# POSIX Thread

## I thread

Un **thread** è un **singolo flusso di istruzioni**, **all’interno di un processo**, che lo scheduler può fare eseguire separatamente e concorrentemente con il resto del processo. Per fare questo un thread deve possedere un proprio **contesto di esecuzione**.

In un **processo**:

* ci possono essere **più thread**;
* tutti i thread di uno stesso processo **condividono le risorse** del processo ed eseguono nello stesso spazio utente;
* ciascun thread può usare tutte le **variabili globali** del processo e condivide la tabella dei File Descriptor del processo;
* ciascun thread in più potrà avere anche dei **propri dati locali**.

Pro e contro nell’uso di thread:

**Pro**:

* + Visibilità dei dati globali, comporta una condivisione di oggetti semplificata.
  + Più flussi di esecuzione.
  + Gestione semplice di eventi asincroni.
  + Comunicazioni e context switch veloci.

**Contro**:

* + Concorrenza invece di parallelismo: occorre gestire la mutua esclusione per evitare che più thread utilizzino in maniera scoordinata i dati condivisi, modificandoli in momenti sbagliati.

La **concorrenza** deve essere **gestita** sia **da chi scrive il programma** che usa i thread, sia **dall’OS** che implementa le funzioni di libreria e le system calls utilizzate “contemporaneamente” da più thread di uno stesso processo.

## Operazioni atomiche

Un'**operazione**, che usa dati condivisi, è **atomica**, rispetto alle operazioni che usano gli stessi dati condivisi, se viene **eseguita in modo indivisibile**.

Può però capitare che una operazione indivisibile debba essere interrotta per permettere l'esecuzione di una operazione più urgente. In caso di interruzione, un modo per rendere comunque l'operazione atomica consiste nell'attendere la fine dell'interruzione e far ripartire dall'inizio l'operazione interrotta che si voleva fosse eseguita atomicamente, poiché in caso di interruzione riparto da capo, le condizioni iniziali da cui l'operazione riparte potrebbero essere nel frattempo cambiate.

## Pthread

I thread sono standardizzati. IEEE POSIX 1003.1c (1995) specifica l’interfaccia di programmazione (Application Program Interface – API) dei thread. I thread POSIX sono noti come **Pthread**.

Le API per Pthread distinguono le funzioni in tre gruppi:

* **Thread management**: funzioni per creare, eliminare, attendere la fine dei pthread.
* **Mutexes**: funzioni per supportare un tipo di sincronizzazione semplice chiamata “*mutex*” (abbreviazione di mutua esclusione). Comprende funzioni per creare ed eliminare la struttura per mutua esclusione di una risorsa, acquisire e rilasciare tale risorsa.
* **Condition variables**: funzioni a supporto di una sincronizzazione più complessa, dipendente dal valore di variabili, secondo i modi definiti dal programmatore. Comprende funzioni per creare ed eliminare la struttura per la sincronizzazione, per attendere e segnalare le modifiche delle variabili.

Convenzioni sui nomi delle funzioni:

* **pthread\_** 🡪 gestione dei pthread in generale
* **pthread\_attr\_** 🡪 gestione proprietà dei thread
* **pthread\_mutex\_** 🡪 gestione mutua esclusione
* **pthread\_mutexattr\_** 🡪 proprietà delle strutture per mutua esclusione
* **pthread\_cond\_** 🡪 gestione delle variabili di condizione
* **pthread\_condattr\_** 🡪 proprietà delle variabili di condizione
* **pthread\_key\_** 🡪 dati speciali dei thread

Il file **pthread.h** contiene le definizioni dei pthread.

Durante la fase di linking, usando il gcc, aggiungere il flag **-lpthread** per usare la libreria **libpthread.so**.

## API per la creazione ed esecuzione di Pthread

### pthread\_create

int **pthread\_create** ( pthread\_t \* **threadID**,

pthread\_attr\_t \***attr**,

void\* (\***start\_routine**)(void \*),

void \* **arg** );

Crea un thread e lo rende eseguibile, cioè lo mette a disposizione dello scheduler che prima o poi lo farà ripartire. I parametri:

* **threadID** 🡪 è un puntatore ad un identificatore di thread in cui verrà scritto l’identificatore del thread creato.
* **attr** 🡪 seleziona caratteristiche particolari, può essere posto a NULL per ottenere il comportamento di default.
* **start\_routine** 🡪 è il nome (indirizzo) della procedura da fare eseguire dal thread.
* **arg** 🡪 è un puntatore che viene passato come argomento a start\_routine. tale puntatore solitamente punta ad un’area di memoria allocata dinamicamente.

### pthread\_exit

void **pthread\_exit** (void \*retval);

Termina l’esecuzione del thread da cui viene chiamata, immagazzina l’indirizzo retval, restituendolo ad un altro thread che attende la sua fine. Il sistema libera le risorse allocate al thread.

### pthread\_self

pthread\_t **pthread\_self** (void);

Restituisce l’identificatore del thread che la chiama.

### pthread\_equal

int **pthread\_equal** (pthread\_t pthread1, pthread\_t pthread2);

Restituisce 1 se i due identificatori pthread sono uguali.

## Mutex variables

**Mutex** è l’abbreviazione di *mutua esclusione*. Una **variabile mutex** (più formalmente, di tipo *pthread\_mutex\_t*) è una variabile che serve per regolare l’accesso a dei dati che debbono essere protetti dall’accesso contemporaneo da parte di più thread. Ogni thread, prima di accedere a tali dati, deve effettuare una operazione di lock su una stessa variabile mutex. L’operazione detta “lock” di una variabile mutex blocca l’accesso da parte di altri thread. Finito l’accesso, il thread effettua un’operazione detta di “unlock” che libera la variabile mutex.

int **pthread\_mutex\_trylock** (pthread\_mutex\_t \*mutex);

Questa funzione è come la lock(), ma se si accorge che la mutex è già in possesso di un altro thread, restituisce immediatamente il risultato EBUSY. Se invece si ottiene la proprietà della mutex, restituisce 0.

## Condition variables

Le mutex consentono di effettuare sincronizzazioni tra pthreads nel momento in cui ciascun pthread tenta l’accesso in mutua esclusione. L’accesso viene consentito senza poter valutare a priori il valore delle variabili condivise.

Al contrario, le **condition variables** sono delle variabili che consentono di effettuare delle sincronizzazioni tra pthreads mentre questi già detengono la mutua esclusione, bloccandoli o facendoli continuare in base a condizioni definite dal programmatore e dipendenti dal valore attuale delle variabili condivise.

Ci sono tre principali funzioni per sincronizzare con le condition variables:

* **pthread\_cond\_wait** 🡪 blocca un thread fino a che un altro thread lo risveglia e può prendere la mutua esclusione.
* **pthread\_cond\_signal** 🡪 sveglia un altro thread bloccato sulla pthread\_cond\_wait, ma l’altro thread, prima di ripartire, deve ottenere la mutua esclusione.
* **pthread\_cond\_broadcast** 🡪 sveglia tutti i threads bloccati sulla pthread\_cond\_wait, ma ciascuno di questi, prima di ripartire uno alla volta, deve ottenere la mutua esclusione.