**Processi e Thread**

Un processo è un programma in esecuzione e le risorse che utilizza e il contesto vengono usate

La memoria usata è detto spazio d’indirizzamento, è l’insieme delle celle di memoria usate dal processo.

Il contesto di esecuzione comprende:

* Il contesto hardware (program counter, stack pointer …)
* Spazio d’indirizzamento
* Sistema di controllo delle informazioni
* Credenziali

Il processo su linux è un oggetto, è una struct.

Programma è diverso da processo

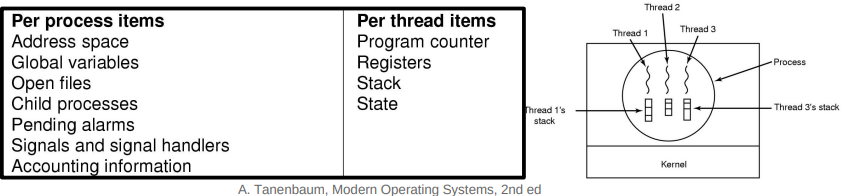
Il programma è il software su disco

Il processo è il programma in esecuzione

Si possono avere più istanze diverse dello stesso programma.

**Thread**

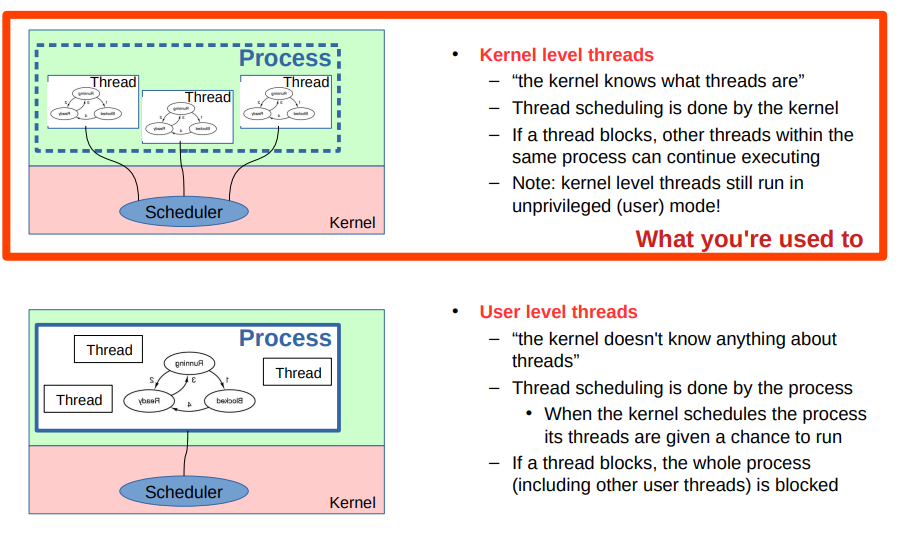
La differenza tra processo e thread è che il thread ha solo le informazioni che li servono.



Quando un processo ha più thread in esecuzione si chiama multi-threaded process, altrimenti è chiamato single-threaded process.

Le implementazioni thread possono essere:

* Kernel level
* User level

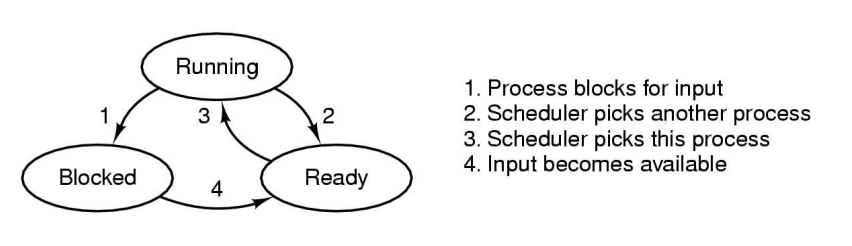


Solitamente non tutti i processi possono essere eseguiti allo stesso tempo, alcuni devono attendere

(esempio: risorsa occupata)

Lo stato del thread si compone di 3 stati:

* Running: in esecuzione
* Ready: pronto per essere schedulato
* Blocked: in pause, in attesa



Le implementazioni possono cambiare da sistema operativo a sistema operativo

**Creazione e distruzione dei processi**

Un processo può essere creato in 3 casi:

* Boot
* richiesta di un altro processo (system call)
* quando l’esecuzione termina (sistemi batch)

Un processo può terminare:

* Volontariamente
  + Fine del programma
  + Situazione di errore
* Involontariamente
  + Errore fatale(Segmentation fault ad esempio)
  + Quando viene terminato da un altro processo

In windows la chiamata di sistema è CreateProcess

In Linux si usa clone, a differenza di windows non richiede l’eseguibile

In Linux il processo restituito come un intero o long

**Identificazione dei processi**

I processi sono identificati da un PID, è un valore numerico.

Ciò consente di avere più istanze dello stesso programma in contemporanea.

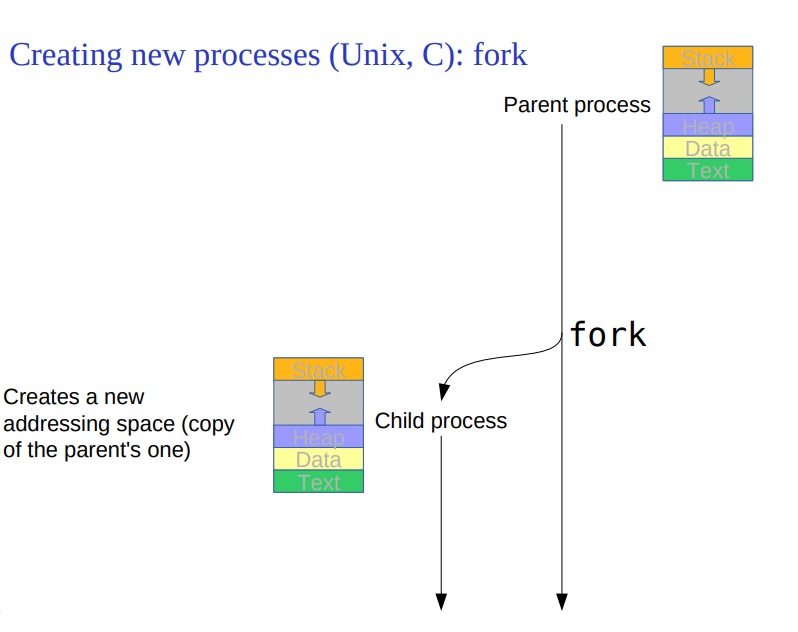
*Su Linux i processi sono allocati in /proc, è un filesystem virtuale, non risiede sul disco, in esso contiene le informazioni dei processi.*

Fork

Crea un nuovo processo partendo da 1

(Vedi dettagli su Example 01.c)

Clona il processo ed esegue le istruzioni dall’istruzione successiva all’istruzione di fork



Fork restituisce un pid. Se è il padre restituisce il pid del figlio, se il figlio restituisce 0

Le modifiche nei fork se modificate nel fork non si vedono le modifiche negli altri

Exec

Consente di eseguire altri processi

Execl primitiva più semplice

Parametri: pathname, args

Sostiuisce il codice del processo che lo chiama con il codice del processo chiamato,

Non richiama un nuovo processo.

Le risorse (file, ecc...) sono ereditati dai figli, sono disponibili anche ai figli.

**Attendere un processo**

Wait() e waitpid()

Wait attende un figlio qualsiasi, waitpid attende un figlio specifico.

La chiamata wait è bloccante

Se il valore di uscita è 0 allora ha terminato correttamente altrimenti no, solitamente.

Cosa succede se il padre non fa il wait?

Si ha uno stato zombie, **da evitare**

Ci sono 3 possibili soluzioni:

1. Il padre fa waitpid
2. Il padre termina
3. C'è il processo init che ha il compito di recuperare i processi zombie e fa un wait per terminarli.