

Principles of Concurrent Programming

Alessio Vecchio

alessio.vecchio@unipi.it

Pervasive Computing & Networking Lab. (PerLab)

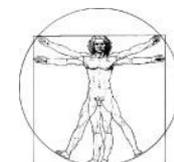
Dip. di Ingegneria dell'Informazione

Università di Pisa



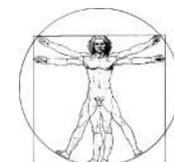
PerLab





- Concetti preliminari
- Interazione fra processi
- Modelli di cooperazione
- Specifica della concorrenza

- Introdurre i concetti basilari della programmazione concorrente
- Presentare le varie modalità di interazione fra i processi
- Introdurre i modelli secondo cui può avvenire la cooperazione
- Presentare alcuni costrutti per la specifica della concorrenza



- **Concetti preliminari**
- Interazione fra processi
- Modelli di cooperazione
- Specifica della concorrenza

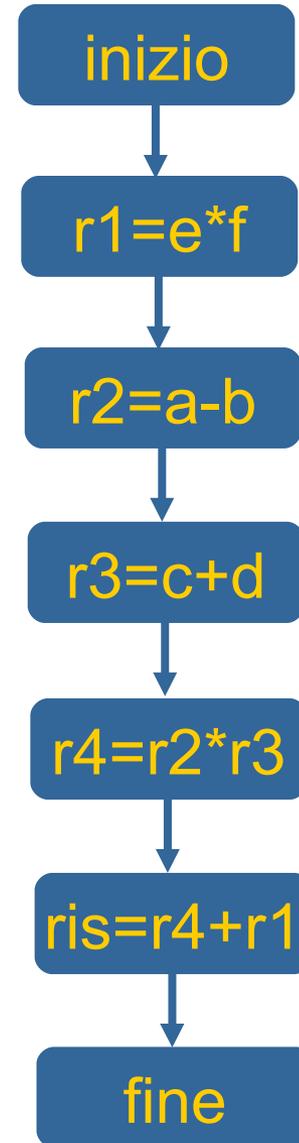
- Rappresenta graficamente l'evoluzione di un processo
- **Nodi** del grafo
 - Eventi generati dal processore - cambiamenti di stato prodotti dalle azioni del processore
- **Archi**
 - Specificano precedenze temporali fra gli eventi

■ Problema

- Calcolo di
 $(a-b)*(c+d)+(e*f)$

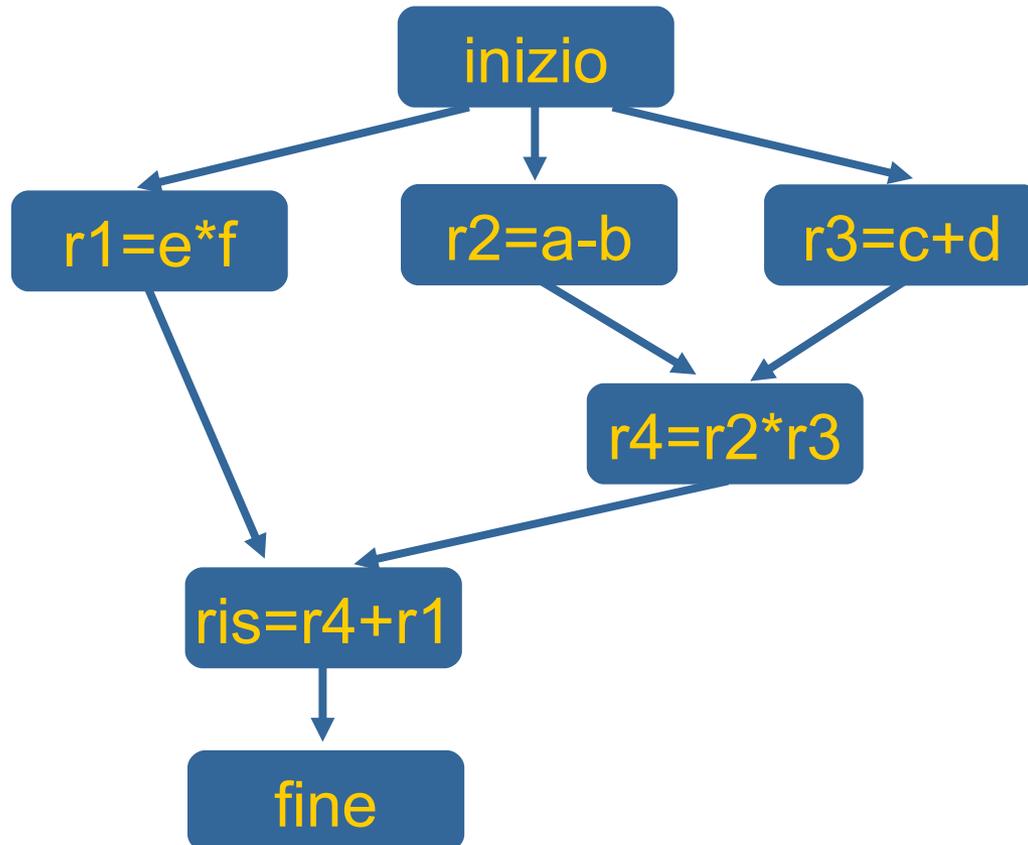
■ Algoritmo sequenziale

- $r1=(e*f)$
- $r2=(a-b)$
- $r3=(c+d)$
- $r4=r2*r3$
- $ris=r4+r1$



■ Problema

- Calcolo di $(a-b)*(c+d)+(e*f)$



■ Lettura, Elaborazione e Scrittura di N dati da/su file sequenziale

....

T buffer;

....

```
for(int i=0; i<N; i++) {
```

```
    leggi(buffer);
```

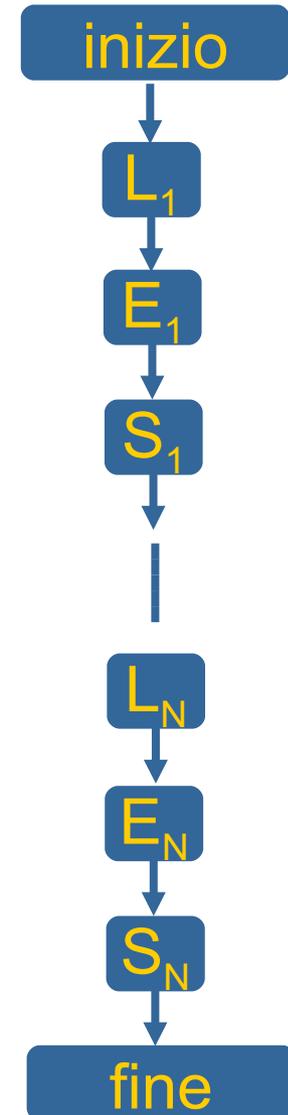
```
    elabora(buffer):
```

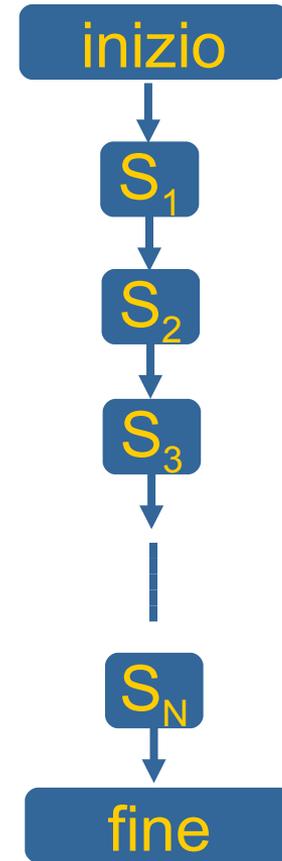
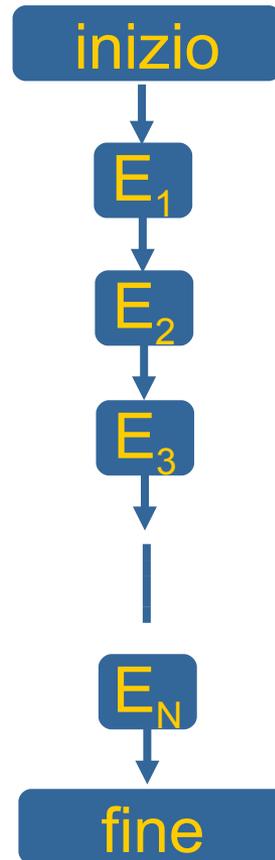
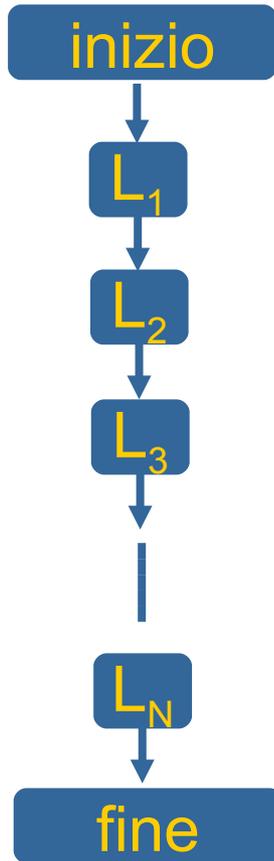
```
    scrivi(buffer);
```

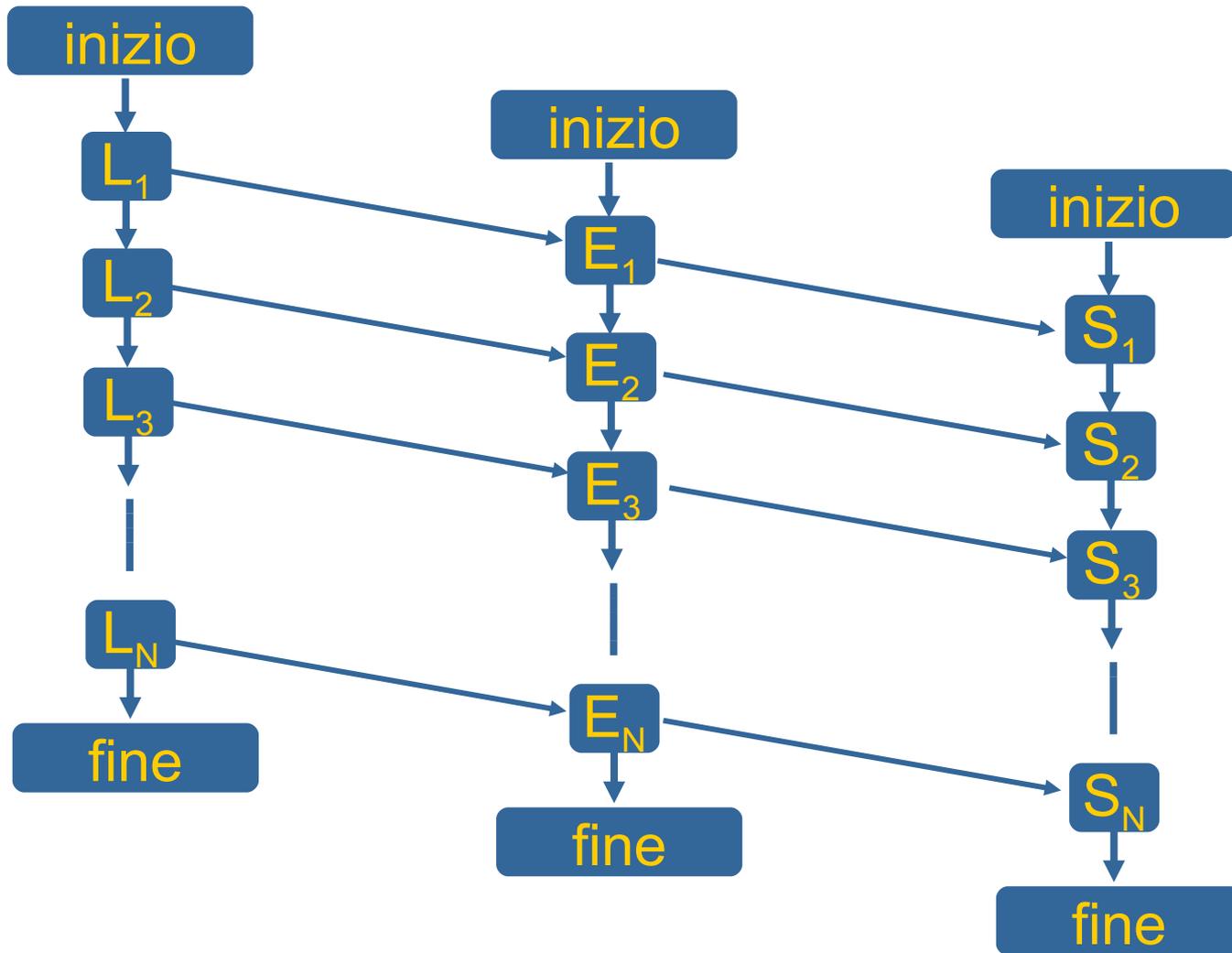
```
}
```

...

Grafo di precedenza a ordinamento totale





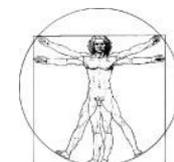


Grafo di precedenza a ordinamento **parziale**

- Gli archi nel grafo denotano vincoli di sincronizzazione
 - I processi per la lettura, elaborazione e scrittura NON sono indipendenti
 - Fra i processi avviene uno scambio di dati
 - In una esecuzione concorrente i processi, per interagire, devono sincronizzare le loro velocità

■ Elaborazione concorrente

- I processi sono **asincroni**
 - ▶ le velocità di esecuzione non sono uguali e non sono note a priori)
- I processi sono **interagenti**
 - ▶ Devono perciò sincronizzare le loro esecuzioni per poter interagire
- Necessità di un **linguaggio concorrente**
 - ▶ Per descrivere il programma come più sequenze da eseguire concorrentemente
- Necessità di **macchina concorrente**
 - ▶ Macchina in grado di eseguire più processi sequenziali contemporaneamente. Deve disporre di più processori (reali o virtuali)



- Concetti preliminari
- **Interazione fra processi**
- Modelli di cooperazione
- Specifica della concorrenza

- Cooperazione
- Competizione
- Interferenza

■ Interazione prevedibile e desiderata

- La loro presenza è necessaria perché insita nella logica del programma

■ Esempi:

- Un processo P non può eseguire un'azione B prima che il processo Q abbia eseguito A (scambio di un segnale di sincronizzazione)
- il processo P invia dati al processo Q perché li elabori (scambio di dati fra due processi)

■ Vincoli di sincronizzazione

■ Meccanismi di comunicazione fra processi

- Linguaggio
- Macchina concorrente

- Interazione prevedibile e necessaria, ma non desiderata
- Esempio:
 - Accesso contemporaneo di più processi a una risorsa (fisica o logica)
- Vincoli di sincronizzazione
 - Il vincolo di sincronizzazione non è più fisso (cooperazione) ma si possono 2 diverse forme
 - ▶ A aspetta che B abbia finito di utilizzare la risorsa per poterla utilizzare a sua volta
 - ▶ B aspetta che A abbia finito di utilizzare la risorsa per poterla utilizzare a sua volta

- Interazione **non prevista e non desiderata**
- Tipologie
 - Presenza di interazioni spurie, non richieste dalla natura del problema
 - Interazioni necessarie per la corretta soluzione del problema, ma programmate erroneamente
- Si manifestano come **errori dipendenti dal tempo**

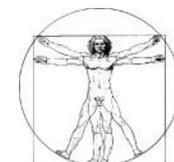
- Accesso non previsto di un processo P_h a una risorsa R privata di P_k
 - P_k accede a R senza precauzioni
 - P_h e P_k si possono trovare a modificare insieme la risorsa R

- Controllo degli accessi
 - Può eliminare le interferenze del primo tipo
 - Inefficace sulle interferenze del secondo tipo

- Macchina dotata di tanti processori (**virtuali**) quante sono le attività concorrenti



- Multi-programmazione
 - Realizza il concetto di processore virtuale
- Sincronizzazione/Comunicazione
 - Permette l'interazione fra processi
- Controllo degli accessi
 - Rileva e previene alcuni tipi di interferenza
- Meccanismi primitivi
 - Forniscono delle funzionalità atomiche (come fossero istruzioni della macchina fisica)



- Concetti preliminari
- Interazione fra processi
- **Modelli di cooperazione**
- Specifica della concorrenza

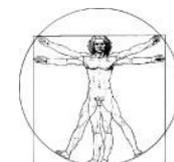
■ Modelli di Interazione

● Memoria Comune

- ▶ I processi condividono un'area di memoria attraverso la quale possono comunicare
- ▶ Un processo scrive informazioni nella memoria comune e gli altri leggono le stesse informazioni
- ▶ Il SO mette a disposizione i meccanismi necessari per la sincronizzazione

● Scambio di messaggi

- ▶ L'interazione avviene attraverso scambio di messaggi fra i processi
- ▶ Il meccanismo di comunicazione è supportato dal sistema operativo



- Concetti preliminari
- Interazione fra processi
- Modelli di cooperazione
- **Specifica della concorrenza**

- La programmazione concorrente necessita di costrutti linguistici per
 - Dichiarare
 - Creare
 - Attivare
 - Terminare
 - Sincronizzare
 - Far comunicare
- processi concorrenti

- Fork/Join
- Cobegin/Coend
- Process
- Libreria Pthread

Process P;

....

A

P=fork fun;

B

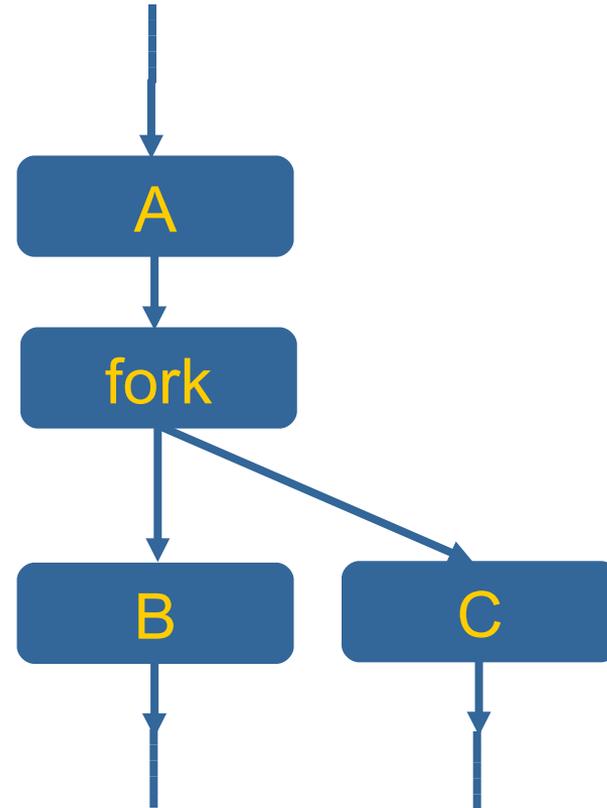
...

void fun() {

C

...

}



Process P;

....

A

P=fork fun;

B

...

join P;

D

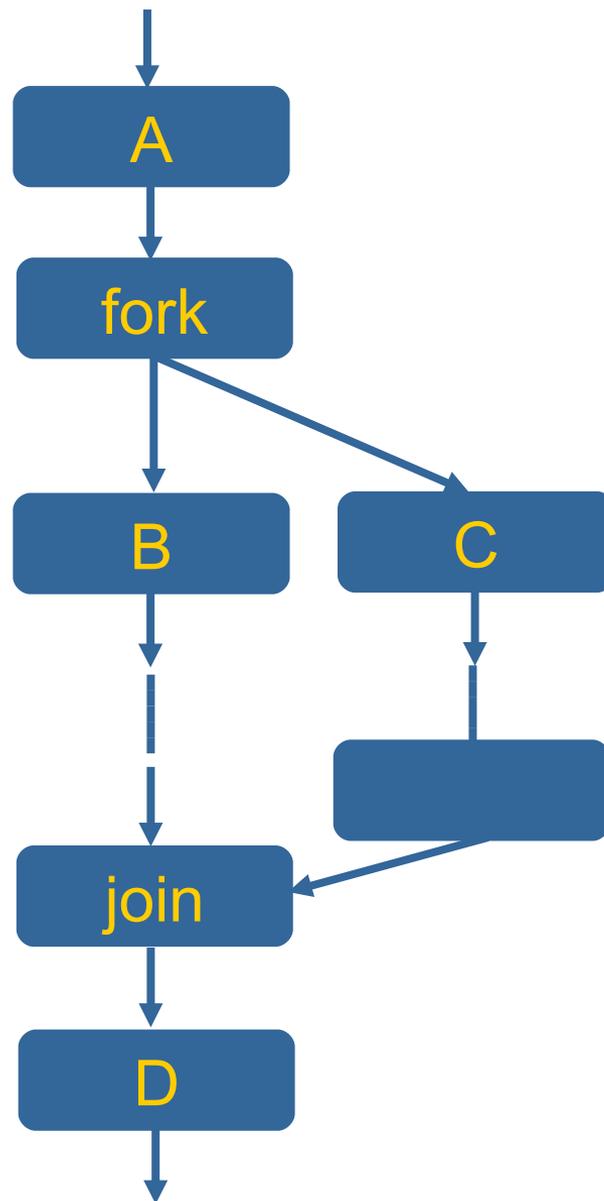
...

```
void fun() {
```

```
  C
```

```
  ...
```

```
}
```



```
#include <iostream>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
using namespace std;

int main(int argc, char* argv[]) {
    pid_t pid;
    pid=fork(); /* genera un nuovo processo */
    if(pid<0) { /* errore */
        cout << "Errore nella creazione del processo\n";
        exit(-1);
    } else if(pid==0) { /* processo figlio */
        execlp("/usr/bin/touch", "touch", "my_new_file", NULL);
    } else { /* processo genitore */
        int status;
        pid = wait(&status);
        cout << "Il processo figlio " << pid << " ha terminato\n";
        exit(0);
    }
}
```

- Proposto da Dijkstra, obbliga il programmatore a seguire uno schema di strutturazione

A;

Cobegin

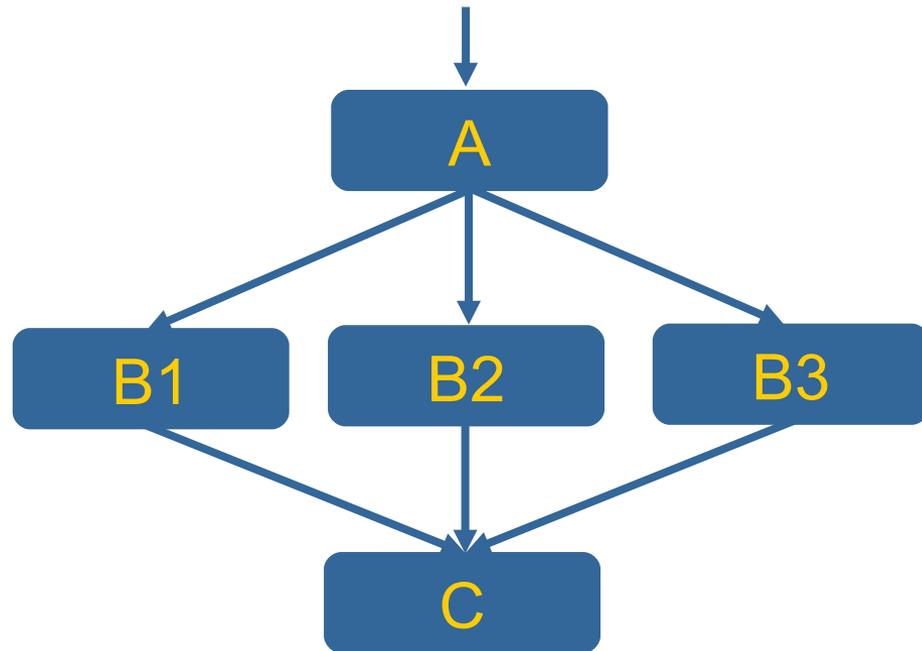
B1;

B2;

B3;

Coend

C;



- Dichiarazione simile a quella di una funzione
- Denota una parte di programma che verrà eseguita in concorrenza con le altre

```
Process <identificatore> (<par formali>) {  
    <dichirazioni var locali>;  
    <corpo del processo>;  
}
```

- Definita in ambito POSIX
 - Portable Operating System Interface
- Consente lo sviluppo di applicazioni multi-threaded in linguaggio C
- Offre primitive per
 - Creazione di thread (`pthread_create()`)
 - Attivazione dei thread
 - Sincronizzazione/Comunicazione dei thread
 - Terminazione dei thread (`pthread_exit()`)



Pthread Creation and Termination



```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS 5

void *PrintHello(void *threadid) {
    long tid;
    tid = (long) threadid;
    printf("Hello World! It's me, thread #%ld!\n", tid);
    pthread_exit(NULL);
}
```

```
int main(int argc, char *argv[]) {  
    pthread_t threads[NUM_THREADS];  
    int rc;  
    long t;  
    for(t=0; t<NUM_THREADS; t++) {  
        printf("In main: creating thread %ld\n", t);  
        rc = pthread_create(&threads[t], NULL, PrintHello, (void *) t);  
        if (rc) {  
            printf("ERROR; return code from pthread_create() is %d\n", rc);  
            exit(-1);  
        }  
    }  
    /* Last thing that main() should do */  
    pthread_exit(NULL);  
}
```



thread Creation and Termination (Con'd)



In main: creating thread 0

In main: creating thread 1

Hello World! It's me, thread #0!

In main: creating thread 2

Hello World! It's me, thread #1!

Hello World! It's me, thread #2!

In main: creating thread 3

In main: creating thread 4

Hello World! It's me, thread #3!

Hello World! It's me, thread #4!

Questions?

