Il Sistema Operativo ed i Processi

Giuseppe Pozzi

Impianti di Elaborazione
Facoltà di Ingegneria di Como
Politecnico di Milano

giuseppe.pozzi@polimi.it
- versione del 22 ottobre 2003 -

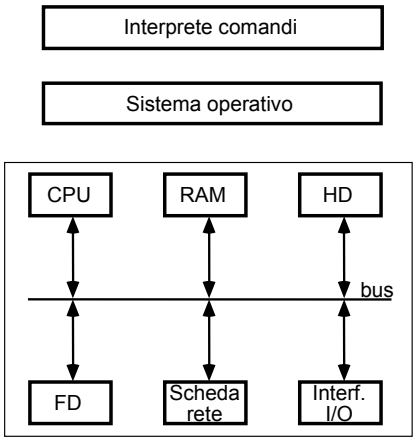
Sistema operativo e processi

- **Bibliografia**:
  Pelagatti G.
  "**Sistemi di Elaborazione** – Architetture hardware e software",McGraw-Hill, 1992.
  Per approfondimenti:
  Tanenbaum A. S., Woodhull A. W., Operating Systems - Design and Implementation, 2nd ed. Prentice Hall, 1997.
- **Sommario**:
  - Architettura di un sistema operativo.
  - Processi e loro realizzazione.
  - Inizializzazione del sistema operativo.

Il sistema operativo

- Il sistema operativo:
  - è un insieme di moduli software;
  - controlla le risorse del sistema;
  - interagisce direttamente con i software che controllano l'hardware (*device driver*);
  - mette a disposizione dell'utente una macchina "virtuale", in grado di eseguire comandi dati dall'utente, utilizzando una macchina "reale", di livello inferiore e meno potente.

Il sistema operativo



Il processo dal punto di vista del sistema operativo

- Il processo è:
  - un'attività sequenziale (come ad es. l'istanza di un programma in esecuzione[1]),
  - con uno stato associato. Lo stato a sua volta si divide in:
    - stato interno;
    - stato esterno.

[1] il processo è anche definito come l'esecutore di un programma, creato dinamicamente.

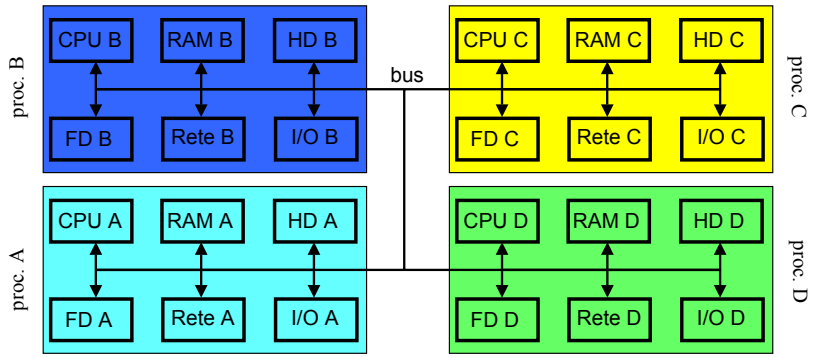
Lo stato di un processo

- Lo stato interno indica:
  - la prossima istruzione del programma che deve essere eseguita;
  - i valori delle variabili e dei registri utilizzati dal processo.
- Lo stato esterno indica se il processo è:
  - in attesa di un evento, come ad es. la lettura da disco o l'inserimento di dati da tastiera;
  - in esecuzione;
  - pronto per l'esecuzione, e quindi attende di accedere all'utilizzo della CPU.

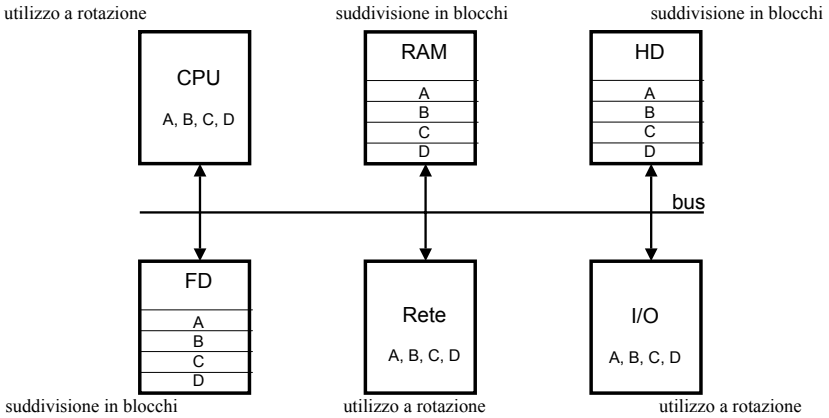
Il sistema operativo e le macchine virtuali

- Il sistema operativo:
  - controlla la macchina multiprogrammata, cioè che esegue più processi contemporaneamente, rendendo visibile ad ogni processo una macchina "virtuale" ad esso interamente dedicata e quindi con risorse proprie.

Il sistema operativo e le macchine virtuali



Il sistema operativo e la macchina reale



Il sistema operativo

- Il sistema operativo:
  - gestisce eventuali conflitti di accesso contemporaneo alle risorse "reali" condivise, accodando i processi richiedenti.

    Ad es. un componente, detto *scheduler*, seleziona i processi in stato di pronto ed assegna loro a rotazione l'utilizzo della CPU per un quanto di tempo di esecuzione (politica detta *round robin*).

I processi ed il sistema operativo

- Anche il sistema operativo è implementato tramite processi;
- il sistema operativo è garante che i conflitti tra i processi siano controllati e gestiti correttamente;
- il sistema operativo viene eseguito in modalità privilegiata (*kernel mode* o *supervisor*), così da poter controllare gli altri processi eseguiti in modalità *user*.

Chiamate al *supervisor*

- I processi utente per eseguire operazioni privilegiate, come ad es.,
  - accesso a file;
  - accesso ad altre risorse;
  - operazioni di I/O diretto

invocano il *supervisor* tramite chiamate di sistema (*system calls*).

Motivazioni dell'uso del *supervisor*

- Le operazioni di I/O sono operazioni riservate:
  - un processo *A* non deve poter andare a scrivere messaggi su un terminale non associato allo stesso processo *A*;
  - un processo *A* non deve poter leggere caratteri immessi da un terminale non associato allo stesso processo *A*.

Motivazioni dell'uso del *supervisor*

- Un processo non deve poter sconfinare al di fuori del proprio spazio di memoria:
  - per non accedere allo spazio di memoria associato ad un altro processo, modificando codice e dati di quest'ultimo;
  - per non occupare tutta la memoria disponibile nel sistema, bloccandolo e rendendolo così inutilizzabile da altri processi.

Motivazioni dell'uso del *supervisor*

- La condivisione di risorse, quali ad esempio dischi, CPU, schede di rete ...
  - deve essere tale da cautelare i dati di ogni utente;
  - deve evitare che un utente occupi l'intero spazio disponibile sui dischi, non lasciando spazio per altri utenti;
  - se un processo *A* entra in loop, ciò non deve bloccare l'intero sistema;
  - … ed altri esempi ancora.

Motivazioni dell'uso del *supervisor*

- Essendo un sistema multiprogrammato e/o multiutente:
  - eventuali conflitti devono essere gestiti da un "arbitro" (il sistema operativo) che funzioni secondo regole chiare e ben stabilite.

Necessità di parallelismo nei sistemi operativi

- Un utente che si collega ad una macchina (ad es. Unix tramite il programma *telnet*) dispone di un interprete comandi, eventualmente scelto dall'utente stesso tramite *chsh*.
- Nella macchina deve essere presente un processo interprete comandi per ogni sessione.

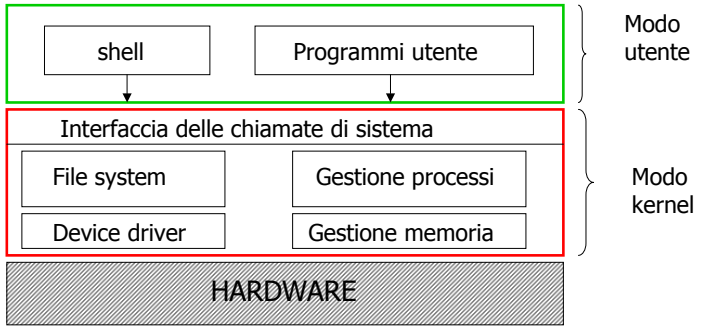
Necessità di parallelismo nei sistemi operativi

- Ad ogni nuova connessione il sistema operativo genera un processo interprete comandi specifico per quella connessione.
- Ci sono quindi più copie dello stesso programma.

Parallelismo in Unix

- Esistono due principali metodi per ottenere l'esecuzione parallela in ambiente Unix:
  - generazione di nuovi processi;
  - utilizzo di thread.
- Considereremo la generazione di nuovi processi (meccanismo più standard e meno complesso).

Architettura del sistema operativo



Struttura a strati di UNIX

Architettura del sistema operativo

- I programmi utente e la shell interagiscono con il sistema operativo invocando un insieme ben definito di chiamate di sistema (system calls).
- Le chiamate di sistema istruiscono il kernel a eseguire operazioni per il programma chiamante e a scambiare dati tra kernel e programmi utente.

Processo

- Rappresenta una istanza del programma in esecuzione con tutte le informazioni necessarie:
  - codice eseguibile;
  - dati del programma;
  - spazio di indirizzamento virtuale che contiene il codice eseguibile, i dati e lo stack;
  - informazioni relative al suo funzionamento (stato).

Esecuzione di un processo

- L'esecuzione del processo avviene all'interno dello spazio di indirizzamento assegnato e non può saltare nello spazio di indirizzamento di altri processi.
- I processi possono comunicare con l'esterno attraverso le chiamate di sistema.
- Un processo in UNIX è una entità creata da una chiamata di sistema *fork()*.

Esecuzione di un processo

- Ogni processo, eccetto il processo 0, è creato da un altro processo mediante una fork.
- Il processo che invoca la fork è il processo padre.
- Il processo creato è un processo figlio.
- Il sistema operativo associa ad ogni processo un identificatore PID (process ID).

Creazione di processi

- Processo 0: processo speciale creato nella fase di bootstrap del sistema operativo.
- Dopo aver creato il processo figlio 1, il processo 0 diventa il processo swapper.
- Processo 1, init, è l'antenato di ogni altro processo del sistema operativo ed è responsabile dell'inizializzazione dei nuovi processi.

Modalità di esecuzione dei processi

- Un processo può essere eseguito in due modalità:
  - kernel
  - user
- Ad ogni processo sono associati due stack:
  - uno stack user che contiene le informazioni memorizzate durante l'esecuzione in modalità user (es. chiamate a funzione);
  - uno stack kernel che contiene le informazioni memorizzate durante l'esecuzione in modalità kernel (es. chiamate a funzione).
- Sono quindi necessari due Stack Pointer.

Ipotesi sull'architettura hardware

- Processi in modalità user possono accedere al proprio codice e dati ma non ai dati e al codice del sistema operativo.
- I processi in modalità kernel possono accedere sia allo spazio di indirizzamento del sistema operativo sia a quello utente.
- Alcune istruzioni macchina sono privilegiate: se vengono eseguite in modalità user generano un errore (es. istruzioni che manipolano il Processor Status Word).

Modalità di esecuzione di un processo

- Il passaggio dalla modalità di esecuzione user a quella kernel avviene ad ogni chiamata di sistema.
- Una chiamata di sistema attiva una "trap" che causa il cambiamento di stato del processo da modalità utente a modalità kernel.
- Questa informazione è generalmente contenuta in 1 bit del registro PSW.

Tabella dei processi

- SO dispone di una tabella dei processi che contiene una riga per ogni processo.
- Informazioni memorizzate per ogni processo:
  - informazioni relative alla gestione dei processi;
  - informazioni relative alla gestione della memoria;
  - informazioni relative alla gestione dei file.

U-area

- Ad ogni processo è allocata una zona di memoria chiamata u-area che contiene informazioni relative al processo che possono essere manipolate solo dal sistema operativo.
- U-area: estensione delle informazioni contenute nella tabella dei processi.
- La u-area contiene informazioni relative al processo che devono essere accessibili quando il processo è in esecuzione.

Informazioni della u-area

- Le principali informazioni contenute nella u-area sono:
  - puntatore all'elemento della tabella dei processi che identifica il processo corrente in esecuzione;
  - i parametri della chiamata di sistema corrente, i valori di ritorno e i codici di errore;
  - la tabella dei descrittori di file aperti dal processo;
  - parametri interni di I/O;
  - direttorio corrente;
  - dimensioni massime del processo e dei file.

Utilizzo della u-area

- Il sistema operativo può accedere ai campi della u-area del processo in esecuzione ma non degli altri processi.
- La variabile *u* del sistema operativo identifica l'indirizzo della u-area del processo in esecuzione che permette di identificare anche l'elemento della tabella dei processi.
- Quando viene mandato in esecuzione un nuovo processo, la variabile u conterrà l'indirizzo della u-area del nuovo processo.

Contesto di un processo

- Contesto: insieme delle informazioni che caratterizzano lo stato di un processo:
  - se il processo è fisicamente in esecuzione parte del contesto si trova nei registri della CPU (Program Counter, Stack Pointer, PSW, registri utente);
  - se il processo non è in esecuzione il contesto è in memoria.

Contesto di un processo

- Corrisponde allo stato del processo ed è caratterizzato dalle seguenti informazioni:
  - codice;
  - valori delle variabili globali e strutture dati a livello user;
  - i valori contenuti nei registri del processore;
  - le informazioni memorizzate nella tabella dei processi e nella u-area;
  - il contenuto dello stack user e dello stack kernel.

Cambiamento di contesto

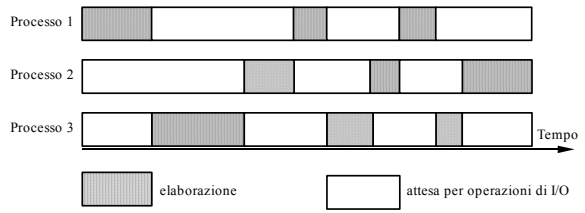
- Operazione che avviene quando il sistema operativo decide di mandare in esecuzione un altro processo.
- Il sistema operativo deve salvare tutte le informazioni necessarie a ripristinare esattamente lo stato del processo in esecuzione nel futuro.
- Il sistema operativo gestisce gli interrupt nel contesto del processo in esecuzione in quel momento, anche se non è il processo che ha causato o attende questo interrupt.

Sistema operativo multiprogrammato

Multiprogrammato: in memoria vi sono più processi.

La CPU può passare dall'esecuzione di un processo ad un altro quando il processo in esecuzione non può proseguire per l'attesa di una risorsa o di un evento.

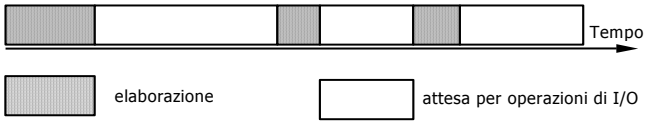
Multiprogrammazione: esempio



Sistema operativo in time sharing

- Permette la condivisione della CPU tra più processi interattivi.
- Il tempo di esecuzione del processore è condiviso tra più utenti.
- Ogni processo in esecuzione ha a disposizione un quanto di tempo di utilizzo della CPU, al termine del quale viene sospeso per lasciare il posto ad un altro processo in attesa di esecuzione.

Esecuzione di un processo

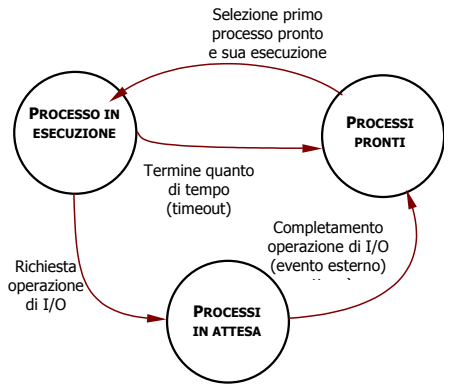


- Un processo utente può effettivamente essere in esecuzione sulla CPU.
- Ogni operazione di I/O consiste in una chiamata al sistema operativo e quindi in una sospensione del processo utente per l'esecuzione dell'operazione di I/O da parte del kernel.

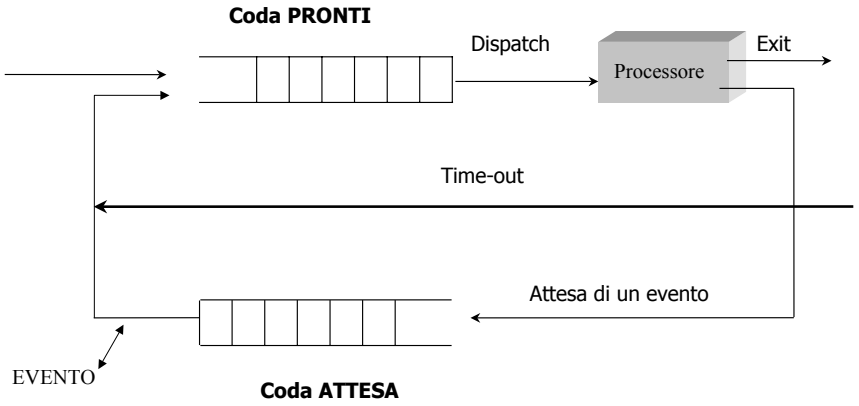
Processi non in esecuzione

- Processi in attesa di un evento esterno (ad esempio I/O).
- Processi pronti ad essere eseguiti in attesa della CPU.
- Si tratta di due stati diversi: PRONTO e ATTESA realizzati con due code diverse.

Diagramma a tre stati



Realizzazione del modello a tre stati



Transizioni di stato

ESECUZIONE ➔ PRONTO

- Al termine del quanto di tempo il SO deve salvare tutte le informazioni necessarie per poter riprendere l'esecuzione del processo dal punto in cui è stata interrotta.
- Informazione salvata in memoria nella tabella dei processi.

Transizioni di stato

ESECUZIONE ➔ ATTESA

- Si verifica quando il processo richiede delle risorse che non sono disponibili o attende un evento.
- il SO salva tutte le informazioni necessarie a riprendere l'esecuzione e l'informazione relativa all'evento atteso nella tabella dei processi.

Transizioni di stato

ATTESA ➔ PRONTO

- Quando l'evento atteso da un processo si verifica, il SO sposta tutti i processi in attesa di quell'evento o di quella risorsa nella coda dei processi pronti.

Transizioni di stato

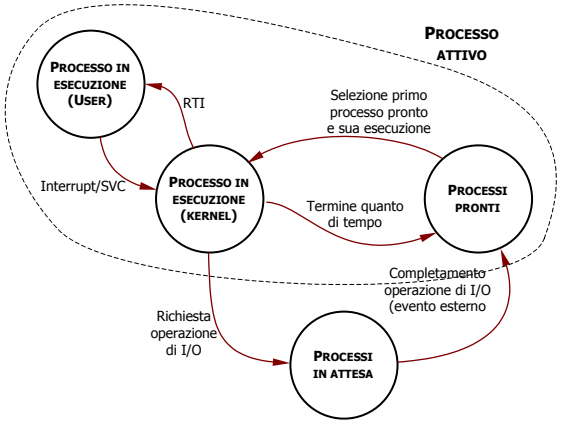
PRONTO ➔ ESECUZIONE

- Il SO stabilisce quale dei processi accodati nello stato di PRONTO debba essere mandato in esecuzione.
- La scelta è effettuata dall'algoritmo di scheduling che deve bilanciare efficienza e fairness.

Modalità user e modalità kernel

- I processi possono essere eseguiti in modalità kernel o user
  - necessario sdoppiare lo stato di esecuzione in due stati.
- Transizione di stato da esecuzione in modalità user a modalità kernel avviene in due casi:
  - chiamata di sistema (SVC);
  - interrupt.
- Transizione di stato da esecuzione in modalità kernel a modalità user avviene quando:
  - termina l'esecuzione della routine di risposta all'interrupt.

Transizioni di stato



Esempio

- Esecuzione di due processi P1 e P2:
  1. P1 in esecuzione(modalità user): richiesta di un servizio di I/O (chiamata di sistema – SVC), per esempio una lettura di un carattere da tastiera;
  2. Sistema Operativo in esecuzione in modalità kernel nel contesto di P1 decide di sospendere P1 finché non viene completata l'operazione di I/O richiesta. P1 viene posto in stato di Attesa sull'evento lettura da tastiera;
  3. Sistema Operativo sceglie quale tra i processi pronti viene mandato in esecuzione: sceglie P2;
  4. Cambiamento di contesto e inizio esecuzione di P2 in modalità user.

Esempio

5. Si verifica un interrupt da tastiera: l'esecuzione della risposta ad interrupt avviene in modalità kernel nel contesto di P2 che era in esecuzione. Il carattere viene copiato in un buffer di sistema e viene risvegliato il processo P1 in attesa dell'evento;
6. P1 passa dalla coda dei processi in attesa alla coda dei processi pronti;
7. La terminazione della risposta all'interrupt (IRET) riporta P2 in esecuzione in modalità user.

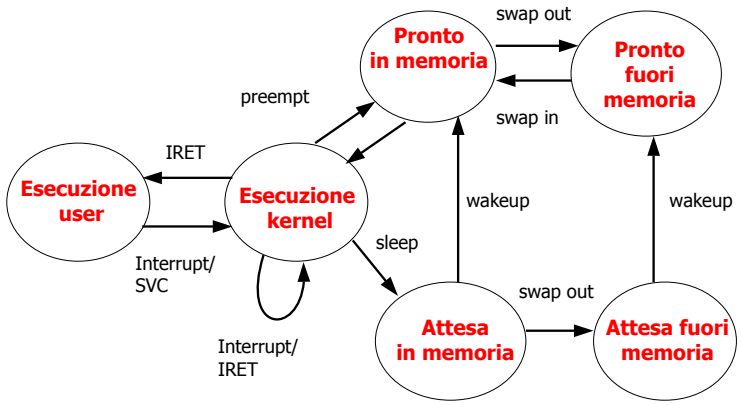
Stati di un processo

- Nel caso di memoria virtuale i processi possono trovarsi in memoria o fuori memoria (su disco).
- Gli stati di Attesa e Pronto si sdoppiano:
  - attesa in memoria e Attesa fuori memoria
  - pronto in memoria e Pronto fuori memoria
- Il processo di spostare un processo fuori o in memoria centrale si chiama swapping.
- Il sistema operativo sposta un processo fuori memoria se necessita di spazio in memoria.

Transizioni fuori memoria

- Lo spazio di memoria può essere richiesto nei seguenti casi:
  - l'esecuzione di una fork richiede l'allocazione di spazio di memoria per il processo figlio;
  - una richiesta di memoria dell'area dati (mediante *brk*) incrementa le dimensioni del processo;
  - crescita del segmento stack implica la crescita delle dimensioni globali del processo;
  - il Sistema Operativo vuole liberare spazio in memoria per i processi che ha portato fuori memoria precedentemente e vuole ora riportare in memoria i processi pronti.

Stati di un processo



Procedure di transizione di stato

- Le transizioni da uno stato all'altro richiedono il cambiamento del contesto del processo.
- IL SO salva il contesto del processo in esecuzione quando:
  - deve gestire una interruzione;
  - viene eseguita una chiamata di sistema;
  - viene eseguita una preempt;
  - viene eseguita una sleep.

Sleep

- Un processo esegue una SLEEP tipicamente durante una chiamata di sistema:
  - il processo da modo user passa a modo kernel e viene posto in attesa della risorsa richiesta;
  - la sleep richiede l'esecuzione di un cambiamento di contesto, in cui tutte le informazioni relative al processo vengono poste sullo stack;
  - la sleep prende come informazione l'evento sul quale il processo è in attesa;
  - gli eventi sono mappati su un insieme di indirizzi virtuali del sistema operativo.

Wakeup

- Quando si verifica un evento, viene eseguita una wakeup per tutti i processi in attesa su quell'evento e passano tutti in stato di PRONTO.
- Viene eseguita durante l'esecuzione di una chiamata di sistema o durante l'esecuzione della risposta alle interruzioni.

Creazione di un processo (fork)

- Realizzazione della *fork* da parte del kernel:
  1. verifica se esistono sufficienti risorse per completare la fork con successo (es. Verifica se esiste spazio sufficiente in memoria o su disco ed eventualmente deve allocare delle pagine di memoria);
  2. alloca una riga nella tabella dei processi per il nuovo processo figlio;
  3. assegna un PID univoco al processo figlio;
  4. esegue una copia del contesto del processo padre (u-area, codice, dati, stack);
  5. incrementa di 1 i contatori nelle tabelle dei file degli i-node per i file associati con il processo padre e ora anche con il figlio;
  6. pone lo stato del processo a PRONTO;
  7. ritorna il PID al processo padre e 0 al processo figlio.

Terminazione di un processo

- Avviene con l'esecuzione della chiamata di sistema *exit (status)*.
- Realizzazione da parte del kernel:
  - chiude tutti i file aperti (close) e rilascia gli i-node associati al direttorio corrente;
  - libera la memoria dello spazio utente;
  - scrive informazioni relative a statistiche run-time riguardanti l'esecuzione del processo in un file globale;
  - in Unix il processo passa nello stato *zombie*;
  - sgancia il processo dall'albero dei processi facendo in modo che il processo 1 adotti tutti i suoi processi figli (se esistono);
  - esegue un cambiamento di contesto in modo da poter mandare in esecuzione un altro processo pronto.

Attesa della terminazione di un processo

- Un processo può sincronizzare la propria esecuzione con la terminazione di un processo figlio mediante l'esecuzione della chiamata di sistema wait.
- L'esecuzione della wait comporta la ricerca da parte del kernel di un figlio zombie del processo, e se il processo non ha figli, ritorna un errore.
- Se identifica un figlio in stato di zombie, ne estrae il PID e il parametro fornito dalla exit del processo figlio e ritorna questi valori.
- Il kernel libera la riga della tabella dei processi occupata dal processo figlio zombie.

Esecuzione di wait

- Se il processo che esegue la wait ha processi figli ma nessuno è zombie, il processo esegue una sleep e verrà risvegliato al momento della terminazione di un suo processo figlio.

  Al risveglio il processo inizia nuovamente l'esecuzione della wait e trova un processo figlio zombie.

Inizializzazione del sistema operativo

- Obiettivo della fase di inizializzazione è caricare una copia del sistema operativo in memoria e iniziarne l'esecuzione.
- La procedura di bootstrap sulle macchine UNIX legge il blocco di bootstrap (blocco 0) da un disco e lo carica in memoria.
- Il programma contenuto nel blocco di bootstrap carica il kernel dal file system e poi trasferisce il controllo all'indirizzo di inizio del kernel che va in esecuzione.

Inizializzazione del sistema operativo

- Il kernel inizializza le strutture dati interne
  - Costruisce le liste di i-node e buffer, inizializza la tabella delle pagine,....
- *Mount* del file system di root e crea il contesto per il processo 0, creando la u-area, inizializzando la riga 0 della tabella dei processi.
- Quando il contesto del processo 0 è pronto il sistema è in esecuzione come processo 0.
- Il processo 0 esegue una *fork* e crea il processo 1.

Inizializzazione del sistema operativo

- Il processo 1 viene creato artificialmente.
- Il processo 1, in esecuzione in modo kernel, crea il suo spazio di indirizzamento utente e copia il codice da eseguire dallo spazio di indirizzamento kernel a quello utente.
- Il processo 1 esegue il codice copiato in modo utente.
- Questo codice consiste una chiamata di sistema *exec* per mandare in esecuzione il programma "*/etc/init*" che è responsabile dell'inizializzazione dei nuovi processi.
- Il processo 1 è anche chiamato *init*.

Inizializzazione del sistema operativo

- Il processo *init* legge il file */etc/inittab* che indica quali processi creare.
- In un sistema multiutente *inittab* richiede la creazione di tanti processi *getty* quanti sono i terminali disponibili.
- Il processo 0 crea i processi kernel come per esempio il processo *swapper* per la gestione della memoria virtuale.