THREAD

Un thread è un flusso di esecuzione all'interno di un processo. Creare un thread significa creare una **CPU virtuale** e allocare uno **stack** per il nuovo flusso di esecuzione.
Da un certo punto di vista c'è una somiglianza tra creare un thread e creare un processo;
la grossa differenza tra un thread e un processo è il fatto che più thread vedono lo stesso spazio di indirizzamento, quindi la stessa virtualizzazione della memoria, mentre un processo è isolato da altri processi.
Quando uno scheduler passa da un thread di un processo ad un altro thread dello stesso processo, non deve cambiare lo spazio di indirizzamento.
A livello di kernel un sistema operativo ha una struttura dati per i processi chiamata **PCB**, analogamente ogni thread ha una struttura dati che rappresenta i suoi dati ed è chiamata **Thread Control Block.**
Dal punto di vista logico ogni thread ha:

- **Stack:** Regione di memoria dove vengono allocate le variabili locali, metadati, dati ecc. Tutti questi stack vivono nello stesso spazio di indirizzamento.

- **Variabile errno:** essenziale per permettere di fare ai thread system call. Se una system call fallisce il motivo per cui è fallita viene inserita all'interno di essa. Queste variabili sono uniche sennò ogni volta si sovrascriverebbero.

Thread Local Storage = spazio locale a ogni thread, accessibile tramite interfaccia di tipo dizionario chiave/valore

Per creare un Thread usiamo la funzione **posix phtread** che ha come parametri :

1. Identificatore: pthread_t *thread
2. Parametro per gli attributi: praticamente mai usato
3. Start_routine: prende una funzione void e mi restituisce una funzione void, sarebbe il codice per i thread
4. Argometno per start_routine

Se voglio uscire da un thread devo usare **pthread_exit** , con exit esco da tutti i thread di un processo. Se voglio aspettare un thread uso **phtread_join.**

Sezione critica = Pezzo di codice che va ad accedere ad una risorsa condivisa.
Si può verificare il caso di **race condition** ovvero la situazione in cui più thread accedono ad una stessa risorsa contemporaneamente. Per risolvere questa situazione dobbiam usare la sincronizzazione per obbligare ai thread a non entrare insieme nella sezione critica. Il metodo più comune è quello di usare dei lock. Un **lock** è un concetto generale di "blocco" per la sincronizzazione. Un **mutex** è un tipo di lock. Un esempio di lock è mutex. **Mutex** è un oggetto che serve a bloccare una risorsa per farla usare solo a un thread per volta. Quindi mutex è un tipo di lock che garantisce la **mutua esclusione**, ovvero se un thread sta accedendo ad una risorsa nessun altro thread può accedere alla risorsa in quel momento.
Quanti lock per programma? Non abbiamo una regola fissa ma è intelligente usare lock diversi per strutture dati diverse.
Quando implementiamo i lock ci dobbiamo preoccupare di:

- Se la nostra implementazione offre mutua esclusione
- Fairness e starvation
- Performance

La fairness si riferisce al fatto che tutti i thread abbiano una possibilità equa di acquisire il lock.
La starvation si verifica quando uno o più thread non riescono mai ad accedere alla risorsa condivisa.

Senza un supporto hardware non possiamo implementare. Diversi processori forniscono delle primitive atomiche quali:

- **Test-and-set:** singola operazione che testa il valore di un'allocazione di memoria e se trova 0 lo mette a 1. 0 significa libero, 1 occupato.
- **Scambi atomici:** inverte il valore di 2 allocazioni di memoria.

Spin lock: è un tipo di lock usato nei programmi per impedire che due thread accedano contemporaneamente alla stessa risorsa. Se un thread trova il lock occupato, non si mette in pausa, ma rimane in attesa attiva ovvero "gira a vuoto" finchè il lock non si libera.

Inversione delle priorità: supponiamo di avere uno scheduler a priorità a due thread:

- T1 = bassa priorità, T2 = alta priorità
- Se T2 è in attesa di qualcosa, allora va in esecuzione T1
- T1 acquisisce un certo lock che chiamiam 1
- T2 torna nello stato ready
- Lo scheduler deschedula T1 e manda in esecuzione T2
- T2 va in spin-wait per il lock1
- T1 è l'unico che può rilasciare il lock1, non viene messo in esecuzione, perché T2 ha la priorità.

Ci sono modi per risolvere il problema, per esempio "ereditare la priorità" di un thread in attesa (se maggiore di quella corrente)

Funzione rientrante(single-thread): è una funzione che si comporta correttamente anche se interrotta a metà di un'esecuzione per essere nuovamente chiamata.

Funzione thread-safe: funzione che si comporta correttamente anche se eseguita da più thread contemporaneamente.

Bug tipici dovuti alla concorrenza, ossia codici che in una situazione single-threaded non danno problemi ma in situazioni multi-thread protebbero.

Deadlock: è una situazione in cui due o più processi si bloccano a vicenda perchè ciascuno sta aspettando una risorsa che è già bloccata da un altro processo del gruppo, e nessuno può andare avanti.