread_ID;
```

- **Per creare un thread si usa pthread_create, con quattro argomenti:**

```
int pthread_create( pthread_t *thread,
                    const pthread_attr_t *attr,
                    void *(*start_routine)(void*),
                    void *arg);
```

  - Il primo argomento è un puntatore ad una variabile di tipo pthread_t che conterrà l'ID del thread
  - Il secondo argomento è un puntatore ad un oggetto di tipo thread_attribute;
    se NULL il thread viene creato con i suoi parametri di default
  - Il terzo argomento è un puntatore alla thread function;
  - Il quarto argomento di tipo `void*` è il parametro da passare alla thread function.

Thread – Creazione (3)

- **Esempio:**

```c
//EX4_es01
#include <stdio.h>
#include <pthread.h>

void * thread_func (void * p){
    printf("This is the thread\n");
    return NULL;
}

int main(){
    pthread_t t;
    int i;
    pthread_create(&t, NULL, &thread_func,NULL);
    for(i=0;i<10000; i++){
        //do something
    }
}
```

- **proviamo..**

Thread – Creazione (3)

- Su alcune "distro":

```
test.c:(.text+0x7a): undefined reference to `pthread_create'
test.c:(.text+0xc1): undefined reference to `pthread_join'
```

- Manca la lib (NON header..):

```
gcc -Wall -pthread test.c -o test
```

- rilanciamo..

Thread – Creazione (3)

- **NON appare il messaggio.. perche main finisce**

  "temp FIX" ... (seguono dettagli..)

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

void * thread_func (void * p){
    printf("This is the thread\n");
    return NULL;
}

int main(){
    pthread_t t;
    int i;
    pthread_create(&t, NULL, &thread_func,NULL);
    for(i=0;i<10000; i++){
        //do something
    }
    scanf("%d", &i); // keep main alive..
}
```

Thread – simmetria con PID

- **Ogni Thread ha un id**
- **Per conoscere proprio id**

```
pthread_t pthread_self(void);
```

(Esempio piu' avanti.)

Thread vs Process..

1) thread + leggeri
2) I "worker" difficoltoso scambio dati fra processi
3) processi "isolati"
4) context switching +pesante x processi
5) comunicazione inter-task

Thread vs Process (1)

Threads share a common address space, thereby avoiding a lot of the inefficiencies of multiple processes.

- The kernel does not need to make a new independent copy of the process memory space, file descriptors, etc. This saves a lot of CPU time, making thread creation ten to a hundred times faster than a new process creation. Because of this, you can use a whole bunch of threads and not worry about the CPU and memory overhead incurred. This means you can generally create threads whenever it makes sense in your program.
- ...

Thread vs Process

Threads share a common address space, thereby avoiding a lot of the inefficiencies of multiple processes.

- ...
- Less time to terminate a thread than a process.
- Context switching between threads is much faster then context switching between processes (context switching means that the system switches from running one thread or process, to running another thread or process)
- ...

Thread vs Process

Threads share a common address space, thereby avoiding a lot of the inefficiencies of multiple processes.

- ...

On the other hand, because threads in a group all use the same memory space, if one of them corrupts the contents of its memory, other threads might suffer as well. With processes, the operating system normally protects processes from one another, and thus if one corrupts its own memory space, other processes won't suffer.

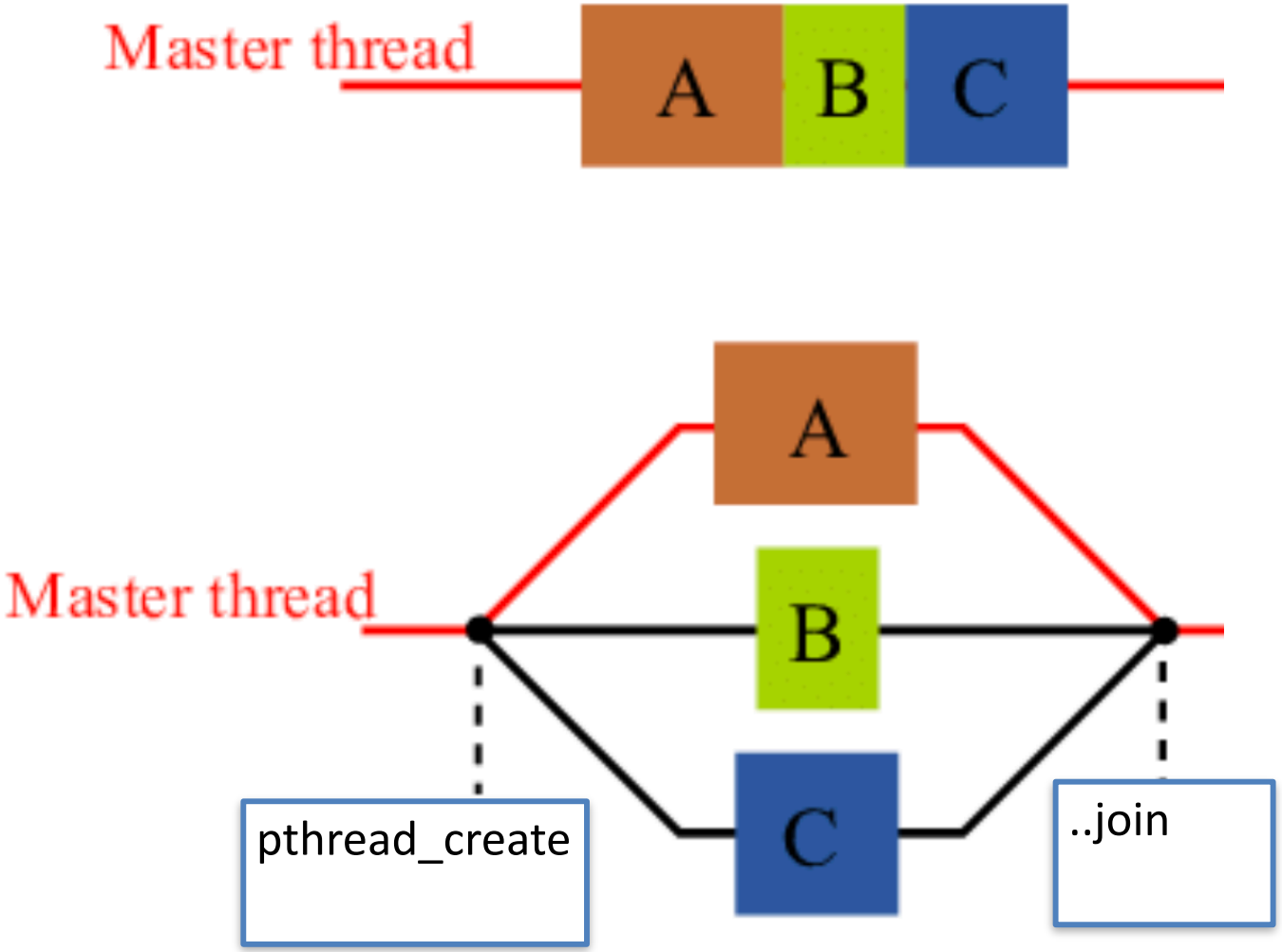
Thread perche' si usano

- UI non bloccata da operazioni lunghe / rete
- sfruttano i core della CPU

...

Thread perche' si usano

Esempio:



Thread perche' si usano: network

Esempio da:

https://www.cs.dartmouth.edu/~campbell/cs50/echoServer.c

Parti salienti:

```
listen(listenfd, LISTENQ);

 printf("%s\n","Server running...waiting for connections.");

 for ( ; ; ) {

  clilen = sizeof(cliaddr);
  connfd = accept(listenfd, (struct sockaddr *) &cliaddr, &clilen);
  printf("%s\n","Received request...");

  while ( (n = recv(connfd, buf, MAXLINE,0)) > 0)  {
   printf("%s","String received from and resent to the client:");
   puts(buf);
   send(connfd, buf, n, 0); // send back...
  }
```

NON gestisce client multipli.. in piu accept && recv sono bloccanti.

Thread – Terminazione e attesa (1)

- **Un thread termina con la funzione** `pthread_exit` **o con la normale** `return`

```
void pthread_exit(void *value_ptr);
```

Il parametro passato alla pthread_exit, opportunamente castato a void*, è il return value del thread

- **Una chiamata a** `exit(int)` **all'interno del thread causa la terminazione del processo padre e di conseguenza di tutti gli altri thread**

- **Se il processo padre termina prima di uno dei suoi thread possono nascere problemi in quanto la memoria cui tali thread fanno accesso viene deallocata e tali thread vengono terminati insieme al padre**

Thread – Terminazione e attesa (2)

- **Per prevenire tale effetto, nonché per attendere un thread all'interno di un altro thread, si usa la funzione**

```
int pthread_join(pthread_t thread, void **value_ptr);
```

Il primo parametro è il thread ID da attendere,
il secondo è un puntatore a puntatore a void che prende il valore di uscita del thread

Se non serve, 2' param usare NULL

Thread – Terminazione e attesa (3)

- **Un thread non dovrebbe mai attendere se stesso, per evitare tale circostanza è opportuno controllare il proprio thread ID attraverso la funzione pthread_self**

- **Per controllare l'uguaglianza di due thread ID si usa la funzione pthread_equal**

```
int pthread_equal(pthread_t t1, pthread_t t2);
```

Thread – Terminazione e attesa

- **Esempio: (EX4_es02.c)**

```c
//EX4_es02
#include <stdio.h>
#include <pthread.h>

void * thread_func (void * p){
    // cast back:
    long n = (long)p;
    printf("This is the thread\n got: %ld\n", n);
    return NULL;
}

int main(){
    pthread_t t1,t2;
    long t_num=1;
    pthread_create(&t1, NULL, &thread_func,(void*)t_num);
    t_num=2;
    pthread_create(&t2, NULL, &thread_func,(void*)t_num);

    pthread_join(t1,(void*)&t_num);
    printf("Received thread %ld\n",t_num);
    pthread_join(t2, (void*)&t_num);
    printf("Received thread %ld\n",t_num);
    return 0;
}
```

- NOTARE cosa stampa.... Provare a ritornare il DOPPIO...

Thread – Terminazione e attesa

- **Esempio: (EX4_es02_B.c)**

```c
void * thread_func (void * p){
    // cast back:
    long n = (long)p;
    printf("This is the thread\n got: %ld\n", n);
    return (void*)(n*2);    // was: return NULL;
}

int main(){
    pthread_t t1,t2;
    long t_num=1;
    pthread_create(&t1, NULL, &thread_func,(void*)t_num);
    t_num=2;
    pthread_create(&t2, NULL, &thread_func,(void*)t_num);

    long ris1, ris2;
    pthread_join(t1,(void*)&ris1);
    printf("Received thread %ld\n",ris1);
    pthread_join(t2, (void*)&ris2);
    printf("Received thread %ld\n",ris2);
    return 0;
}
```

Thread – Terminazione e attesa (2)

- **Esempio: (EX4_es02_C.c)**

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

void * thread_func (void * p){
    // cast back:
    long n = (long)p;
    printf("thread got: %ld\n", n);
    return (void*)(n*2);    // was: return NULL;
}

#define MAX_THREADS 10

int main(){
    pthread_t t[MAX_THREADS];
    long t_num=0;
    for (t_num=0; t_num<MAX_THREADS; t_num++)
        pthread_create(&t[t_num], NULL, &thread_func,(void*)t_num);

    for (t_num=0; t_num<MAX_THREADS; t_num++){
        long ris;
        pthread_join(t[t_num],(void*)&ris);
        printf("Received from thread %ld\n",ris);
    }

    return 0;
}
```

- **Al run si noti la sequenza di stampa di ritorno... join aspetta su un ID in ordine...**

Thread - passaggio parametri: struct/Array

A) non e' possible cambiare il prototipo della thread f.

B) si usa Cast

```c
//EX4_es04
#include <stdio.h>
#include <pthread.h>

typedef struct MyPoint{
    int x,y;
}MyPoint;


void * thread_func_Arr (void * p){
    // cast back to string
    char * s = (char*)p;
    printf("got: %s\n", s);
    return NULL;
}
void * thread_func_Struct (void * p){
    // cast back:
    MyPoint * pp = (MyPoint*)p;
    printf( "got: %d %d \n", pp->x, pp->y);
    return NULL;
}

int main(){
    pthread_t t1,t2;
    char msg[] = "HELLO";
    pthread_create(&t1, NULL, &thread_func_Arr,(void*)msg);

    MyPoint p = {10,20};
    pthread_create(&t2, NULL, &thread_func_Struct,(void*)&p);

    pthread_join(t1,NULL);
    pthread_join(t2,NULL);
    return 0;
}
```

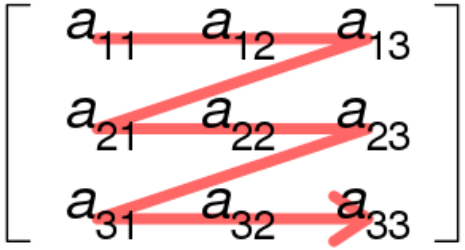
Thread - core/CPU: suddividere carico:

Es matrice molto grande di cui voglio p.es. trovare il massimo:

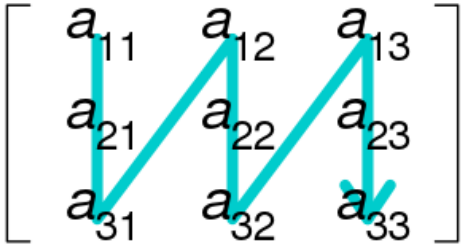
- "Spezzo" in sotto matrici
- alla "join" cerco il MAX dai max parziali

* Come "spezzo"?
* Quante "fette"?

Row-major order



Column-major order



Thread - core/CPU: sched_getcpu

```c
// LINUX ONLY
#define _GNU_SOURCE
#include <assert.h>
#include <sched.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>

void * thread_func (void * p){
    int cpu = sched_getcpu();
    pthread_t t_id = pthread_self();
    printf("Th id %d  - on CPU: %d\n",t_id, cpu);
    return NULL;
}


int  main(void){
    pthread_t t1,t2;
    long t_num=1;
    pthread_create(&t1, NULL, &thread_func,(void*)t_num);
    t_num=2;
    pthread_create(&t2, NULL, &thread_func,(void*)t_num);
    pthread_join(t1,NULL);
    pthread_join(t2, NULL);
    printf("DONE!\n");
}
```

Thread - Approfondimenti

- **I concetti affrontati sono da considerarsi base per quanto riguarda i thread**

- **Per approfondimenti si consiglia il seguente libro, al capitolo 4:**

  Mitchell, M., Oldham, J., and Samuel, A., Advanced Linux Programming. Boston, MA: New Riders, 2001.