**Che cos’è un thread?**

I Threads sono l’unità minima d’uso della CPU e comprende un identificatore di thread un program counter un insieme di registri e un stack.

**Aree dati che vengono condivise dai thread?**

Un thread condivide con gli altri thread che appartengono allo stesso processo la sezione del codice la sezione dei dati e altre risorse le quali possono essere file aperti, segnali ecc.

Un processo può essere composto da un solo thread (processo tradizionale) in questo caso i trhead vengono eseguiti uno alla volta in time sharing oppure può essere composto da più tread che lavorano a più compiti in modo concorrente.

**Quali sono i suoi vantaggi dei thread?**

I vantaggi di avere una programmazione multithread sono:

* Un programma continua l’esecuzione anche se una parte di esso è bloccato
* I thread condividono a priori la memoria e le risorse del processo alla quale appartengono e il vantaggio della condivisione del codice consiste nel fatto che un’applicazione può avere molte thread per attività diverse tutti nello stesso spazio indirizzi.
* La creazione, gestione e cancellazione dei thread richiedono meno tempo in confronto alle rispettive operazioni effettuate sui processi, queste economia di tempo è dovuta al fatto che non cambio lo spazio indirizzi.
* I vantaggi della programmazione multithread aumentano notevolmente nelle architetture multiprocessore dove i thread si possono eseguire in parallelo.

**Differenza tra thread implementati a livello utente e thread implementati a livello kernel?**

**Thread a livello utente:** il sistema operativo non sa che esistono i thread, utilizza librerie realizzate a livello utente che forniscono tutto il supporto per la gestione dei thread: creazione, terminazione, sincronizzazione nell’accesso di variabili globali del processo, per lo scheduling, etc.; tutte queste funzioni sono realizzate nello spazio utente.

**Vantaggi:** Quando creo un thread o passo da un thread ad un altro non ho bisogno dell’intervento del sistema operativo quindi tutte queste operazioni avvengo velocemente (le chiamate al sistema sono lente).

**Svantaggi**

I thread possono chiamare le system call, ad esempio per operazione di I/O; in questo caso interviene il SO che blocca il processo e di conseguenza tutti i thread in esso contenuti.

Non è possibile, con i thread a livello utente, sfruttare il parallelismo delle architetture multiprocessore, dato che quando un processo è assegnato ad uno dei processori, tutti i suoi thread sono eseguiti, uno alla volta, su quel solo processore.

**Thread a livello Kernel:** il sistema operativo è al corrente della presenza dei thread, in questo caso a ciascuna funzione di gestione dei thread, comprese la creazione, la terminazione, la sincronizzazione e lo scheduling corrisponde a una chiamata di sistema.

A differenza dei thread a livello utente, ora se un thread esegue una chiamata di sistema, il thread si blocca ma gli altri thread possono continuare la loro esecuzione e ho la possibilità di mandarli su processori multpli.

**Thread in Linux**

Linux non ha i thread, ma esiste un trucco che permette di realizzare una cosa simile ai thread.

La fork in linux non è la chiamata al sistema che crea un nuovo processo, in linux quando chiamiamo la fork a sua volta la fork chiama quella che è effettivamente la chiamata al sistema, quindi la fork non fa tutta da sola perché non è un chiamata al sistema ma la chiamata al sistema in linux si chiama clone che è tipica di linux.

La clone crea un nuovo processo a differenza della fork però possiamo indicare separatamente cosa duplicare e che cosa no, in questo modo possiamo avare dei processi che condividono un parte dell’area di memoria con quello di prima così facendo abbiamo un effetto simile a quello dei thread, ma non è un thread è un nuovo processo con un PID diverso.