Thread.
Processo —> fa delle richieste: messaggi, accessi a memoria —> per effettuare le richieste si deve bloccare in attesa del risultato.
Per evitare che questo accada e si perda tempo —> Esecuzione di più processi “contemporaneamente”.
Quando si blocca uno, eseguo gli altri = CPU più efficiente.
Eseguire + processi : richiede + info, + risorse , per ogni processo e attenzione a non far interferire i processi.

L’idea era di creare un Grande Processo, con all’interno piccoli processi, ognuno dei quali si occupa di un compito ben specifico. Per rendere questi eventi indipendenti e alle volte concorrenti.

DOS = sistema operativo ai tempi, era monoprocesso e mono thread 1:1
APPLE = sviluppa l’idea di eventi indipendenti concorrenti = più processi, ma un solo thread per processo M:1
JAVA = primo tipo di ambiente che permetteva più thread in un processo
WINDOWS = all’inizio DOS camuffato con tante diverse, più tardi ambiente multiprocesso e multithread

Thread hanno due caratteristiche principali:
Resource ownership = dove risiede il processo, risorse aperte, zone di memoria che lo riguardano
Scheduling/ execution = lo stato del processo dal punto di vista della CPU
Queste due caratt. vengono trattate in maniere diverse da sistemi operativi diversi

Thread sono dei processi leggeri che vivono nello spazio del processo:
Il processo con il suo stato, spazio virtuale e risorse esiste ancora, ha un suo stack pointer con l’istruzione da eseguire.
Il thread può accedere alle risorse dei processi, ma di se stesso sa solo uno stack pointer = puntatore all’istruzione da eseguire. Più leggero, minor informazioni di contesto possibile, per rendere i switch tra i thread più veloci possibile.

Ogni thread ha:
Uno stato di esecuzione, Un contesto (molto leggero) che va salvato, Una stack di esecuzione, l’accesso alla memoria e alle risorse dei suoi

processi. I thread di un processo condividono questo accesso che va gestito.
Possiamo vedere il thread come un program counter indipendente all'interno di un processo.
Sebbene ci siano più thread c'è una CPU e quindi sarà un thread alla volta in esecuzione.

Benefici:
Meno tempo ad avviare nuovo thread, rispetto a nuovo processo
Switch tra thread immensamente più veloce rispetto a switch tra processo, oltre al fatto che tra processi avvengono chiamate al kernel (+ pesante)
Velocità di esecuzione perché più thread occupano meglio spazio della CPU rispetto a un singolo processo

Ciclo di Vita di Thread:
nascono come processo leggero all'interno di un processo padre, se il processo padre viene sospeso anche i thread son sospesi, se il processo padre termina anche i thread terminano.
Questo accade perché il sistema operativo considera il processo , non il thread.
Anche se oggi esistono anche CPU multi-core che supportano multi-thread = parallelo fisico

Attività Principlai dei Thread:
Salvataggio di Stato di esecuzione
Sincronizzazione

dove gli Stati possono essere:

- Spawn = lancio di un thread
- Block = attesa di evento
- Unblock = termina l'attesa ritorno in esecuzione
- Finish = termina il thread

<u>Esempio.</u>
Remote Procedure Call = primo tipo di interazione complessa avvenuta su internet. L'invio di una richiesta al server, avviene una procedura remota (fatta da un altro utente), ritorno di un risultato.
Un programma vuole effettuare due RPC a due hosts diversi e ottenere i due risultati e combinarli.

*Single-Thread:

1. Lancio RPC Request
2. Elaborazione Server
3. Torna Risultato
4. Lancio 2 RPC Request
5. Elaborazione Server
6. Risultato
7. Risultato Finale

**Multi-thread:
Un Thread gestisce una Richiesta, nello stato di thread bloccato, il processo fa un switch e passa al thread 2 che lancia la 2 richiesta. Entrambe i thread son bloccati, ma le richieste avvengono contemporaneamente.
Tornano i due thread, processano i risultati e combino il risultato finale.

= Utilizzo dello slice temporale della CPU da parte del Processo

Implementazione dei Thread

Livello Utente: Un processo a livello Kernel, più thread per il processo a livello Utente che vanno gestiti dall'applicazione. Il Kernel non è a conoscenza dei thread. L'applicazione gestisce lo scheduling dei thread.
Vantaggi:

1. Scheduling gestito dall'applicazione che conosce cosa fanno i thread e quanto tempo hanno bisogno.
2. Switch tra thread non richiede uno switch del kernel
3. Può essere eseguito su qualsiasi sistema operativo, perché l'OS vede solo il processo

Svantaggi:

1. Una chiamata a sistema blocca un thread, ma si blocca anche tutto il processo per via della chiamata al kernel in attesa della richiesta.
2. Non sfrutta abbastanza la potenza del multi-core del kernel

Livello Kernel: Il kernel gestisce il processo e i thread. Lo scheduling dei thread vien fatto da parte del Kernel decidendo quali thread su quali core vanno etc.
Vantaggi:

1. Posso schedare differenti thread dello stesso processo su diversi processori.
2. Posso eseguire dei thread di un processo anche se un altro si è bloccato in attesa di richiesta
3. Il kernel stesso può essere multithread per evitare di fare switch di

tutto il kernel ma solo delle system call attivate da un utente.

Svantaggi:

1. Il kernel non conosce il livello applicativo e quindi non schedula in maniera efficiente i thread.
2. Switch tra un thread all'altro fa una chiamata al kernel, rende complesso il switch

Approcci Combinati: best of both worlds
Crea thread in user space..**

Symmetric MultiProcessors (SMP)

Un tempo il PC visto come macchina sequenziale: si eseguono le istruzioni sequenzialmente
Recentemente (2000) esplosione del trend di piattaforme multiprocessore o cluster di PC.
Tassonomia di Flynn:categorizza le macchine parallele

- SISD (Single Instruction Single Data)
- SIMD(Single Instruction Multiple Data
- MISD
- MIND (le + importanti)

In sistemi paralleli vanno gestiti gli accessi a zone condivise

Architetture MultiProcessori:

- Thread e processi simultanei e concorrenti
- Scheduling
- Vanno sincronizzati gli accessi a zone condivise
- Memory management diventa più complesso
- Reliability and Tolleranza ai Guasti = data dal sistema distribuito, ovvero muore un pc ma gli altri resistono. e' una capacità intrinseca ai sistemi distribuiti ma vanno implementati.

MicroKernel

Non è più una stratificazione di risorse che servono al Kernel
Ma una serie di applicazioni che si sviluppano verticalmente e il microkernel funziona da bus per inviare messaggi tra applicazioni utente e applicazioni di sistema.

Posix Thread (PThreads)