# Sistemi Operativi

Leonardo Mengozzi

Titoletti indice link a rispettive sezioni, in alto a sinistra "←Indice" link a pagina Indice.

## Contents



# 1 Batch

| V.Ambiente | Descrizione |
|---|---|
| PATH | Modificabile, sequenza di percorsi assoluti, divisi da ":", di directory contenenti eseguibili (lanciabili senza digitare path). Ricerca secondo ordine specificato in PATH, si ferma a primo eseguibile con nome uguale. Eventuale ErrorNotFoutd.<br>Altre variabili d'ambiente: $HOME, $USER, $SHELL, $TERM. |
| env | Visualizza l'elenco delle variabili d'ambiente. |

| File speciali | Descrizione |
|---|---|
| /etc/passwd | Righe sono info ogni utente divise da ":". |
| /etc/shadow | Righe sono password utente codificate. |
| /etc/group | Righe sono info ongi gruppo divise da ":". |
| /usr/bin/passwd | Cambia la pass utente. |

| Directory | Descrizione |
|---|---|
| cd *percorso* | Sposta logicamente in una diversa directory, secondo un path asssoluto o relativo. |
| ls | Visualizza i files/direcotry contenuti nella directory corrente. |
| ls -a | ls ma mosta anche file nascosti (anche ., ..). |
| ls -l | ls ma mostra più informazioni sui files. |
| ls *nomefile* | Dice se il file esiste o no, e con i parametri opzinali da altre info solo per quel file. |
| pwd | Visualizza il percorso assoluto, da / fino alla directory corrente. |
| touch *nomeFile.estensione* | Crea un file vuoto nella dyrectory corrente. |

| Scripting | Descrizione |
|---|---|
| echo *testo* | Visualizza a video la sequenza di caratteri passata fino al primo "INVIO". |
| \ | Disabila interpretazione per il carattere successivo, andata a capo, permettendo di stamparlo. |
| echo "*testo*" | Come echo, ma disabilita interpretazione caratteri speciali e andate a capo. |
| ; | Separatore di più comandi lanciati durante la stessa digitazione. |
| nome=*valore* | Simboli con nome e valore, stringa modificabile, alfanumerici casesensitive. No spazi prima o dopo "=". Sono d'ambiente o ex-novo locali.<br>Solo la bash in cui sono create le variabili le può usare. I programmi lanciati dalla bash hanno una speudocopia della bash. |
| export nomevar \|nomevar=valore | Variabile d'ambiente. Un shell figlia riceve una copia modificabile che non influenza variabile d'ambiente del padre. |
| unset *nomevariabile* | Elimina una variabile esistente (vuota o no).<br>Quotare ("...") sempre variabili per evitare errori con variabili vuote o inesistenti. |
| ${nomeVari} | Fa sostituire il nome della variabile con il valore. graffe opzionali se nome variabile seguito da uno spazio. |
| #... | Commeto. |
| #!... | Se indicato nella prima riga indica quale interprete deve eseguire lo script. Se non specificato usato quello corrente. |
| *comando1* \|*comando2* | pipe (speudo-file temporaneo): collega automaticamente l'output di un comando all'input di un altro. Unidirezionale. |

| Privilegi | Descrizione |
|---|---|
| chmod u+x *script.sh* | Modifica permessi file mediante formato numerico: u+x terna 0-7. Ogni numero è la somma dei valori associati hai permessi di r(4), w(2), x/s(1). Ordine: proprietario, gruppo, altri utenti.<br>Può diventare un quartetto aggiungendo per primo l'identificatore numero dei privilegi di *setuid, setgid, sticky bit.* |
| chgrp ??? | Modifica il gruppo di appartenenza di un file. |
| chown *newOwner nameFile* | Modifica il proprietario (e anche gruppo) di un file. |
| ls -al *nomeFile.estensione* | ls che mostra permessi, anche dei file nascosti. Interpretazione: 1°carattere tipo file (- file, d dyrectory, c collegamento seriale, b device a blocchi), 9 caratteri successivi terzine di permessi (r read,w write,x/s execute) per proprietario, gruppo, altri utenti. |
| whoami | Dice all'utente corrente le sue informazioni. |
| sudo *comando* | fa eseguire il comando come administratore, può essere chiesta userPass. Solo utenti gruppo sudo (gestito dall'admin) possono usarlo. |

| **Subshell** | **Descrizione** |
|---|---|
| bash | Crea una bash figlia, eredita dal padre: dir. corrente, copia variabili d'ambiente. Non sono ereditate le variabili locali.<br>Creata in automatico da: comandi raggruppati, script, processi in background.<br>Nota: i built-in eseguiti in shell corrente/padre. |
| bash -c *script.sh* | Esegue in una subshell, con l'interprete opzionalmente indicato, lo script specificato. |
| var=val comando | Scrivendo le assegnazioni prima dell'esecuzione di un comando si creano delle var. d'ambiente solo per l'imminente subshell. Non saranno ereditate da successive subshell. |
| . \|source *script.sh* | Esegue lo script nella shell corrente.<br>Utile a impostare/modificare variabili shell.<br>Ignorata prima riga opzionale e eseguito con interprete corrente. |
| exit | Termina bash corrente, elimina l'ambiente e sale alla padre. |
| top | Mostra in tempo reale processi in esecuzione e risorse di sistema usate. |
| ps | Mostra i processi in esecuzione.<br>Con l'opzione *-all* vedo più informazioni (PID, PPID,ecc). |
| set | Visualizza tutte variabili della shell corrente. |

| **Vari** | **Descrizione** |
|---|---|
| ./ *conado* | Esegue comando presente nella directory corrente (percorso relativo).<br>Alternativa è inserire il percorso relativo o aggiungere il percorso assoluto alla variabile PATH. |
| strace *comando* | Dice la sistem-call usata. |
| clear | Pulisce la CLI. Non modifica variabili create. |
| which *comando* | Cerca in PATH il comando e se lo trova mostra il path in cui si trova. |
| cat *nomeFile.estensione* | Visualizza il contenuto del file. |
| lsmod | Elenca tutti i moduli attivi. |
| modinfo *nomeModulo* | Dice le informazioni sul modulo specificato. |
| sudo modprobe *nomeModulo* | Carica il modulo specificato. |
| history | Visualizza comandi, numerati, precedentemente eseguiti, anche da shell precedentemente chiuse.<br>Con !NUMERO lancio il comando corrispondente nell'elenco di history. Con !STRINGA lancio il comando più recente che corrisponde alla stringa. |

Directory "proc" info sui processi memorizzati con una dir. per ogni processo attivo, create e cancellate continuamente.

# 2 Programmazione Concorrente

## 2.1 Multi-processo

Un processo possiede: Il proprio PID (gestito dall'os), il PID del processo figlio o 0 e il pid del processo "iziale" (gestito da dscrittore di file). Comandi:

- **fork()** Crea un processo figlio. Condivide codice successivo alla fork e possiede una copia dei dati.
  Restituisce il pid del figlio al padre ($PID_F > 0$) e 0 al figlio ($PID_F = 0$) o un codice d'errore (¡0).
- **exit(0)** Termina un processo restituendo lo stato indicato come parametro (0 stato successo).
  Se padre termina prima del figlio non si possono liberare le risorse.
- **wait(NULL)** Fa attendere al processo la terminazione del processo figlio. Restituisce il pid del processo terminato.
- **wait(pidProcesso)** Fa attendere al processo corrente uno specifico processo.
- **waitpid(pidProcesso, &status, statoAttesso)** Fa aspettare un thread specifico con uno stato specifico.
- **exec** Sostituisce codice e dati a un processo. Non crea processi figlio.

Struttura base programma multi-processo:

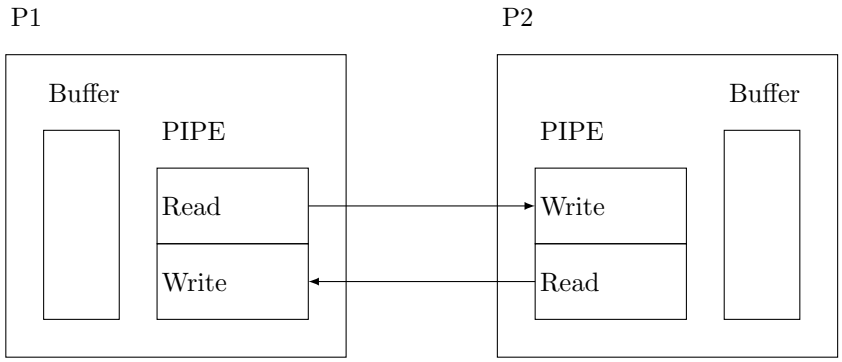
```
int main() {
 pid_t pid;

 pid = fork();

 if (pid<0) {"gestione errore"}
 else if (pid==0) {"processo figlio"}
 else {"processo padre"}
}
```

Ogni processo tiene referenza di un unico figlio, anche se ne può creare diversi. Quindi dopo ogni fork è buona cosa separare i flussi dei due processi.

### 2.1.1 Comunicazione fra Processi (PIPE)

Permette di comunicare fra processi correlati usando la funzione **pipe(nomePipe)** e i descrittori di file. Esiste il collegamento fino a eliminazione esplicita o del processo.

Ogni processo ha un buffer di caratteri e una pipe (array di due celle) per scrivere e leggere con l'altro processo i dati contentuti nel buffer.



Comandi:

- **pipe(varPipe[2] : int) : int** Crea una pipe unidirezionale.
  Su varPipe[0] si leggerà. Su varPipe[1] si scriverà.
  Restituisce -1 in caso di errore, 0 se creazione avviene con successo.
- **close(varPipe[0|1])** Chiude una estremità della pipe.
- **read(varPipe[0], buffer, bufferSize)** Blocca esecuzione in attesa di dati da leggere.
- **strcpy(buffer, "...")** Prepara il buffer con il messaggio da inviare.
- **write(varPipe[1], buffer, sizeBuffer+1)** Scrive nella pipe.

Nota: I processi sono visti come file, per questo le operazioni si chiamano come quelle dei file.

#### 2.1.2 Inclusioni speciali

- **#include ¡sys/types.h¿** Definire tipi di dato speciali usati nelle chiamate di sistema.
- **#include ¡unistd.h¿** Include funzioni e costanti del sistema operativo Linux/Unix.

### 2.2 Multi-thread

- **nomeFunzioneThread(arg : void*) : void*** Funzione assegnata da eseguire a un thread.
  Necessita questa firma specifica per accettare e restituire qualsiasi tipo di dato. Serve eseguire un cast esplicito.
- **pthread_create(&varThread, &pthreadAttribut, tFunction, &args)** Crea un nuovo thread dentro al processo corrente. I parametri sono:
  1. Variabile tipo thread.
  2. Puntatore a struttura di attributi del thread. Default è NULL.
  3. Funzione che sarà eseguita dal thread.
  4. Puntatore a struttura contenente parametri usati dalla funzine.
- **pthread_join(&varThread, NULL)** Fa attendere processo la fine del thread indicato.

Struttura base programma multi-processo:

```
void *tFunction(void *args) {...}
int main() {
 pthread_t thread;
 ... args = ...;

 ptread_create(&thread, NULL, tFunction, &args);
}
```

Nota: La creazione di un processo è più lenta della creazione di un thread perchè nella prima bisogna creare un intero file descriptor, mentre nella seconda parziale.