

Molti SO forniscono meccanismi per la comunicazione tra processi (IPC), che comunicano fra loro in ambienti multiprogrammati e in rete (es. Un Web recupererà dati da un server remoto), è essenziale per i processi che devono coordinare le attività per raggiungere un obiettivo comune

Segnali

Le interruzioni software che notificano ad un processo l'occorrenza di un evento che non permette ai processi di specificare dati da scambiare con altri processi, che possono ricevere, ignorare o mascherare un segnale:

- **Ricevere** un segnale comporta specificare
- **Ignorare** un segnale si appoggia sull'azione di default del SO per gestire il segnale
- **Mascherare** un segnale indica al SO di non consegnare segnali di quel tipo fino a quando il processo cancella quel mascheramento di segnale, **sospensione temporanea**, non fa vedere il processo

Impedendo la ricezione e l'invio, una procedura che tratta il segnale

Interprocess communication (IPC) è basata sullo scambio di messaggi (Message Passing) e tali messaggi possono essere trasmessi in una direzione alla volta (un processo è il mittente e l'altro è il ricevente)

Lo scambio di messaggi può essere bidirezionale, in pratica ogni processo può agire sia come un mittente o un ricevitore e questi messaggi possono essere bloccati o non bloccati (primitive (send, receive)), il blocco richiede al ricevente di notificare al mittente quando viene ricevuto il messaggio e il non bloccante permette al mittente di continuare altre elaborazioni, controlla se è avvenuto l'invio

L'implementazione comune è **pipe** che è una regione di memoria protetta dal SO che funge da **buffer**, consentendo a due o più processi di **scambio dati**

L'IPC nei **sistemi distribuiti**, i messaggi trasmessi possono essere corrotti o **pesimisti**, sono meccanismi di riscontro, protocolli dove prevedono un invio di un messaggio della ricezione del messaggio

- **Protocolli** di conferma, confermano che le trasmissioni siano stati correttamente ricevuti (acknowledgement)
- **Meccanismi** di timeout con ritrasmissione messaggi se gli ack non vengano ricevuti

I processi denominati in modo ambiguo portano ad errori di riferimento di messaggio, che vengono passati tra sistemi che utilizzano porte numerate sulle quali i processi ascoltano, evitando questo problema, come identifico il processo destinatario

Il garante dell'autenticazione si occupa dei problemi di sicurezza

Thread

I fattori che hanno influito sullo sviluppo dei threads sono:

1. **Progetto Software:** soddisfa la modularità (parallelismo), e consiste nell'esprimere in modo più naturale le attività intrinsecamente parallele, quindi anche la concorrenza, in pratica alcune attività si sospendono temporaneamente
2. **Prestazioni:** si adatta meglio ai sistemi multiprocessore siccome è più leggero del processo, più veloce da gestire quindi viene sfruttato meglio se è in I/O bound (non CPU bound (Orientato alla CPU))
3. **Cooperazione:** condivisione dello spazio degli indirizzi e dei dati, quindi c'è una riduzione dell'overhead dovuto alla IPC

La definizione di **Thread** è di un processo lightweight (LWP: LightWeight Process) e sono un flusso di istruzioni o flusso di controllo, condividono con il proprio processo lo spazio di indirizzamento, file aperti e altre informazioni globali, registri, stack, maschere dei segnali e altri dati specifici del thread (TSD: Thread Specific Data) sono locali per ogni thread e possono essere gestiti dal SO o da un'applicazione utente

Processo: Heavyweight process

Thread: Lightweight process

Quando cambio thread le info comuni rimangono ma quelle personali si modificano

Ogni thread si sposta fra una serie di stati discreti e ha molte operazioni comuni con i processi (es.: creazione, terminazione, ripresa e sospensione), la creazione di un thread non richiede al SO di inizializzare le risorse condivise tra i processi padri e i suoi thread, questo riduce l'overhead di creazione e terminazione dei thread rispetto alla creazione e terminazione di un processo

Differenze fra thread e processi

Un processo contiene e raggruppa risorse in relazione fra loro (spazio di indirizzamento, altre risorse, ...) e un thread è in esecuzione su un programma ed è schedulato per essere eseguito sul processore, scelto dallo scheduler (dipende dalla CPU), operazioni come, PC, registri per le variabili e stack per chiamate di procedura, condivide lo spazio di indirizzamento e altre risorse con altri thread dello stesso processo, quindi nei Multithreading in uniprocessori con parallelismo virtuale dei thread, quindi gli elementi del processo sono:

1. Contatore di programma (PC)
2. Registri
3. Stack
4. Stato

E servono per il cambiamento di contesto e sono privati ad ogni thread rispetto al processo

Gli stati di un thread sono analoghi a quelli del processo (Ready, Blocked, Running) ma viene gestito dallo scheduler dei thread, più in dettaglio possiamo distinguere gli stati:

1. Born
2. Ready (stato pronto)
3. Running
4. Dead
5. Blocked (stato bloccato da un evento di I/O esterno)
6. Waiting (stato bloccato da un evento di un altro thread interno e predefinito)
7. Sleeping (stato bloccato per un tempo prefissato interno e predefinito) e specifica per quanto tempo un thread rimarrà in tale stato (es autosalvataggio)

I threads e i processi hanno delle operazioni in comune:

- Creazione
- Exit (terminazione)
- Sospensione
- Recupero (resume)
- Sleep
- Risveglio

Le operazioni sui thread non corrispondono esattamente alle operazioni sui processi:

- **Cancellazione:** indica che un thread dovrebbe essere terminato, ma non è garantito che lo sarà e possono disabilitare o mascherare i segnali di cancellazione (eccetto la abort)
- **Join:** se esiste un thread primario quando termina allora termina anche il processo e il thread primario può dover aspettare che tutti gli altri thread siano terminati, in tal caso ha fatto una join con i thread creati da lui e se un thread A fa una join con il thread B, allora A non va in esecuzione finché B non ha terminato (indicazione di relazione)

Lo standard POSIX (standard IEE) il suo obiettivo è la portabilità, usato anche su altri sistemi, i thread che utilizzano questo standard si chiamano Package di thread (Pthread), ed è usato prevalentemente dai sistemi Unix ed include oltre 60 chiamate di sistema ed hanno:

- Un identificatore
- Un insieme di registri (PC, ...)
- Un insieme di attributi (parametri di scheduling, di stack, ...)

Thread a livello utente

Dipende da dove vengono definite, eseguono operazioni di threading nello spazio utente e sono creati da librerie a tempo di esecuzione che non possono eseguire istruzioni privilegiate o accedere direttamente al nucleo.

La loro **implementazione** viene chiamata, Mapping di thread **multi-a-uno**, in pratica il SO associa tutti i threads di un processo multithread ad un unico contesto di esecuzione, ed ha vantaggi e svantaggi:

- **Vantaggi:** le librerie di livello utente possono schedulare thread per ottimizzare le prestazioni, la sincronizzazione avviene al di fuori del kernel, evitando il cambio di contesto e permette ai thread (da utente) dove il SO non supporta i thread (interni), quindi **Portabilità**
- **Svantaggi:** il nucleo vede un processo multithread come un singolo thread di controllo, può portare a prestazioni non ottimali se un thread ha problemi di I/O e non può essere schedulato su più processori in una sola volta

Più thread a livello utente dello stesso processo condividono l'unico contesto di esecuzione e quelli che sono gestiti a livello utente sono raggruppati per

ogni processo in una **tabella di thread** e viene gestita da un sistema a tempo di esecuzione che mantiene la lista dei thread bloccati e la lista dei thread pronti

Il cambio di contesto fra thread interni ad un processo è molto più rapido rispetto al cambio di contesto fra processi, scheduling ad hoc

Thread nello spazio kernel

I thread a livello nucleo cercano di superare i limiti del thread a livello utente mappando ogni thread al proprio contesto di esecuzione e forniscono un mapping di thread **uno-a-uno**

Questa implementazione ha dei:

- **Vantaggi:** Aumento della scalabilità, interattività e throughput (quanti thread posso eseguire, efficienza)
- **Svantaggi:** overhead dovuti al cambio di contesto e ridotta portabilità dovuto alle API specifici per il SO

I thread a livello nucleo non sono sempre la soluzione ottima per le applicazioni multithread

Non occorre una tabella di thread e un sistema di esecuzione real-time per ogni processo e siccome il nucleo ha la tabella di thread e anche quella dei processi con i registri, stato e informazioni di tutti i thread

Le operazioni sui thread avvengono con chiamate al nucleo, quindi il costo è maggiore del cambiamento di contesto però è possibile “riciclare” i thread (ne creano di default, Pool di thread)

Lo scheduler è a livello nucleo e può confrontare thread di processi diversi, un thread bloccato può non bloccare tutti il processo

Thread Combinati a livello Utente e Nucleo

La combinazione di implementazione dei thread a livello utente e livello nucleo utilizza il Mapping thread **multi-a-multi** (m-to-n thread mapping), in pratica il numero di thread di livello utente e livello kernel non deve essere uguale e può ridurre l'overhead rispetto al mapping uno-a-uno implementando il thread pooling

I Worker threads sono threads di livello nucleo persistenti che occupano il pool di thread, questo migliora le prestazioni in ambienti in cui i threads sono

spesso creati e distrutti e ogni nuovo thread viene eseguito da un thread worker

L' **attivazione dello Scheduler** è tecnica che permette alla libreria di livello utente di programmare i suoi thread viene attuata quando il SO chiama una libreria di threading a livello utente che determina se uno qualsiasi dei suoi thread devono essere rischedulati

L'attivazione dello scheduler: ha l'obiettivo di mimare le funzionalità dei threads di livello nucleo e migliorare le prestazioni dei thread del livello utente, per evitare transazioni non necessarie tra utente o kernel, il nucleo assegna processori virtuali ad ogni processo che:

- Permette al sistema di allocare a tempo di esecuzione i thread ai processori (a livello utente)
- Segnalazioni dal nucleo per possibili situazioni di thread bloccati (upcall) al sistema a run-time che può attivare lo scheduler a livello utente

Si può usare sui multiprocessore (con processori reali), ma il problema è che c'è l'uso intensivo del nucleo (basso livello) che chiama procedure nello spazio utente (alto livello)

Standard POSIX	
Chiamata	Descrizione
Pthread_create	Crea un nuovo thread
Pthread_exit	Termina il thread chiamante
Pthread_join	Attende che un thread specifico esca
Pthread_yield	Rilascia la CPU perché venga eseguito un altro thread
Pthread_attr_init	Crea e inizializza la struttura attributi di un thread
Pthread_attr_destroy	Rimuove la struttura attributi di un thread

Alcune chiamate di Pthread

A. Tanenbaum - Modern Operating Systems

S. Balsamo - Università Ca' Foscari Venezia - SO4.55

Modelli di thread

- Tre modelli di threading

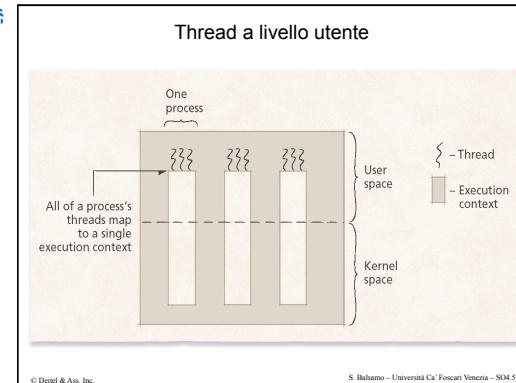
- Threads a livello di utente
- Threads a livello di kernel

- Combinazione (ibridi) di threads a livello di utente e a livello kernel *Scegli uno*

S. Balsamo - Università Ca' Foscari Venezia - SO4.56

Thread a livello utente	
<ul style="list-style-type: none"> Threads a livello utente eseguono operazioni di threading nello spazio utente <ul style="list-style-type: none"> Threads sono creati da librerie a tempo di esecuzione che non possono eseguire istruzioni privilegiate o accedere direttamente al nucleo Implementazione dei thread a livello utente <ul style="list-style-type: none"> Mapping di thread molto-a-uno S.O. associa tutti i threads di un processo multithread ad un unico contesto di esecuzione Vantaggi <ul style="list-style-type: none"> Le librerie di livello utente possono schedulare thread per ottimizzare le prestazioni La sincronizzazione avviene al di fuori del kernel, evitando il cambio di contesto Permette i thread (da utente) dove il s.o. non supporta i thread (interni) Portabilità Svantaggi <ul style="list-style-type: none"> Il nucleo vede un processo multithread come un singolo thread di controllo Può portare a prestazioni non ottimali se un thread ha problemi di I/O Non può essere schedulato su più processori in una sola volta 	<i>Dipende da dove vengono definiti</i>

S. Balsamo - Università Ca' Foscari Venezia - SO4.57



S. Balsamo - Università Ca' Foscari Venezia - SO4.58

Classi di Interrupt	
Classi di eccezioni nell'architettura Intel IA-32	
Classe di Eccezione	Descrizione
Fault (guasto)	Diverse possibili cause durante l'esecuzione di istruzioni di linguaggio macchina. Es. divisione per zero, dati utilizzati che sono in formato errato, tentativo di eseguire un codice di operazione non valido, tentativo di riferire una locazione di memoria in una zona proibita, tentativo da parte di un processo utente di eseguire istruzioni privilegiate o riferire risorse protette.
Trap	Generata da eccezioni quali overflow e quando il programma si trova in un breakpoint nel codice.
Abort	Un processore identifica un errore dal quel non si può recuperare l'esecuzione di un processo. Es. quando l'esecuzione di una routine che gestisce una eccezione a sua volta causa una eccezione, il processore potrebbe non riuscire a gestire entrambi gli errori sequenzialmente. Si parla di eccezione double-fault che porta alla terminazione del processo che l'ha attivata.

© Deitel & Associates Inc.

S. Balsamo - Università Ca' Foscari Venezia - SO3.32

Interprocess Communication	
• Molti SO forniscono meccanismi per la comunicazione tra processi (IPC)	
– I processi devono comunicare tra loro in ambienti multiprogrammati e in rete	<ul style="list-style-type: none"> Esempio: un Web browser recupera dati da un server remoto
– Essenziale per i processi che devono coordinare le attività per raggiungere un obiettivo comune	

S. Balsamo - Università Ca' Foscari Venezia - SO3.33

Segnali	
<ul style="list-style-type: none"> Interruzioni software che notificano ad un processo l'occorrenza di un evento <ul style="list-style-type: none"> – Non permette ai processi di specificare dati da scambiare con altri processi – I processi possono ricevere, ignorare o mascherare un segnale <ul style="list-style-type: none"> • Ricevere un segnale comporta specificare una routine che il SO chiama quando manda il segnale • Ignorare un segnale si appoggia sull'azione di default del SO per gestire il segnale • Mascherare un segnale indica al SO di non consegnare segnali di quel tipo fino a quando il processo cancella quel mascheramento di segnale 	<p><i>specificare</i></p> <p><i>ignorare la ricezione e l'invio, una preceduta che tratta il seguente</i></p> <p><i>sospensione temporanea, non fa vedere il processo</i></p>

S. Balsamo - Università Ca' Foscari Venezia - SO3.34

Message Passing	
<ul style="list-style-type: none"> – Interprocess communication basata su scambio di messaggi <ul style="list-style-type: none"> – I messaggi possono essere trasmessi in una direzione alla volta <ul style="list-style-type: none"> • Un processo è il mittente e l'altro è il ricevitore – Lo scambio di messaggi può essere bidirezionale <ul style="list-style-type: none"> • Ogni processo può agire sia come un mittente o ricevitore – I messaggi possono essere bloccanti o non bloccanti <ul style="list-style-type: none"> • Il blocco richiede al ricevente di notificare al mittente quando viene ricevuto il messaggio • Non bloccante permette al mittente di continuare altre elaborazioni – L'implementazione comune è pipe <ul style="list-style-type: none"> • Una regione di memoria protetta dal SO che funge da buffer, consentendo a due o più processi di scambiare dati 	<p><i>14.00</i></p> <p><i>mittente (send/receive)</i></p> <p><i>controlla se c'è avvenuto l'inizio</i></p>

S. Balsamo - Università Ca' Foscari Venezia - SO3.35

Message Passing

Mecanismi di riscontro, protocolli: dove prendono un ruolo di meccanismo della ricezione del messaggio

- IPC in sistemi distribuiti
 - I messaggi trasmessi possono essere corrotti o persi
 - protocolli di conferma **confermano** che le trasmissioni siano stati correttamente ricevuti (acknowledgement)
 - Meccanismi di **timeout** con ritrasmissione messaggi se gli ack non vengono ricevuti
 - I processi **denominanti** in modo ambiguo portano ad errori di riferimento di messaggio
 - I messaggi sono passati tra sistemi che utilizzano porte numerate sulle quali i processi ascoltano, evitando questo problema
 - La sicurezza: problema
 - garantire l'autenticazione

S. Balsamo – Università Ca' Foscari Venezia – SO3.36

Caso di studio: processi UNIX

- Processo UNIX
 - Tutti i processi hanno un insieme di indirizzi di memoria, chiamato **spazio di indirizzi virtuali** *~ spazio di indirizzamento*
 - Il PCB di un processo è mantenuto dal kernel in una regione protetta della memoria che i processi utente non possono accedere
 - Un **PCB** UNIX memorizza:
 - Il contenuto dei registri del processore
 - PID
 - Il contatore di programma
 - Lo stack di sistema
 - Tutti i processi sono elencati nella **tabella dei processi**

S. Balsamo – Università Ca' Foscari Venezia – SO3.37

Caso di studio: processi UNIX

- Processo UNIX
 - Tutti i processi interagiscono con il SO con **chiamate di sistema**
 - Un processo può generare un processo figlio usando la chiamata di sistema **fork**, che crea una copia del processo padre
 - Il processo figlio riceve una **copy** delle risorse del genitore (immagini di memoria distinte)
 - I file aperti dal processo padre sono condivisi con il figlio
 - La fork restituisce 0 al figlio e il PID del figlio al genitore
 - le priorità dei processi sono numeri interi in [-20, 19]
 - Un valore di priorità numerica inferiore indica una priorità più alta di scheduling
 - UNIX fornisce meccanismi IPC, come **pipes**, per consentire a processi diversi di trasferire i dati

S. Balsamo – Università Ca' Foscari Venezia – SO3.38

Caso di studio: processo UNIX

Chiamate di sistema UNIX

Chiamata di sistema	Descrizione
fork	Crea un processo figlio e alloca una copia delle risorse del processo padre al figlio
exec	Carica da un file le istruzioni e dati di un processo nel suo spazio di indirizzamento
wait	Il processo chiamante si blocca fino a quando il processo figlio non ha terminato
signal	Permette ad un processo di specificare un gestore di segnalazione per un dato tipo di segnale
exit	Termina il processo chiamante
nice	Modifica la priorità del processo usata dallo scheduling

© Detel & Ass. Inc.

S. Balsamo – Università Ca' Foscari Venezia – SO3.39

2 – Processi e Thread

39

<p>Processi</p> <ul style="list-style-type: none"> modello operazioni: creazione, chiusura gerarchie stati, ciclo di vita transizioni di stato descrittore di processo Process Control Block (PCB) sospensione, ripresa, cambio di contesto Interrupt comunicazione tra processi: segnali e messaggi 	<p>Sommario</p>
--	------------------------

Thread

- modello e uso**
- Scheduling**
- Obiettivi**
- Scheduling di processi: algoritmi**
- Vari tipi di sistemi**
- Scheduling di thread**

S. Balsamo – Università Ca' Foscari Venezia – SO2.39

Obiettivi

40

- Motivazioni per creare i **thread**
- le somiglianze e le differenze tra i **processi e thread**
- i diversi **livelli di supporto** per i thread
- il **ciclo di vita** di un thread
- segnalazione e cancellazione di thread
- le basi di thread Linux e Windows

S. Balsamo – Università Ca' Foscari Venezia – SO2.40

Motivazioni per l'uso dei thread

41

- **Fattori che hanno influito sullo sviluppo dei threads**
 - **Progetto Software**
 - Modularità → parallelismo
 - Esprimere in modo più naturale attività intrinsecamente **parallele**
 - **Concidenza**
 - Alcune attività si sospendono temporaneamente
 - **Prestazioni**
 - Si adatta meglio ai sistemi multiprocessore
 - Il thread è **più leggero** del processo, più veloce da gestire
 - Migliore sfruttamento se i thread sono I/O bound (non CPU bound)
 - **Cooperazione**
 - **condivisione** dello spazio degli indirizzi e dei dati → riduzione dell'overhead dovuto alla IPC

S. Balsamo – Università Ca' Foscari Venezia – SO2.41

Definizione di Thread

42

- **Thread**
 - Processo **lightweight** (LWP) → **Definizione di Thread**
 - **Threads** → flusso di istruzioni o flusso di controllo
 - condividono con il proprio processo lo **spazio di indirizzamento**, file aperti e altre informazioni globali
 - Registri, stack, maschere dei segnali e altri dati specifici del thread (**TSD Thread Specific Data**) sono locali per ogni thread
- I threads possono essere gestiti
 - dal S.O.
 - o da un'applicazione utente

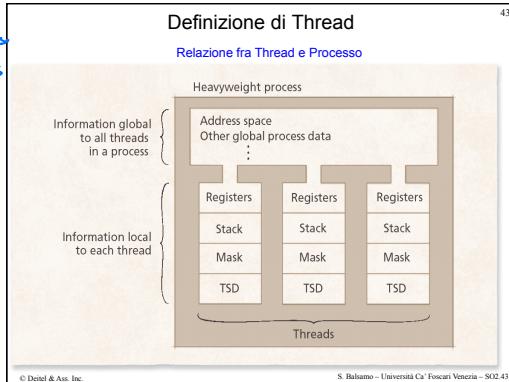
Esempi:

- Win32 threads – sistemi Windows a 32 bit
- C-threads – libreria microkernel Mach, anche su Solaris, Windows NT
- Pthreads – specifica POSIX portabile su vari S.O. **portati su S.O.**

S. Balsamo – Università Ca' Foscari Venezia – SO2.42

Processo: Heavyweight process
Thread: Lightweight process

Quando cambio Thread le info
 come rimangono ma quelle
 presenti si modificano



Motivazioni ai thread

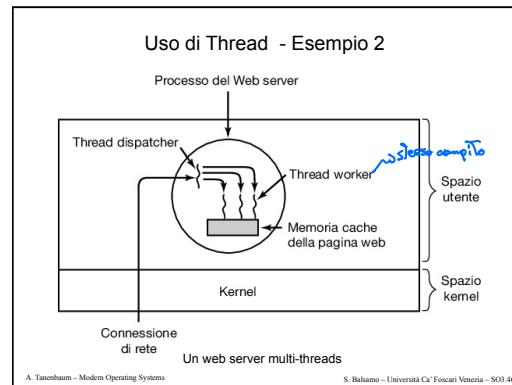
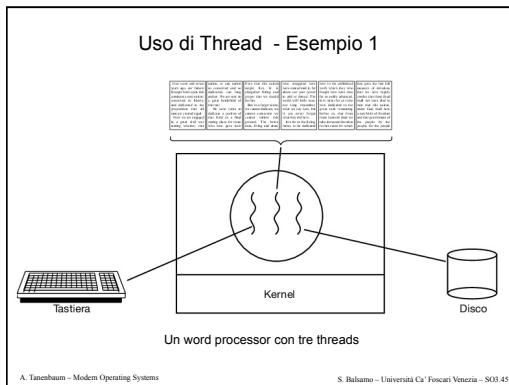
- Ogni **thread** si sposta fra una serie di **stati discreti**
- Thread e processi hanno molte operazioni in comune (es.: creazione, terminazione, ripresa e sospensione)
- La creazione di thread non richiede al S.O. di inizializzare le risorse condivise tra i processi padri e i suoi thread**
 - **Riduce l'overhead di creazione e terminazione** dei thread rispetto alla creazione e terminazione di un processo

grande d'Thread
 overhead minore, meno oper. da svolgere

Esempi di utilizzo di thread:
 Server (es. web server)
 Word processors

© Deniel & Ass. Inc.

S. Balsamo - Università Ca' Foscari Venezia - SO2.44



Uso di Thread - Esempio 2

```

while (TRUE) {
    get next request(&buf);
    handoff work(&buf);
}

while (TRUE) {
    wait for work(&buf);
    look for page in cache(&buf, &page);
    if (page not in cache(&page)) {
        read page from disk(&buf, &page);
        return page(&page);
    }
}

```

(a) (b)

Un web server **multi-thread**: schema del codice

(a) Thread dispatcher (b) Thread worker

Multiprogrammazione
Incremento delle prestazioni

A. Tanenbaum - Modern Operating Systems

S. Balsamo - Università Ca' Foscari Venezia - SO3.47

Differenze fra thread e processi

Un **processo** contiene e **raggruppa risorse** in relazione fra loro

- spazio di indirizzamento con testo/dati/pila
- altre risorse
- file
- processi figli
- stato e segnali
- account
- ...

Un **thread** è esecuzione di un programma

- è **schedulato** per essere eseguito sul processore
- Program Counter
- registri per le variabili
- stack per chiamate di procedura
- condivide** lo spazio di indirizzamento e altre risorse con gli altri thread dello stesso processo

Multithreading

in uniprocessori: parallelismo virtuale dei thread

S. Balsamo - Università Ca' Foscari Venezia - SO3.48

sotto della schedula (dipende dalla CPU)

Modello Thread

Elementi per processo	Elementi per thread
Spazio degli indirizzi	Contatore di programma
Variabili globali	Registri
File aperti	Stack
Processi figli	Stato
Allarmi in sospeso	
Segnali e gestori dei segnali	
Informazioni relative agli account	

servono per i cambiamenti di contesto

Elementi **condivisi** da tutti i thread in un processo - Elementi **privati** ad ogni thread

A. Tanenbaum - Modern Operating Systems

S. Balsamo - Università Ca' Foscari Venezia - SO3.49

Stati di un thread: ciclo di vita

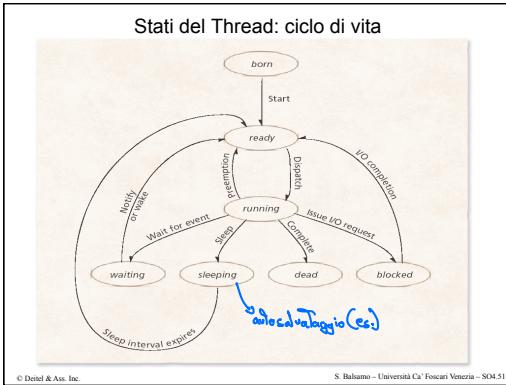
- Stati del thread analoghi agli stati del processo
- Più in dettaglio possiamo distinguere gli stati
 - 1 - Born
 - 2 - Ready (stato pronto)
 - 3 - Running (stato in esecuzione)
 - 4 - Dead
 - 5 - Blocked (stato bloccato da un evento di I/O)
 - 6 - Waiting (stato bloccato da un evento di un altro thread)
 - 7 - Sleeping (stato bloccato per un tempo prefissato)
- L'intervallo di sleep specifica per quanto tempo un thread rimarrà in tale stato

Scheduler dei Thread

Coda Thread : Pronto e Bloccati

settempo

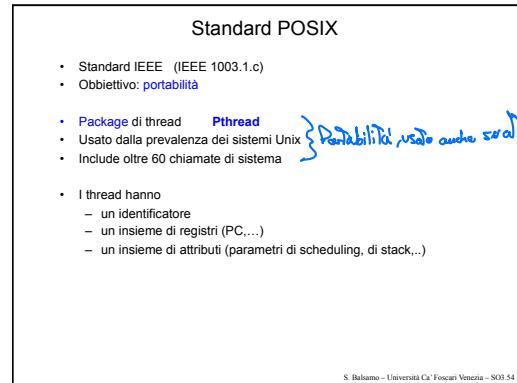
S. Balsamo - Università Ca' Foscari Venezia - SO4.50

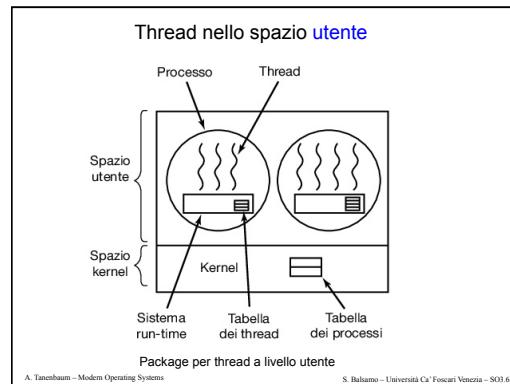
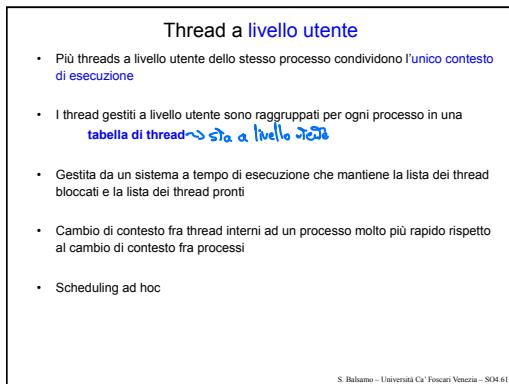
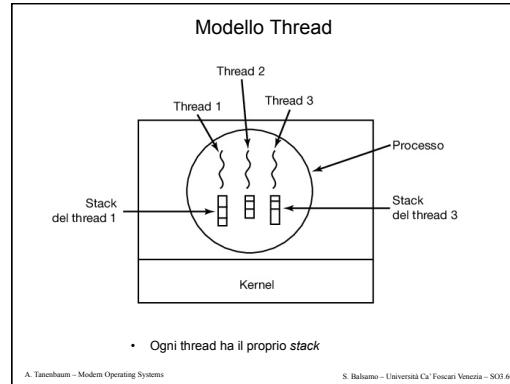
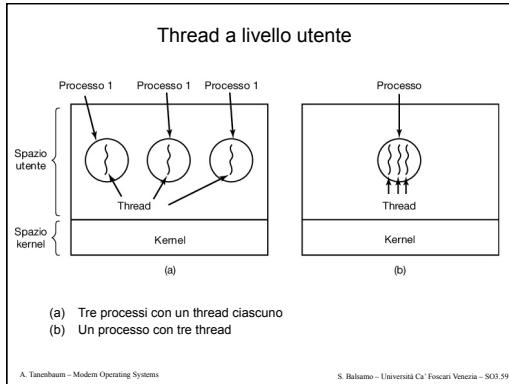


Operazioni sui Thread

- Threads e processi hanno delle operazioni in comune
 - Creazione
 - Exit (terminazione)
 - Sospensione
 - Recupero (resume)
 - Sleep
 - Risveglio

S. Balsamo – Università Ca' Foscari Venezia – SO4.52



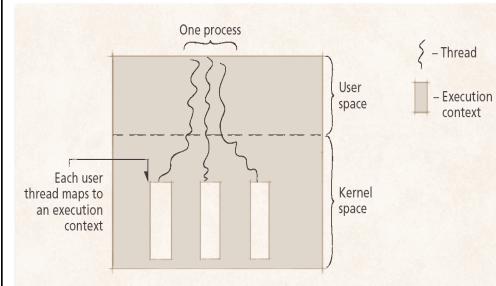


Thread nello spazio kernel

- I thread a livello nucleo cercano di superare i limiti del thread a livello utente mappando ogni thread al proprio **contesto di esecuzione**
 - I thread a livello di kernel forniscono un mapping di thread **uno-a-uno**
- Implementazione dei thread a livello utente
 - Vantaggi:** Aumento della **scalabilità, interattività, e throughput**
 - Svantaggi:** overhead dovuti al cambio di contesto e **ridotta portabilità** dovuto alle API specifici per S.O.
- I thread a livello nucleo non sono sempre la soluzione ottima per le applicazioni multithread

S. Balsamo - Università Ca' Foscari Venezia - SO4.63

Thread a livello kernel



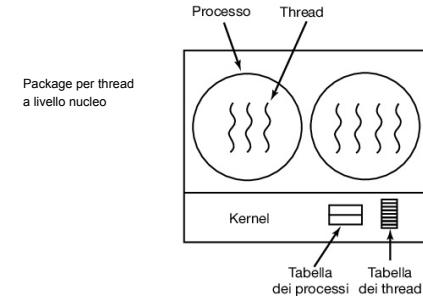
S. Balsamo - Università Ca' Foscari Venezia - SO4.64

Thread nello spazio kernel

- Non occorre una tabella di thread e un sistema di esecuzione real-time per in ogni processi
- Il nucleo ha la **tavella di thread** con i registri, stato e informazioni di tutti i thread
- Le operazioni sui thread avvengono con **chiamate** al nucleo costo maggiore del cambiamento di contesto possibile 'riciclo' di thread (strutture)
- Lo scheduler è a livello nucleo e può confrontare thread di processi diversi un thread bloccato può non bloccare tutto il processo
- Il nucleo ha anche la **tavella di processi**

S. Balsamo - Università Ca' Foscari Venezia - SO4.65

Thread a livello kernel



A. Tanenbaum - Modern Operating Systems

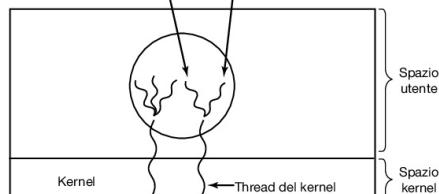
Threads combinati a livello utente e nucleo

- La combinazione di implementazione di thread di livello utente e livello nucleo
 - Mapping thread **molte-a-molti** (*m-to-n* thread mapping)
 - Numero di thread di livello utenti e livello kernel non deve essere uguale
 - Può ridurre l'overhead rispetto al mapping di thread uno-a-uno implementando il **thread pooling**
- Worker threads**
 - threads di livello nucleo **persistenti** che occupano il pool di thread
 - migliora le prestazioni in ambienti in cui i threads sono spesso creati e distrutti
 - ogni nuovo thread viene eseguito da un thread worker
- Attivazione dello **Scheduler**
 - Tecnica che permette alla libreria di livello utente di programmare i suoi thread
 - Si attua quando il S.O. **chiama una libreria di threading a livello utente** che determina se uno qualsiasi dei suoi thread devono essere rischedulati

S. Balsamo - Università Ca' Foscari Venezia - SO3.67

Thread ibridi a livello utente e nucleo

Molteplici thread utente su un thread del kernel



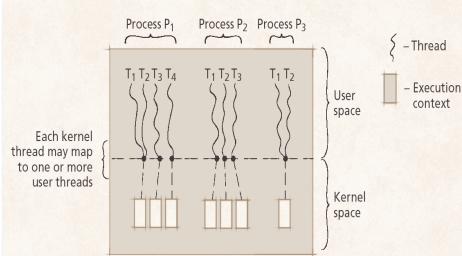
Package per thread a livello nucleo

A. Tanenbaum - Modern Operating Systems

S. Balsamo - Università Ca' Foscari Venezia - SO3.68

Thread ibridi a livello utente e nucleo

Modello ibrido di threading



S. Balsamo - Università Ca' Foscari Venezia - SO4.69

Attivazione dello Scheduler

Obiettivo

mimare le **funzionalità** dei threads di livello nucleo
migliorare le **prestazioni** dei thread del livello utente

Evitare transizioni non necessarie utente/kernel

Il nucleo assegna **processori virtuali** ad ogni processo

Permette al sistema di allocare a tempo di esecuzione thread ai processori (a livello utente)
Segnalazioni dal nucleo per possibili situazioni di thread bloccati (upcall) al sistema run-time che può attivare lo scheduler a livello utente

Si può usare su multiprocessori (con processori reali)

Problema

Uso intensivo del nucleo (basso livello)
che chiama procedure nello spazio utente (alto livello)

A. Tanenbaum - Modern Operating Systems

S. Balsamo - Università Ca' Foscari Venezia - SO3.70