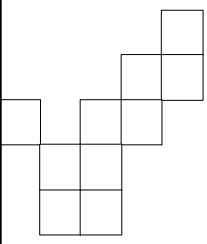


"Thread"

Corso di "Sistemi per Elaborazione dell'Informazione"
Prof. Carpentieri Bruno
A.A. 2004/2005

Aliberti Filly
Pacileo Roberta

Overview

Introduzione

- Nel modello a processi, l'attivazione di un processo, il cambio di contesto sono operazioni molto complesse che richiedono ingenti quantità di tempo per essere portate a termine
- Tuttavia a volte l'attività richiesta ha vita relativamente breve rispetto a questi tempi
- Es. Invio di una pagina html da parte di un server Web
- "troppo leggera per motivare un nuovo processo"

Il modello a thread

- Le idee di base dietro il modello a thread sono:
  - Permettere la definizione di attività 'leggerè (**ligthweight processes--LWP**) con costo di attivazione e terminazione limitato
  - Possibilità di condividere lo stesso spazio di indirizzamento
- Ogni processo racchiude più flussi di controllo (thread) che condividono le aree testo e dati

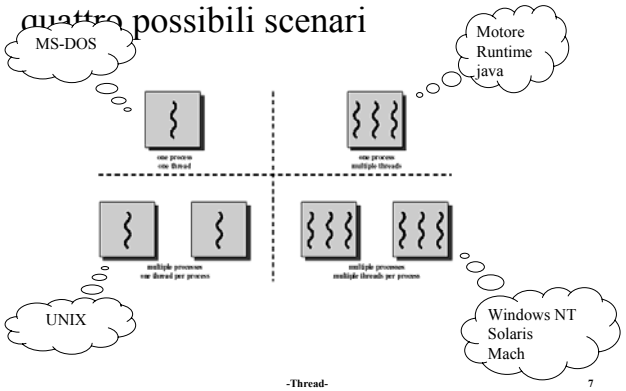
Thread

- Un thread è l'unità di base d'uso della CPU e comprende:
  - Identificatore di thread
  - Program counter
  - Insieme di registri
  - Stack
- Condivide con gli altri thread che appartengono allo stesso processo
  - Sezione codice
  - Sezione dati
  - Risorse di sistema (es.: file aperti)

Multithread

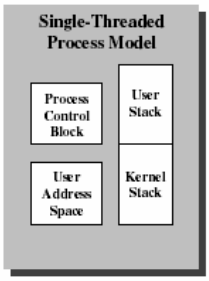
- Molti programmi per i moderni PC sono predisposti per essere eseguiti da processi multithread
- Un'applicazione, solitamente, è codificata come un processo a sè stante comprendente più thread di controllo
- Il multithreading è la capacità di un sistema operativo di supportare un thread di esecuzione multipli per ogni processo

Thread e Processi: quattro possibili scenari



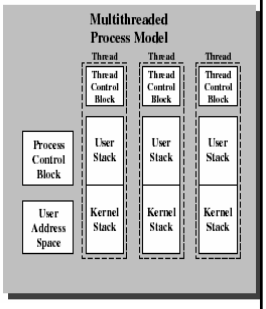
Modello dei processi a thread singolo

- In un modello a thread singolo la rappresentazione di un processo contiene il suo PCB e il suo spazio indirizzamento utente, lo stack utente e lo stack del kernel



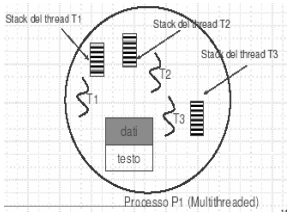
Modello multithread dei processi

- In un ambiente multithreading ogni processo ha associato:
  - un solo PCB
  - un solo spazio di indirizzamento utente
- …ma ogni thread ha un proprio
  - stack
  - un blocco di controllo privato contenente l'immagine dei registri
  - Una priorità
  - Altre informazioni relative al thread



Esempio: struttura processo multithread

- Tutti i thread componenti un processo:
  - condividono lo stato e le risorse del processo
  - risiedono nello spazio di indirizzamento
  - hanno accesso agli stessi dati



- **T1** legge i caratteri da tastiera e li visualizza
- **T2** formatta il testo
- **T3** memorizza periodicamente sul disco

Vantaggi

- La programmazione multithread offre numerosi vantaggi classificabili rispetto a 4 fattori principali:
  - Tempo di risposta
  - Condivisione delle risorse
  - Economia
  - Uso di più unità di elaborazione

Vantaggi:tempo di risposta

- Il multithread permette ad un programma di continuare la ua escuzione anche se una parte di esso è bloccata o sta eseguendo un'operazione particolarmente lunga
  - Es.: Programma di consultazione web
    - Un thread carica un'immagine
    - Un thread permette l'interazione con l'utente

Vantaggi:Condivisione delle risorse e Economia

- Tutti i thread condividono la memoria e le risorse del processo a cui appartengono
- Questa condivisione rende più conveniente creare thread e gestire cambiamenti di contesto piuttosto che creare nuovi processi

Vantaggi:Uso di più unità di elaborazione

- Nelle architetture con più unità di elaborazione i thread si possono eseguire in parallelo uno per ciascuna unità di elaborazione

Svantaggi

- Maggiore complessità di progettazione e programmazione
  - I processi devono essere "pensati" paralleli
  - Difficile sincronizzazione tra i thread
- Inadatto per situazioni in cui i dati devono essere protetti
- la condivisione delle risorse accentua il pericolo di interferenza

Funzionalità dei thread

- Come i processi, anche i thread hanno uno **stato di esecuzione** e possono **sincronizzarsi** fra loro.
- Esaminiamo questi aspetti uno alla volta

Stato dei thread

- Come per i processi anche per i thread gli stati chiave sono:
  - **Running** (sta girando…)
  - **Ready** (pronto per girare ma al momento il processore e occupato con un altro thread)
  - **Blocked**(sta aspettando qualcosa, es.I/O)
- Non ha senso associare stati "swapped" o "suspended" perchè sono operativi a livello di processo

Stato dei thread: Operazioni sui thread

- **Creazione**
  - Quando un nuovo processo viene creato si crea anche un thread.
  - Un thread di un processo può creare un nuovo thread per lo stesso processo fornendogli il puntatore alle istruzioni e gli argomenti
- **Blocco**
  - Quando un thread deve aspettare un evento, entra in stato Blocked (salvando i registri utente, il program counter e lo stack pointer)
- **Sblocco**
  - All'occorrenza dell'evento su cui il thread era bloccato, il thread passa in stato ready
- **Terminazione**
  - Quando un thread completa il suo compito, il suo contesto per i registri e i suoi stack vengono deallocati.

Sincronizzazione dei thread

- Poichè i thread condividono le risorse di uno stesso processo è necesario sincronizzare le attività cosi che essi non interferiscano l'uno con l'altro
- Es.:

  Se due thread cercano entrambi di aggiungere un elemento ad una lista con doppio puntatore un elemento può andare perso o la lista può diventare malformata

Thread al livello di utente e thread al livello di Kernel

- Livello Utente
  - Tutto il lavoro di gestione dei thread viene effettuato dall'applicazione
  - il kernel non conscio della presenza dei thread
  - Sono realizzati tramite una libreria di funzioni per la creazione, lo scheduling e la gestione senza alcun intervento diretto del kernel
- Livello Kernel
  - I thread sono gestiti direttamente dal sistema operativo
  - Il nucleo si occupa della creazione, scheduling e gestione dello spazio di indirizzi del kernel
  - Non c'è codice di gestione dei thread ma un'API per la componente del kernel che gestisce i thread

Vantaggi/Svantaggi: Livello utente

Vantaggi:
- Lo switching non coinvolge il kernel è quindi non ci sono cambiamenti della modalità di esecuzione
- Maggiore libertà nella scelta dell'algoritmo di scheduling che può anche essere personalizzato
- Siccome le chiamate possono essere raccolte in una libreria, c'è una maggiore portabilità tra SO

Svantaggi:
- una chiamata al kernel può bloccare tutti i thread di un processo, indipendentemente dal fatto che in realtà solo uno dei suoi thread ha causato la chiamata bloccante
- In sistemi SMP, due processori non risulteranno mai
- associati a due thread del medesimo processo

Vantaggi/Svantaggi: Livello Kernel

Vantaggi:
- il kernel può eseguire più thread dello stesso processo anche su più processori
- il kernel stesso può essere scritto multithread

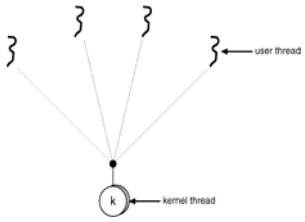
Svantaggi:
- lo switching coinvolge chiamate al kernel e questo, soprattutto in sistemi con molteplici modalità, comporta un costo
- l'algoritmo di scheduling è meno personalizzabile e meno portabile

Modelli di programmazione multithread

- Alcuni S.O. implementano sia thread di sistema che thread di utente.
- Questo genera differenti modelli di gestione dei thread:
  - Molti -a-Uno
  - Uno-ad-Uno
  - Molti -a-Molti

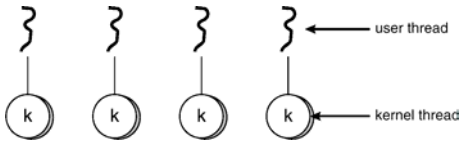
Modello da molti a uno

- Fa corispondere molti thread al livello d'utente ad un singolo thread al livello del nucleo
- La gestione è efficiente poichè si svolge nello spazio utente ma l'intero processo rimane bloccato se un thread invoca una chiamata di tipo bloccante
- Esempi: Solaris 2



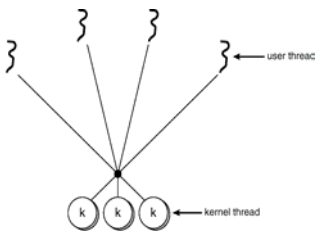
Modello da uno a uno

- Mette in corrispondenza ciascun thread del livello utente con un thread del livello kernel
- Offre un alto grado di concorrenza
  - Se un thread invoca una chiamata bloccante è possibile eseguire un altro thread
- Lo svantaggio di questo modello è che la creazione di ogni thread al livello utente comporta la creazione del corrispondente thread al livello del kernel
  - Maggiore carico (thread kernel più pesanti da gestire)
- Esempi: Windows 95/98



Modello da molti a molti

- Mette in corrispondenza più thread del livello utente con un numero minore o uguale di thread del livello kernel
- Permette al sistema operativo di creare un numero sufficiente thread kernel.
- Se un thread invoca una chiamata bloccante il kernel può fare in modo che si esegua un altro thread
- Esempi: Solaris 2 e Windows NT/2000



Pthreads

- Pthreads è un modello basato sull'API standard POSIX (IEEE 1003.1c) per la creazione e la sincronizzazione di thread.
- Le librerie Pthread specificano il comportamento dei thread e non la loro implementazione e i progettisti di SO possono realizzare le API come meglio credono.
  - Esempi di primitive:
    pthread_create(), pthread_join(), pthread_exit()
- Usato in diverse versioni di UNIX.

Multiprocessing simmetrico (SMP)

- Con l' evoluzione della tecnologia dei computer e la diminuzione dei costi hardwere, i progettisti hanno mostrato maggiore interesse per il parallelismo
  - Per migliorare le prestazioni
  - Per aumentare l' affidabilità
- I' approccio più popolare per realizzare il parallelismo utilizzando più processori è il Multiprocessing simmetrico

Architettura SMP(1)

- un sistema SMP è una macchina nella quale le applicazioni, per ottenere prestazioni sempre più elevate, hanno a disposizione più di un processore
- Flynn classifica i sistemi con processori paralleli in quattro categorie
  - Istruzione singola,dato singolo (SISD).
  - Istruzione singola,dati multipli (SIMD).
  - Istruzioni multiple,dato singolo (MISD).
  - Istruzioni multiple,dati multipli (MIMD).

Architettura SMP(2)

- **Istruzione singola,dato singolo (SISD):**
  - un singolo processore esegue una singola sequenza di istruzioni operando su dati memorizzati in una singola memoria
- **Istruzione singola,dati multipli (SIMD):**
  - una singola istruzione macchina controlla l' esecuzione simultanea di più elaborazioni,sincronizzate passo a passo
- **Istruzioni multiple,dato singolo (MISD):**
  - una sequenza di dati viene trasmessa da un insieme di processori ognuno dei quali esegue una diversa sequenza di istruzioni
- **Istruzioni multiple,dati multipli (MIMD):**
  - un insieme di processi esegue simultaneamente diverse sequenze di istruzioni su dati diversi

Perché MIMD?

- Le macchine MIMD sono flessibili possono:
  - funzionare come macchine a utente singolo con alte prestazioni su una certa applicazione
  - come multiprocessori multiprogrammati che eseguono simultaneamente molti compiti
  - come una combinazione di tali soluzioni;

Overview

- Introduzione
- Il modello a thread
- Multhreading
  - Thraed vs Processi
  - Vantaggi/Svantaggi
- Stati e operazioni sui thread
- Thread a livello utente e a livello kernel
- Modelli di programmazione multithread
  - Uno a Uno
  - Molti a Uno
  - Molti a Molti
- Multiprocessing simmetrico (SMP)
- Architetture SMP
- Architetture centralizzate
  - A memoria condivisa
  - A memoria distribuita
- Kernel e microkernel
- Architettura del kernel
  - Vantaggi/Svantaggi
- Thraed nei Sistemi Operativi
  - Windows NT
  - Windows 2000
  - Linux
  - Solaris

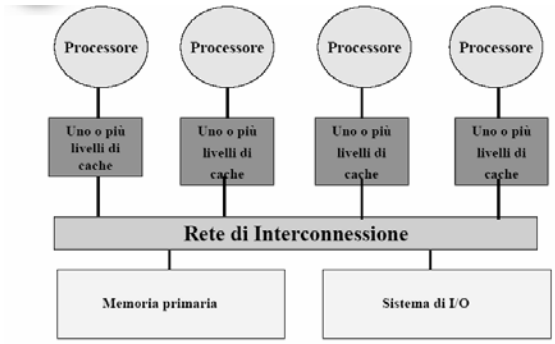
Architetture centralizzate

- I MIMD si possono ulteriormente suddividere in due classi di architetture:

1. ***Architetture multiprocessore a memoria condivisa***
2. ***Architetture multiprocessore a memoria distribuita***

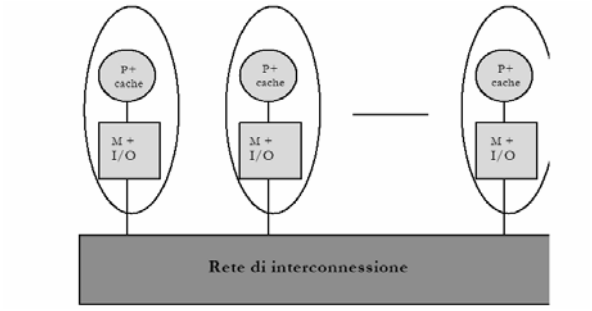
Architetture centralizzate a memoria condivisa (2)

- I vari processori contengono
  - L'unità di controllo
  - L'unità aritmetico-logica
  - Registri
  - Grandi cache associate alle singole CPU
- Ogni processore ha accesso alla memoria condivisa e ai dispositivi di I/O tramite un bus
- Bus e memoria condivisa possono soddisfare le richieste di un piccolo numero di processori.
- La comunicazione è effettuata mediante un ampio spazio di indirizzamento **condiviso**
  - Occorre gestire i "conflitti" per l'accesso alla memoria da parte di più processori
  - Sincronizzare gli accessi alla stessa locazione da parte di più processori.



Architetture a memoria distribuita

- Lo spazio degli indirizzi è costituito da più spazi di indirizzamento privati, logicamente disgiunti
- La comunicazione viene effettuata inviando messaggi che richiedono un'azione o forniscono dei dati;
- l'architettura può constare anche di computer totalmente separati collegati mediante una rete locale (***clusters***) in modo da costituire un approccio molto efficiente in termini di costo.



Kernel (1)

- Il **kernel** costituisce il **nucleo** di un sistema operativo.
- Si tratta di un software avente il compito di fornire ai processi in esecuzione sull'elaboratore un accesso sicuro e controllato all'hardware.
- Il kernel ha anche la responsabilità di assegnare una porzione di tempo-macchina e di accesso all'hardware a ciascun programma (dato che possono esserne eseguiti simultaneamente più di uno)

Kernel (2)

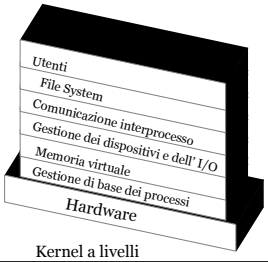
- I kernel si possono classificare - in base al grado di astrazione dell'hardware - in quattro categorie:
  - ***Kernel monolitici***, che implementano direttamente una completa astrazione dell'hardware sottostante.
  - ***Microkernel***, che forniscono un insieme ristretto e semplice di astrazione dell'hardware e usano software per fornire maggiori funzionalità.
  - ***Kernel ibridi*** (o *microkernel modificati*), che si differenziano dai microkernel puri per l'implementazione di alcune funzioni aggiuntive al fine di incrementare le prestazioni.
  - ***Esokernel***, che rimuovono tutte le limitazioni legate all'astrazione dell'hardware e si limitano a garantire l'accesso concorrente, permettendo alle singole applicazioni di implementare autonomamente le tradizionali astrazioni del sistema operativo per mezzo di speciali librerie

Microkernel

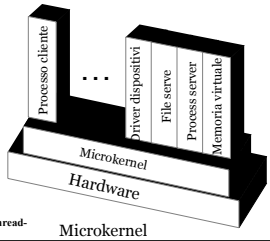
- La "filosofia" del microkernel:
  - Solo le funzioni assolutamente essenziali del nucleo del sistema operativo dovrebbero essere nel kernel;
  - I sevizi meno essenziali e le applicazioni sono costruiti sopra il microkernel e vengono eseguiti in modalità utente.
- La linea di separazione fra cosa è dentro e cosa è fuori dal microkernel varia da un progetto all'altro.

Architettura del Kernel

- L'architettura a microkernel sostituisce la tradizionale stratificazione verticale dei SO con una orizzontale.
- I componenti del SO esterni al microkernel sono implementati come processi server
  - Interagiscono fra di loro su una base di parità, tramite passaggio di messaggi attraverso il microkernel



Kernel a livelli



Microkernel

Vantaggi di un'organizzazione a microkernel (1)

- I vantaggi derivanti dall'uso di microkernel sono:
  - **Interfaccia uniforme**
    - I processi non devono fare distinzione fra i servizi al livello kernel e al livello utente, perché ogni servizio è fornito tramite passaggio di messaggi
  - **Estensibiltà**
    - È permessa l'aggiunta di nuovi servizi senza la creazione di un nuovo kernel
    - Quando si aggiunge una nuova caratteristica, è necessario modificare o aggiungere solamente i server che la riguardano
  - **Flessibilità**
    - Le caratteristiche possono essere aggiunte ma anche rimosse per ottenere un'implementazione più snella ed efficiente

Vantaggi di un'organizzazione a microkernel (2)

- **Portabilità**
  - Tutto il codice che dipende dal tipo di processore è contenuto nel kernel, così le modifiche necessarie a trasferire il sistema su un nuovo processore sono minori
- **Affidabilità**
  - Maggiore è la dimensione di un prodotto software, più è difficile garantire l'affidabilità
  - Un Microkernel piccolo può essere testato rigorosamente
- **Supporto ai sistemi distribuiti**
  - Un processo può inviare un messaggio senza conoscere la macchina su cui il destinatario risiede

Svantaggi del microkernel

- **Prestazioni**
  - Costruire un messaggio , inviarlo, accettarlo e decodificare la risposta, richiede molto più tempo che effettuare una singola chiamata a servizio
  - La progettazione a microkernel porta ad una implementazione modulare e flessibile, ma le prestazioni rimangono un problema ancora aperto.

I Thread nei Sistemi operativi (1)

- I processi supportati da diversi SO differiscono sotto molti aspetti, tra cui:
  - Come sono chiamati i processi
  - Presenza o meno di thread all'interno dei processi
  - Come sono rappresentati i processi
  - Come vengono protette le risorse dei processi
  - Quali meccanismi sono usati per la comunicazione e per la sincronizzazione
  - Come sono correlati i processi

I Thread nei Sistemi operativi (2)

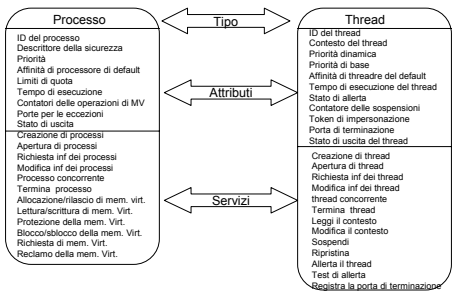
- Panoramica dei thread in:
  - Windows NT
  - Windows 2000
  - Solaris
  - Linux

Thread in Windows NT (1)

- Le strutture per i processi e i servizi forniti dal kernel di NT sono "semplici e generici".
- Le caratteristiche più importanti dei processi di NT sono:
  - I processi di NT sono implementati come oggetti
  - Un processo eseguibile può contenere uno o più thread
  - Gli oggetti di tipo processo e di tipo thread incorporano capacità di sincronizzazione

Thread in Windows NT (2)

- Cosa intendiamo per "*oggetti di tipo processo e di tipo thread*" ?

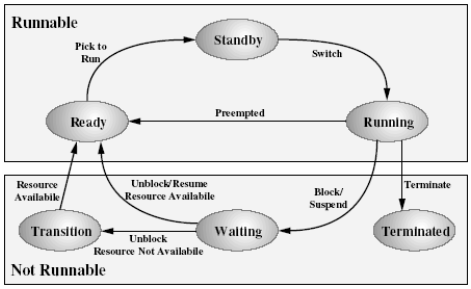


Thread in Windows NT (3)

- Un processo effettua un servizio quando riceve un messaggio appropriato;
- L'unico modo di chiamare tale servizio è tramite messaggi ad un oggetto di tipo processo che fornisce quel servizio.
- Un processo in NT deve contenere almeno un thread per poter essere eseguito, tale thread può poi creare altri thread.
- In un sistema multiprocessore, thread multipli dello stesso processo possono essere eseguiti in parallelo.
  - Esempio: un processo server può servire molti clienti e ogni richiesta dei clienti provoca la creazione di un nuovo thread del server

Windows NT: Stati dei thread

- Un thread di NT, se esiste, è in uno dei sei stati seguenti:



Windows NT: Stati dei thread (1)

- **Ready**:
  - Il thread può essere schedulato per l'esecuzione.
  - L'allocatore del microkernel tiene traccia di tutti i thread in stato Ready e li schedula in ordine di priorità.
- **Standby**:
  - il thread è stato scelto per un particolare processore;
  - Aspetta in questo stato finchè quel processore è disponibile.
  - Se la priorità del thread in stato Standby è sufficientemente alta,
    - il thread in esecuzione su quel processore può essere sostituito da quello in attesa;
    - altrimenti, il thread in stato Standby aspetta finchè il thread in esecuzione si blocca o termina il suo quanto di tempo.

Windows NT: Stati dei thread (2)

- **Running**:
  - Quando il mcrokernel esegue un cambio di thread o di processo, il thread in standby entra in stato Running e comincia l'esecuzione, continuando finchè non viene interrotto oppure finisce il quanto di tempo, si blocca, o termina.
  - Nei primi due casi ritorna nello stato Ready.
- **Waiting**:
  - Un thread entra in stato Waiting quando:
    1. È bloccato su un evento (ad es. I/O)
    2. Aspetta volontariamente per ragioni di sincronizzazione
    3. Un sottostistema dell'ambiente forza il thread ad auto-sospendersi
  - Quando la condizione di attesa è soddisfatta, il thread passa allo stato Ready se tutte le sue risorse sono disponibili

Windows NT: Stati dei thread (3)

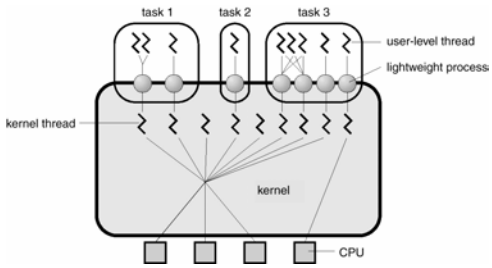
- **Transition**:
  - un thread entra in questo stato dopo lo stato Waiting se è pronto per l'esecuzione, ma le risorse non sono disponibili.
  - Quando le risorse diventano disponibili, il thread passa allo stato Ready
- **Terminated**:
  - Un thread può terminare
    1. per sua richiesta
    2. per richiesta di un altro thread
    3. quando il suo processo genitore termina.
  - Quando le strutture del thread sono state eliminate, esso può essere rimosso dal sistema, oppure conservato per essere inizializzato nuovamente in futuro

Thread in Solaris (1)

- Solaris fa uso di quattro concetti distinti correlati ai thread:
  1. **Processi**
  2. **Thread a livello utente (ULT)**
  3. **Processi leggeri**
     - Un processo leggero si può vedere come una mappatura di ULT in thread del kernel
  4. **Thread del kernel**

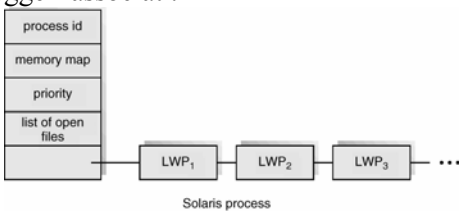
Thread in Solaris (2)

- C'è esattamente un thread del kernel per ogni processo leggero
- Un processo leggero all'interno di un processo è visibile all'applicazione



…ancora Solaris

- Oltre alle normali informazioni, un processo Solaris contiene le informazioni sui processi leggeri associati.



Thread di Windows 2000

- Ogni thread utente è associato ad un thread del kernel (modello uno-ad- uno).
- Con la libreria *fiber* Windows fornisce anche un modello molti-a-molti
- Ogni thread contiene
  - un identificatore del thread
  - un insieme di registri
  - uno stack utente e uno stack kernel
  - un'area di memoria privata del thread.

Thread di Linux (1)

- Introdotti a partire dalla versione 2.2 del kernel
- Linux si riferisce ad essi come *task* e non come *thread*
- Linux usa la stessa rappresentazione interna per processi e thread;
- un thread è semplicemente un nuovo processo che condivide lo stesso spazio di indirizzamento del padre

Thread di Linux (2)

- La differenza tra processo e thread è evidente al momento della creazione di un thread con la chiamata a sistema **clone**
  - **fork** crea un nuovo processo con il proprio nuovo contesto
  - **clone** crea un nuovo processo con la propria identità a cui è permesso di condividere le strutture dati del proprio genitore
- L'uso di **clone** permette alle applicazioni di controllare nel dettaglio la condivisione di strutture dati tra due threads
- Per ogni processo esiste un'unica struttura dati nel nucleo

Bibliografia

- I seguenti argomenti sono stati tratti da "SISTEMI OPERATIVI CONCETTI ED ESEMPI" di SILBERSHATZ:
  - Introduzione
  - Il modello a thread
  - Multhreading
  - Stati e operazioni sui thread
  - Thread a livello utente e a livello kernel
  - Modelli di programmazione multithread
- I seguenti argomenti sono stati tratti da "I SISTEMI OPERATIVI" di STALLING:
  - Multiprocessing simmetrico (SMP)
  - Architetture SMP
  - Architetture centralizzate
  - Kernel e microkernel
  - Architettura del kernel
  - Thraed nei Sistemi Operativi