

## STATO DEL PROCESSO ➤ 3.1.2

Il processo può avere diversi stati durante l'esecuzione:

- New: Il processo è stato appena creato.
- Ready: Il processo è pronto ad essere eseguito.
- Running: È in esecuzione.
- Waiting: È in attesa di un qualche evento.
- Terminated: Ha finito l'esecuzione.

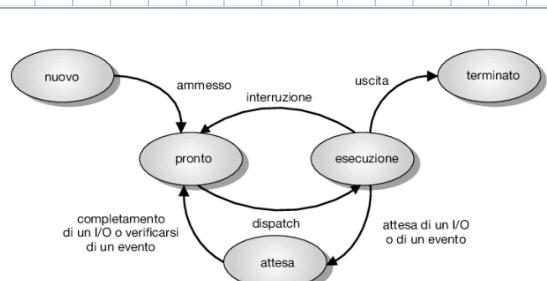


Figura 3.2 Diagramma di transizione degli stati di un processo.

## PROCESS CONTROL BLOCK ➤ 3.1.3

È un blocco di informazioni riguardanti il processo.

Ogni processo ha un PCB. È formato da:

- Stato del processo: New, Ready, Running, ...
- Program Counter:  
Ha una copia del PC che viene salvata SOLO in alcuni momenti.  
Tenerlo sempre sincronizzato avrebbe grossi costi.
- Process Number: PID (Process ID)
- CPU Registers:  
Tiene salvati copie di registri di uso comune, ovvero quelli modificabili con load e store.  
(Quelli di uso non comune sono tanti e non avrebbe senso salvarli)
- Informazioni su scheduling CPU:  
Ovvero la priorità del processo. L'S.O. fa in modo che l'utente non possa "passare davanti" al S.O.

E puntatori alle code di scheduling.

- Memory management information.

Info sullo spazio di indirizzamento del processo.

Soltamente includono il max e il min.

- Accounting information

Info di log relativo a % di utilizzo CPU, durata del processo, limiti di tempo e così via.

- I/O status information

La lista di I/O eseguiti, file aperti ecc.

Ogni I/O ha un file descriptor, che specifica come viene gestito dai drivers.

## CPU CONTEXT SWITCH > 3.2.3

È il processo con cui la CPU passa da un processo all'altro. Il processo viene descritto così:

### 1. Esecuzione Processo Corrente

La CPU sta eseguendo le istruzioni del processo  $P_0$  caricato in memoria.

### 2. Interruzione del processo

Un evento come il Timer consente il cambio.

Il processo stesso entrerà in modalità Kernel e salverà le informazioni in corso nel PCB.

### 3. Cambio di programma

Sarà sempre il Kernel a caricare il PCB del programma entrante e proseguire.

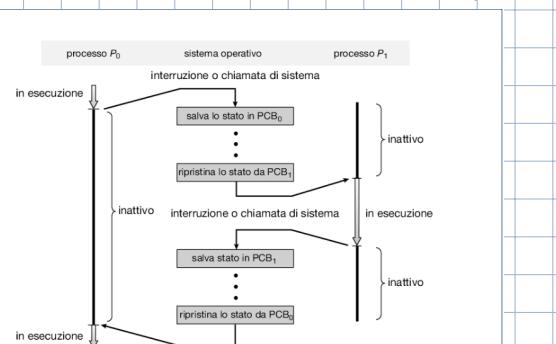


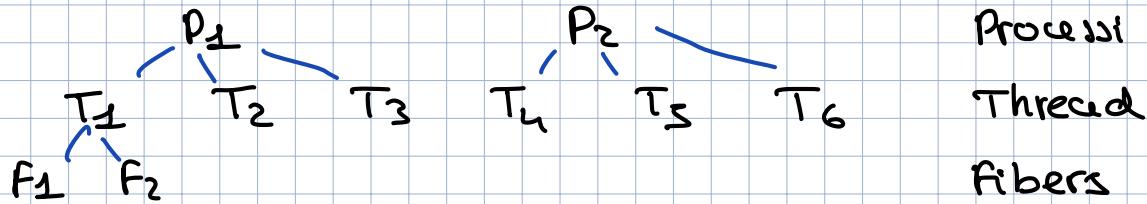
Figura 3.6 Diagramma di cambio di contesto.

## THREAD > 3.1.4

Ogni processo può avere da 1 a N thread.

I thread consentono di avere più percorsi di esecuzione.

A loro volta, ogni thread può avere delle fibers che sono dei "mini thread" collaborativi



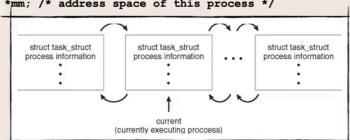
Ogni thread dello stesso processo condividono la memoria.

Le informazioni sui thread vengono salvate nel PCB.

### Process Representation in Linux

Represented by the C structure task\_struct

```
pid_t pid; /* process identifier */  
long state; /* state of the process */  
unsigned int time_slice; /* scheduling information */  
struct task_struct *parent; /* this process's parent */  
struct list_head children; /* this process's children */  
struct files_struct *files; /* list of open files */  
struct mm_struct *mm; /* address space of this process */
```



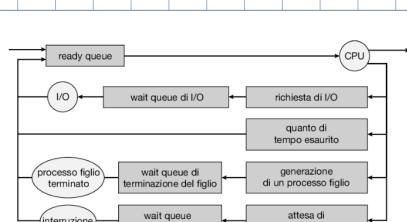
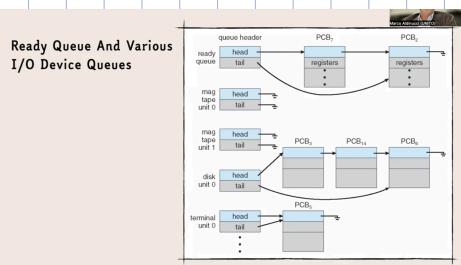
N.B.: Ogni PCB ha dei puntatori per cui il context switch si fa spostando i puntatori.

## CODE DI SCHEDULING > 3.2

de PCB

Sono code per stabilire l'ordine di esecuzione dei vari processi nella multi-programmazione.

Solitamente vengono definite più code per ogni I/O per facilità.



Sono anche definite in base ad alcuni degli stati.

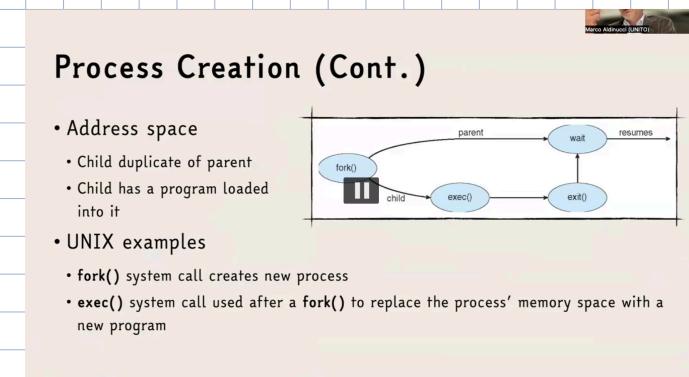
# SCHEDULERS ➤ ?

- Short Term: Schedula il prossimo processo da eseguire
  - Long Term : Seleziona il prossimo processo da portare in ready per la multi programmazione.  
Un po' più lento.

CREAZIONE DI UN PROCESSO > 3.3.1

Dopo il boot, viene creato il processo padre (systemd Linux) che a sua volta avrà dei figli creando una struttura ad albero.

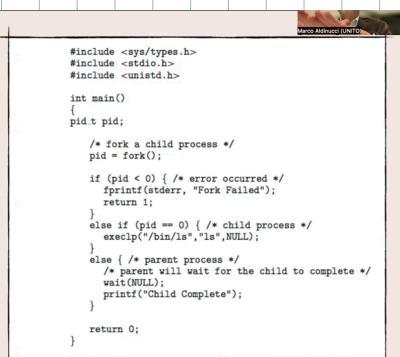
Ogni processo ha un PID (Process Identifier) e un collegamento col padre. [vedere il Lab per i comandi]



# So Linux

Il processo forkato condividerà tutto l'insieme PID e Area di mem.

Il processo padre può aspettare (wait) che il p. figlio termini (exit).



Una volta invocata la fork  
padre e figlio eseguiranno  
il codice seguente.



Orfani (Parent Now ha invocato wait)  
Quando un processo padre muore e il figlio è ancora in exec.  
Viene adottato dal system.

Zombie (Parent ha invocato wait)  
Se il parent non ha fatto  
la wait.

## EXIT E ABORT

Se il processo termina per `abort()`; allora c'è stato un errore non previsto.

Mentre, se termina per `exit()`; se c'è un errore, allora era prevista

## CASCADE TERMINATION

È una policy di alcuni S.O. per i quali la terminazione di un processo prevede la terminazione dei figli ecc.