Posix Threads: l'evoluzione dei processi UNIX

Raffaele Quitadamo,
PhD in Computer Science
Università di Modena e Reggio Emilia
quitadamo.raffaele@unimore.it

Sommario

- Pthreads vs Unix processes
- L'API per programmare coi pthreads
- Compilare coi pthreads
- Le primitive per gestire i thread
- Alcuni esempi di codice in Eclipse
- Conclusioni

Un esempio per partire: Apache web server

Approccio sequenziale

```
while (1) {
    accept_new_connection();
    read_command_from_connection();
    handle_connection(); /* leggi comando, ritorna risultato sulla connessione*/
    close_connection();
}
```

Possibile attacco: denial-of-service
(un client invia comandi lentamente
o solo a meta')

Approccio con processi UNIX

```
while (1) {
    accept_new_connection();
    read_command_from_connection();
    if (fork() == 0) {
        handle_connection();
        close_connection();
        exit(0);
    }
}
```

Rischio: far crashare il sistema per troppi processi
Performance: troppi processi rallentano il sistema
Processi isolati -- Problemi di performance nella comunicazione

Avremmo bisogno di un metodo per:

1. Generare processi piu' velocemente
2. Usare meno memoria
3. Condividere dati tra flussi di esecuzione

Cos'è un thread

- Definizione: "un flusso independente di istruzioni che viene schedulato ed eseguito come tale dal sistema operativo"
- Per capire che significa, ripensiamo al concetto di processo in UNIX.
- Un processo e' creato dal sistema operativo (con un certo overhead) e contiene informazioni sulle risorse del programma e lo stato di esecuzione:

- Process ID, user ID, and group ID
- Variabili d'ambiente
- Working directory
- Codice del programma
- Registri
- Stack
- Heap
- File descriptors
- Signal actions
- Shared libraries
- Inter-process communication tools (e.g. pipes or shared memory).

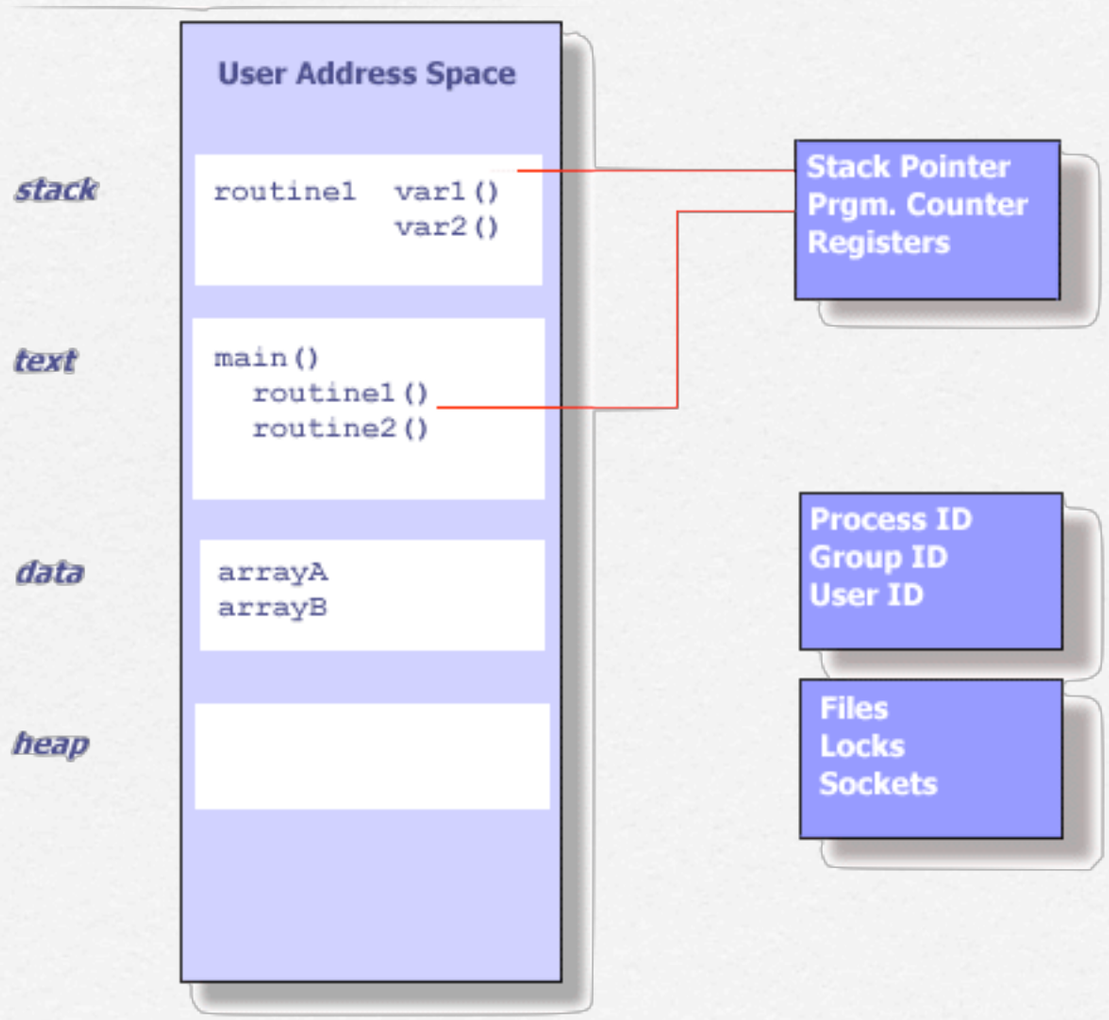
Il rapporto tra thread e processi

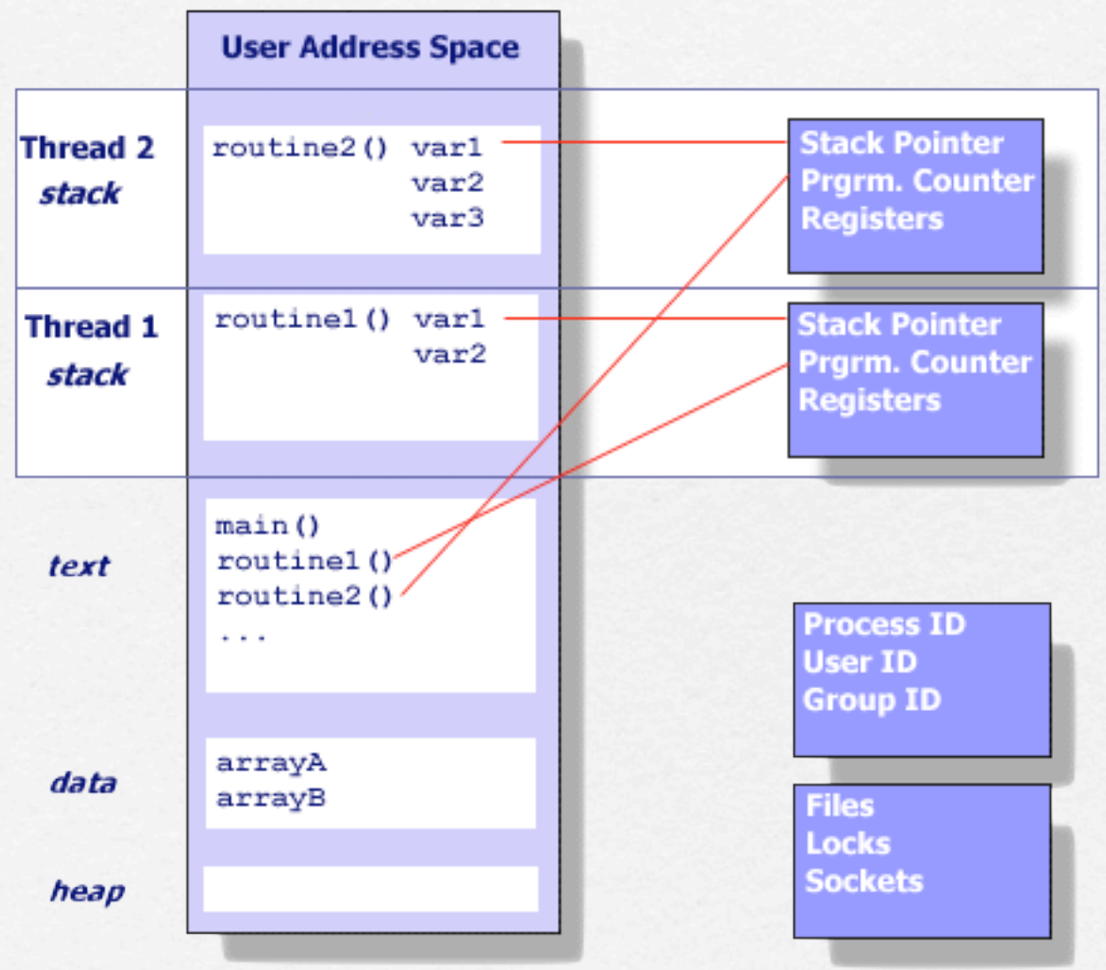
- Un thread mantiene un flusso di esecuzione autonomo, mantenendo il seguente stato:
  - Stack pointer
  - Registeri
  - Proprietà di Scheduling
  - Set di pending e blocked signals
  - Thread specific data

- In ambiente UNIX un thread:
  - Esiste all'interno di un processo e usa le sue risorse
  - Vive come flusso indipendente finché non muore il suo processo
  - Puó condividere risorse con altri thread dello stesso processo
  - É leggero rispetto ad un processo

- Vivendo nello stesso processo:
  - Cambiamenti (es. chiudere un file) fatti da un thread sono visibili ad un altro
  - I puntatori puntano alla stessa area di memoria (stesso spazio di indirizzamento)
  - Letture e scritture nella stessa memoria richiedono una apposita sincronizzazione.

Thread e processi in UNIX





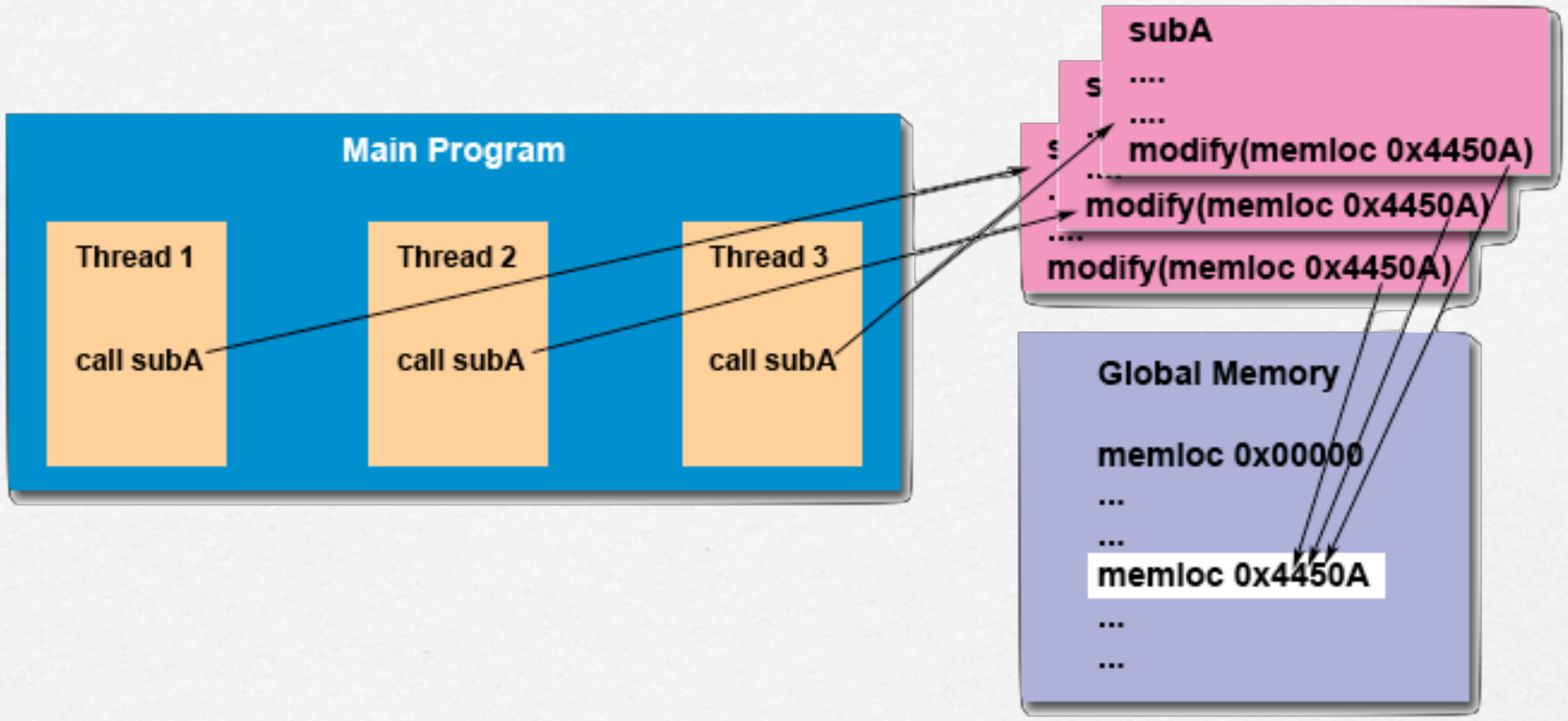
POSIX threads

- L'interfaccia di programmazione dei thread in UNIX e' stata standardizzata da IEEE nel 1995, con lo standard POSIX
- Tutte le implementazioni fedeli a questo standard si chiamano POSIX threads o pthreads.
- La libreria definisce un set di tipi C e primitive, esportate dall'header file pthread.h e implementate tramite la libreria libpthread

- Creare un thread richiede molte meno risorse che creare un processo.
- Confronto tra tempi per eseguire 50.000 fork() e 50.000 pthreads_create() (in secondi)

	fork()			pthread_create()		
Platform	**real**	**user**	**sys**	**real**	**user**	**sys**
IBM 375 MHz POWER3	61.94	3.49	53.74	7.46	2.76	6.79
IBM 1.5 GHz POWER4	44.08	2.21	40.27	1.49	0.97	0.97
IBM 1.9 GHz POWER5 p5-575	50.66	3.32	42.75	1.13	0.54	0.75
INTEL 2.4 GHz Xeon	23.81	3.12	8.97	1.70	0.53	0.30
INTEL 1.4 GHz Itanium 2	23.61	0.12	3.42	2.10	0.04	0.01

Thread safeness



Thread-safeness: "l'abilita' di eseguire diversi thread senza sporcare i dati condivisi"

Esempio: una applicazione crea 3 threads che chiamano delle funzioni di libreria

1. Queste funzioni accedono/modificano una struttura globale in memoria
2. Puo' accadere che i thread cerchino di modificare allo stesso tempo l'area condivisa
3. Una libreria thread-safe sincronizza l'accesso di ciascun thread alla struttura condivisa

Il set di funzioni POSIX

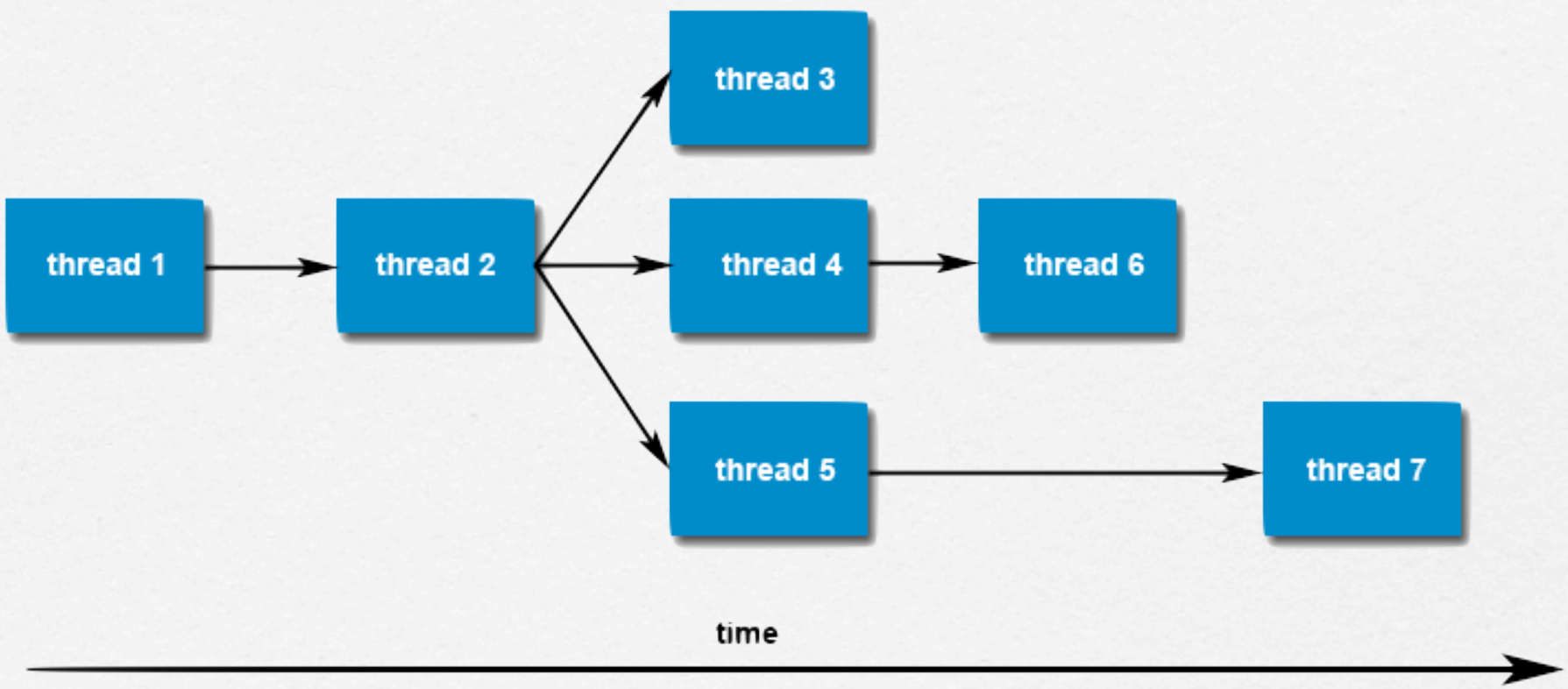
- Thread management: primitive per creare, distruggere, aspettare un pthread.
- Mutexes: Costrutto di "mutua esclusione" per garantire che un solo thread possa eseguire ad un certo istante in un blocco di codice
- Condition variables: Oggetti di sincronizzazione/comunicazione per aspettare o segnalare il verificarsi di condizioni.

Naming conventions: tutte le funzioni della libreria pthread iniziano per "pthread_"

Lo standard POSIX e' definito solo per il linguaggio C

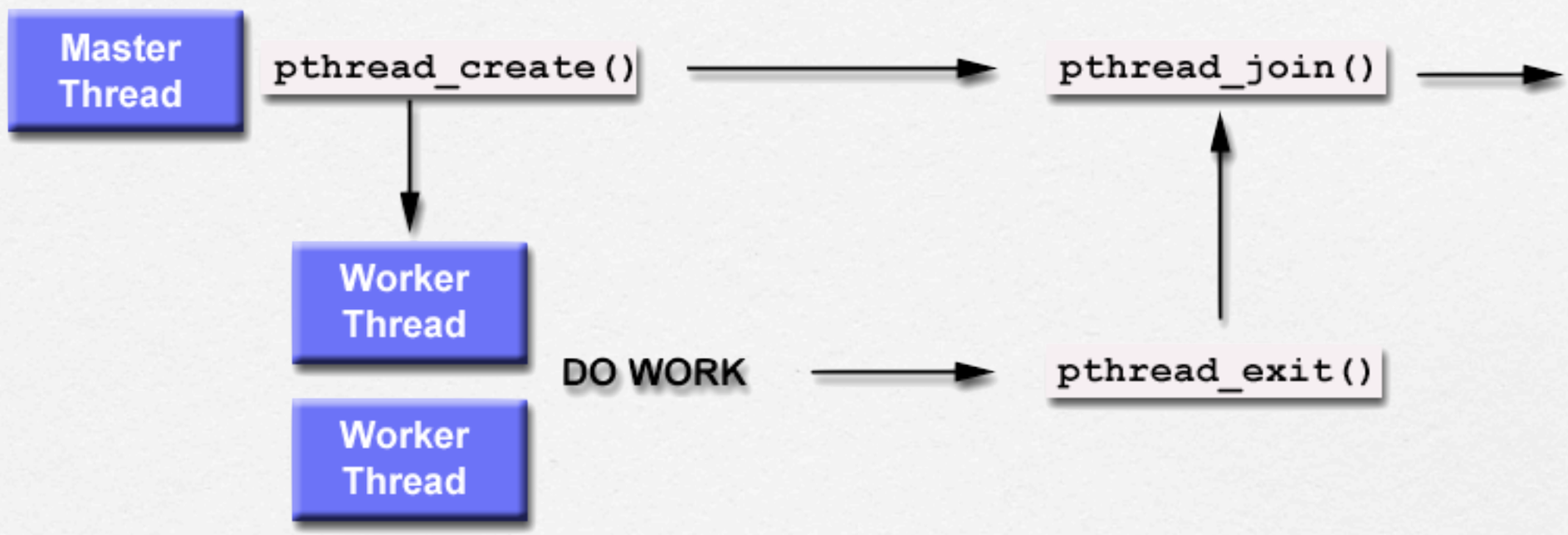
Esempio di compilazione: gcc -pthread main.c -o main

Creazione e distruzione di un pthread



Esempi hello.c, hello_arg2.c, hello_arg3.c

Aspettare un thread: pthread_join()



Quando si crea un pthread, un attributo consente di definire se esso sara:

- joinable (puo' essere aspettato con la join)
- detached (non puo' essere aspettato)

Esempio join1.c

Altre primitive

- pthread_self() -- equivalente di getpid()
- pthread_exit() -- termina il thread corrente (non il processo come exit()!)
- pthread_yield() -- cede lo scheduler ad un altro thread
- pthread_kill() -- per uccidere un pthread
- pthread_sigmask() -- per mascherare segnali ad un thread

Svantaggi dei pthreads

- La sincronizzazione deve essere fatta a mano dal programmatore
  - Errori possibili:
    - deadlock
    - starvation
    - dirty read
- Coi processi (es. pipe) e' invece il sistema operativo a occuparsi della sincronizzazione

Riferimenti per approfondire

- "Pthreads Programming". B. Nichols et al. O'Reilly and Associates.
- "Threads Primer". B. Lewis and D. Berg. Prentice Hall
- "Programming With POSIX Threads". D. Butenhof. Addison Wesley
  www.awl.com/cseng/titles/0-201-63392-2
- "Programming With Threads". S. Kleiman et al. Prentice Hall