

# Processi in Unix release 4

---

## Index

- [Introduction](#)
  - [Transizioni tra stati dei processi](#)
  - [Processo Unix](#)
    - [Livello utente](#)
    - [Livello registro](#)
    - [Livello sistema](#)
  - [Process Table Entry](#)
  - [U-Area](#)
  - [Creazione di un processo in Unix](#)
- 

## Introduction

Utilizza la seconda opzione in cui la maggior parte del SO viene eseguito all'interno dei processi utente in modalità kernel

---

## Transizioni tra stati dei processi



Dal kernel running si può passare a preempted, ovvero quel momento prima che finisca il processo in cui il kernel per qualche motivo decide di toglierli il processore. Quando un processo finisce, prima che muoia, va nello stato zombie, in cui tutta la memoria di quel processo viene deallocata (compresa l'immagine) e l'unica cosa che sopravvive è il process control block con l'unico scopo di comunicare l'exit status al padre; una volta che il padre ha ricevuto che il figlio gli ha dato questo exit status, a quel punto anche il PCB viene tolto e il processo figlio viene definitivamente terminato. Da notare che un processo in kernel mode non è interrompibile che non lo rendeva adatto ai processi real-time.

In sintesi

**User running** → in esecuzione in modalità utente; per passare in questo stato bisogna necessariamente passare per kernel running in quanto è avvenuto un process switch, l'unica cosa che può avvenire è tornare in kernel running in seguito ad una system call o interrupt.

**Kernel running** → in esecuzione in modalità kernel o sistema.

**Ready to Run, in Memory** → può andare in esecuzione non appena il kernel lo seleziona.

**Asleep in Memory** → non può essere eseguito finché un qualche evento non si manifesta e ci è diventato a seguito di un evento bloccante; il processo è in memoria, corrisponde al blocked del modello a 7 stati.

**Ready to Run, Swapped** → può andare in esecuzione (non è in attesa di eventi).

esterni), ma prima dovrà essere portato in memoria

**Sleeping, Swapped** → non può essere eseguito finché un qualche evento non si manifesta; il processo non è in memoria primaria

**Preempted** → il kernel ha appena tolto l'uso del processore a questo processo (*preemption*), per fare un context switch

**Created** → appena creato, ma non ancora pronto all'esecuzione

**Zombie** → terminato tutta la memoria del processo viene deallocata (compresa l'immagine) e l'unica cosa che sopravvive è il process control block con l'unico scopo di comunicare l'exit status al padre; una volta che il padre lo ha ricevuto, anche il PCB viene tolto e il processo figlio viene definitivamente terminato

---

## Processo Unix

Un processo in unix è diviso in:

- livello utente
- livello registro
- livello di sistema

### Livello utente

**Process text** → il codice sorgente (in linguaggio macchina) del processo

**Process data** → sezione di dati del processo; compresi anche i valori delle variabili

**User stack** → stack delle chiamate del processo; in fondo contiene anche gli argomenti con cui il processo è stato invocato

**Shared memory** → memoria condivisa con altri processi, usata per le comunicazioni tra processi

### Livello registro

**Program counter** → indirizzo della prossima istruzione del process text da eseguire

**Process status register** → registro di stato del processore, relativo a quando è stato swappato l'ultima volta

**Stack pointer** → puntatore alla cima dello user stack

**General purpose registers** → contenuto dei registri accessibili al programmatore, relativo a quando è stato swappato l'ultima volta

### Livello sistema

**Process table entry** → puntatore alla tabella di tutti i processi, dove individua quello corrente

**U area** → informazioni per il controllo del processo

**Per process region table** → definisce il mapping tra indirizzi virtuali ed indirizzi fisici (page table)

**Kernel stack** → stack delle chiamate, separato da quello utente, usato per le funzioni da eseguire in modalità sistema

---

## Process Table Entry

Process status	Current state of process.
Pointers	To U area and process memory area (text, data, stack).
Process size	Enables the operating system to know how much space to allocate the process.
User identifiers	The <b>real user ID</b> identifies the user who is responsible for the running process. The <b>effective user ID</b> may be used by a process to gain temporary privileges associated with a particular program; while that program is being executed as part of the process, the process operates with the effective user ID.
Process identifiers	ID of this process; ID of parent process. These are set up when the process enters the Created state during the fork system call.
Event descriptor	Valid when a process is in a sleeping state; when the event occurs, the process is transferred to a ready-to-run state.
Priority	Used for process scheduling.
Signal	Enumerates signals sent to a process but not yet handled.
Timers	Include process execution time, kernel resource utilization, and user-set timer used to send alarm signal to a process.
P_link	Pointer to the next link in the ready queue (valid if process is ready to execute).
Memory status	Indicates whether process image is in main memory or swapped out. If it is in memory, this field also indicates whether it may be swapped out or is temporarily locked into main memory.

---

## U-Area

Process table pointer	Indicates entry that corresponds to the U area.
User identifiers	Real and effective user IDs. Used to determine user privileges.
Timers	Record time that the process (and its descendants) spent executing in user mode and in kernel mode.
Signal-handler array	For each type of signal defined in the system, indicates how the process will react to receipt of that signal (exit, ignore, execute specified user function).
Control terminal	Indicates login terminal for this process, if one exists.
Error field	Records errors encountered during a system call.
Return value	Contains the result of system calls.
I/O parameters	Describe the amount of data to transfer, the address of the source (or target) data array in user space, and file offsets for I/O.
File parameters	Current directory and current root describe the file system environment of the process.
User file descriptor table	Records the files the process has open.
Limit fields	Restrict the size of the process and the size of a file it can write.
Permission modes fields	Mask mode settings on files the process creates.

## Creazione di un processo in Unix

La creazione di un processo unix tramite una chiamata di sistema `fork()`. In seguito a ciò, in Kernel Mode:

1. Alloca una entry nella tabella dei processi per il nuovo processo (figlio)
2. Assegna un PID unico al processo figlio
3. Copia l'immagine del padre, escludendo dalla copia la memoria condivisa (se presente)
4. Incrementa i contatori di ogni file aperto dal padre, per tenere conto del fatto che ora sono anche del figlio
5. Assegna al processo figlio lo stato Ready to Run
6. Fa ritornare alla fork il PID del figlio al padre, e 0 al figlio

Quindi, il kernel può scegliere tra:

- continuare ad eseguire il padre
- switchare al figlio
- switchare ad un altro processo

Creare un processo a partire dal processo padre è il modo più efficiente di avviare un processo in quanto la maggior parte delle volte un programma inizia un processo a partire dal codice sorgente già esistente