THREAD POSIX

TPSIT

Libreria <pthread.h>

- Lo standard ANSI C non prevede l'utilizzo dei Thread.
- Lo standard POSIX definisce una nuova libreria contenente delle nuove funzioni per la gestione dei thread.

Che cos’è un thread?

- Un thread non è altro che una funzione C che viene eseguita in maniera concorrente ad altre funzioni nell'ambito di uno stesso processo.
- Tutti i thread che fanno parte di uno stesso processo ne condividono lo spazio di indirizzamento.
- Le variabili che vengono dichiarate localmente all'interno della funzione che implementa il codice di ciascun thread sono private per quello specifico thread, e pertanto non accessibili da parte degli altri thread del processo.
- La memoria condivisa sarà pertanto rappresentata dalle variabili globali del processo.
- Gli identificativi dei thread sono gestiti mediante il tipo pthread_t, che è la ridefinizione di un Unsigned long int

Abilitazione dei thread

- **int pthread_setconcurrency (int nThread);**

- Comunica al sistema il numero di thread che si intende creare.
- Restituisce 0 in caso di successo, qualsiasi cosa diversa da 0 in caso di insuccesso.
- nThread rappresenta il numero di thread da creare

Creazione di un nuovo thread

- **int pthread_create (pthread_t * tID, const pthread_attr *attr, void *(*foo)(void*), void *arg);**

- Restituisce 0 in caso di successo, qualsiasi cosa diversa da 0 in caso di insuccesso.

- **tID** è l'indirizzo dell'oggetto pthread_t destinato a contenere l'id che il sistema assegna al thread
- **attr** sono gli attributi che l'utente vuole assegnare al thread. (se vale NULL vengono usati quelli di default)
- **foo** è il nome della funzione C che verrà eseguita all'interno del nuovo thread.
- **arg** è il puntatore ai parametri da passare alla funzione foo. Deve assolutamente essere un void *

Attendere un thread

- **int pthread_join (pthread_t tID, void ** risPt);**

- Sospende il thread principale (main) in attesa che il thread identificato da tID giunga al termine
- risPt rappresenta l'indirizzo del puntatore al valore restituito dalla funzione foo eseguita nel thread
- Ritorna 0 in caso di successo, diverso da 0 altrimenti.
- Non è prevista invece, nello standard POSIX, una funzione di attesa di un thread generico.

Funzioni utili

- **int pthread_kill (pthread_t tid, int signo);**
- Invia un segnale di terminazione al thread specificato dal 1° par Ritorna 0 in caso di successo, diverso da 0 altrimenti.
- **signo** è l'identificatore del segnale che si vuole inviare al thread (normalmente SIGKILL definito in "signal.h").

- **int pthread_self ( );**
- restituisce il tID del thread corrente.

Istruzioni per la compilazione

- Per compilare un programma multithread occorre specificare il flag **–l pthread** sulla linea di comando

- Es:
- gcc main.c –o main –lpthread

SEMAFORI DI MUTUA ESLUSIONE: MUTEX

- Un mutex è una variabile che serve per la protezione delle sezioni critiche:
  - Variabili condivise e modificabili da più thread
  - Solo un thread alla volta può accedere ad una risorsa protetta da un mutex
- Il mutex è un semaforo **binario**, cioè un valore può essere 0 (occupato) oppure 1 (libero)

MUTEX

- I mutex possono essere paragonati a classiche serrature:
  - Il primo thread che ha accesso alla coda dei lavori lascia fuori gli altri thread fino a quando il suo compito non è stato portato a termine
- I threads, dunque, «posizionano» un mutex nelle sezioni di codice nelle quali vengono condivisi i dati

MUTEX: Logica d'uso

- Creare ed inizializzare una variabile mutex
- Più thread tentano di accedere alla risorsa invocando l'operazione di lock
- Un solo thread riesce ad acquisire il mutex mentre gli altri si bloccano
- Il thread che ha acquisito il mutex manipola la risorsa
- Lo stesso thread la rilascia invocando la unlock
- Un altro thread acquisisce il mutex e così via
- Distruzione della variabile mutex

Creazione Mutex

- Per creare un mutex è necessario usare una variabile di tipo **pthread_mutex_t** contenuta nella libreria pthread
- pthread_mutex_t è una struttura che contiene:
  - Nome del mutex
  - Proprietario
  - Contatore
  - Struttura associata al mutex
  - La coda dei processi sospesi in attessa che mutex sia libero
  - .....
- Per il tipo di dato pthread_mutex_t, è definita la macro di inizializzazione PTHREAD_MUTEX_INITIALIZER
- l mutex è un tipo definito "ad hoc" per gestire la mutua esclusione quindi il valore iniziale può essergli assegnato anche in modo statico mediante questa macro
- pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;

Mutex: Lock

int pthread_mutex_lock(pthread_mutex_t *mutex)

- Su mutex sono possibili solo due operazioni: locking e unlocking (equivalenti a wait e signal sui semafori)
- Ogni thread, prima di accedere ai dati condivisi, deve effettuare la lock su una stessa variabile mutex
- Blocca l'accesso da parte di altri thread
- Se più thread eseguono l'operazione di lock su una stessa variabile mutex, solo uno dei thread termina la lock e prosegue l'esecuzione, gli altri rimangono bloccati nella lock. In tal modo, il processo che continua l'esecuzione può accedere ai dati (protetti mediante la mutex).

Mutex: trylock

int pthread_mutex_trylock(pthread_mutex_t *mutex)

- Valore di ritorno:
  - 0 in caso di successo e si ottenga la proprietà della mutex
  - EBUSY se il mutex è occupato

Mutex: Unlock

int pthread_mutex_unlock(pthread_mutex_t *mutex)

- Un altro thread che ha precedentemente eseguito la lock della mutex potrà allora terminare la lock ed accedere a sua volta ai dati.
- Valore di ritorno: 0 in caso di successo

Mutex: destroy

int pthread_mutex_destroy(pthread_mutex_t *mutex)

- Elimina il mutex
- Valore di ritorno:
  - 0 in caso di successo
  - EBUSY se il mutex è occupato