

# Sistemi Operativi

Leonardo Mengozzi

Titoletti indice link a rispettive sezioni, in alto a sinistra "[←Indice](#)" link a pagina Indice.

## Contents

<b>1</b>	<b>Bash</b>	<b>2</b>
	Comandi Variabili . . . . .	2
	Comandi File speciali . . . . .	2
	Comandi Directory . . . . .	2
	Comandi Controllo Comandi . . . . .	2
	Comandi Scripting . . . . .	3
	Comandi Espressioni aritmetica . . . . .	4
	Comandi Espressioni condizionali . . . . .	4
	Comandi Tilde Expansion . . . . .	5
	Comandi Privilegi . . . . .	5
	Comandi Subshell . . . . .	5
	Comandi Uso File Descriptor . . . . .	6
	Comandi Standard input e output . . . . .	7
	Comandi Utilities . . . . .	7
	Comandi Process Identifier . . . . .	8
	Comandi Vari . . . . .	10
	Comandi Costrutti controllo flusso . . . . .	10
<b>2</b>	<b>Crittografia</b>	<b>10</b>
	2.1 Crittografia Simmetrica (chiave segreta) . . . . .	10
	2.2 Crittografia Asimmetrica (chiave pubblica) . . . . .	11
	2.2.1 Funzioni Hash Crittografiche . . . . .	11
	2.3 MAC e Firma digitale . . . . .	12
	2.4 GPG e PGP . . . . .	12
<b>3</b>	<b>Programmazione Concorrente</b>	<b>12</b>
	Comandi Multi-processo . . . . .	12
	Comandi PIPE . . . . .	13
	Comandi Multi-thread . . . . .	13
	3.1 SJF . . . . .	14
	3.2 Produttore e consumatore . . . . .	14
	3.3 Buffer limitato . . . . .	15
	3.4 Filosofi orientali a cena . . . . .	16
	3.5 Message Passing . . . . .	17
	3.6 Concorrenza in C . . . . .	17
	Comandi Funzioni semafori . . . . .	17
<b>4</b>	<b>Esercizi Scheduling</b>	<b>18</b>
<b>5</b>	<b>File Descriptors</b>	<b>18</b>
<b>6</b>	<b>Risorse</b>	<b>19</b>
	6.1 Grafo di Holt Generale (classi e processi) . . . . .	19
	6.2 Detaction and recovery - Caaso 1 . . . . .	19
	6.3 Detaction and recovery - Caaso 2 . . . . .	20
	6.4 Detaction and recovery - Knot . . . . .	20
	6.5 Stato SAFE - algoritmo del banchiere . . . . .	20

<b>7 Gestione Pacchetti in Linux</b>	<b>20</b>
7.1 Concetti chiave . . . . .	20
7.2 Installazione Package . . . . .	20
7.2.1 Posizione Repository . . . . .	21

## 1 Bash

Variabili	Descrizione
PATH	V. Ambiente: 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. Altre variabili d'ambiente: \$HOME, \$USER, \$SHELL, \$TERM.
env	Visualizza l'elenco delle variabili d'ambiente.
IFS=\$' \t \n'	Contiene caratteri separatori della parole negli elenchi.
/dev/null	File speciale che scarta tutto quello che gli viene scritto.
LC_CTYPE	Imposta il tipo locale dei caratteri. Usata da <b>cut</b> e <b>ws</b> per capire numero byte caratteri.

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

Directory	Descrizione
cd <i>percorso</i>	Sposta logicamente in una diversa directory, secondo un path assoluto o relativo.
mkdir <i>nomeDir</i>	Crea una nuova directory.
touch <i>nomeFile.estensione</i>	Crea un file vuoto nella directory corrente.
rmdir <i>nomeDir</i>	Rimuove una directory solo se è vuota.
rm <i>file dir</i>	Rimuove una directory vuota o un file. Parametri: <ul style="list-style-type: none"> <li>• <b>-r</b> elimina ricorsivamente sotto cartelle e file.</li> <li>• <b>-f</b> non fa chiedere le autorizzazioni di eliminazione.</li> </ul>
mv <i>file1 file2 dir</i>	Rinomina file1 in file2 o sposta file1 nella directory specificata.
cp <i>file1 dir</i>	Copia file1 nella directory specificata.
ls [ <i>nomefile</i> ]	Visualizza i files/directory contenuti nella directory corrente. Parametri: <ul style="list-style-type: none"> <li>• <b>-a</b> mostra anche file nascosti (anche ., ..).</li> <li>• <b>-l</b> mostra più informazioni sui files.</li> <li>• <b>-h</b> rende i dati più leggibili.</li> <li>• <b>-d</b> fa applicare il comando alla directory e non ai file.</li> <li>• <b>-R</b> mostra ricorsivamente contenuto sotto directory.</li> </ul> <p>Se specifico un file mi dice se esiste e mi da informazioni solo di lui.</p>
pwd	Visualizza il percorso assoluto, da / fino alla directory corrente.

Controllo Comandi	Descrizione
\	Disabla interpretazione per il carattere successivo, andata a capo, permettendo di stamparlo.
"..."	Delimita un argomento e non fa interpretare nessun comando a eccezione dell'espansione di variabili (\$..) e l'esecuzione di comandi.
'...'	Delimita un argomento e non fa interpretare nessun comando.
\$'...'	Espande backslash-escaped direttamente nella stringa espansa in una single-quoted string. Backslash-escaped: \a, \e, \f, \r, \v, \', \b, \E, \n, \t, \\", \", \nnn, \xHH, \cn.
pre{s1,...}post	Stringa di testo racchiusa fra separatori (spazio, tab, a capo) con coppia di graffe (non precedute da \$) e senza separatori. Le stringhe racchiuse dalle graffe vengono composte con il preambolo (pre) e postscritto (post), che sono opzionali. Alternative: Sono annidabili (quelle più esterne eseguite per prime). Vengono eseguite prima le brace expansions delle variable expansions. <ul style="list-style-type: none"> <li>• <math>a_1..a_2</math> lettere da <math>a_1</math> a <math>a_2</math> nell'alfabeto.</li> <li>• <math>n_1..n_2</math> numeri compresi tra <math>n_1</math> e <math>n_2</math>.</li> </ul>
cmd1 ; ...	Separatore di più comandi, e dei rispettivi argomenti, scritti sulla stessa riga di comando e eseguiti dopo la terminazione del precedente (lista di comandi). L'exit status è quello dell'ultimo comando lanciato. Se racchiusi da () eseguiti in una sub-shell.
cmd1   cmd2	Esegue cmd1 e solo se cmd1 fallisce (exit status≠0) esegue cmd2.
cmd1 && cmd2	Esegue cmd1 e solo se cmd1 ha successo (exit status=0) esegue cmd2.
[[...]]	Espressione condizionale, usa && e   , che restituisce 0=true, altro=false.
*	Sostituito con una qualsiasi sequenza di caratteri anche vuota.
?	Sostituito con un singolo carattere (no spazio vuoto).
[c1c2...]	Sostituito con solo uno dei caratteri specificati in elenco. Alternative: <ul style="list-style-type: none"> <li>• <math>a_1..a_2</math> lettere da <math>a_1</math> a <math>a_2</math> nell'alfabeto.</li> <li>• <math>n_1..n_2</math> numeri compresi tra <math>n_1</math> e <math>n_2</math>.</li> <li>• [:digit:] una cifra.</li> <li>• [:upper:] un carattere maiuscolo.</li> <li>• [:lower:] un carattere minuscolo.</li> </ul> Annidabili.

Scripting	Descrizione
<code>echo <i>testo</i></code>	Visualizza a video la sequenza di caratteri passata fino al primo "INVIO". Se passo " <i>testo</i> " si disabilita l'interpretazione dei caratteri speciali e andate a capo. <ul style="list-style-type: none"> <li>• <b>-e</b> Stampa i caratteri speciali (\...).</li> <li>• <b>-n</b> Non fa andare a capo.</li> </ul>
<code>nome=<i>valore</i></code>	Simboli con nome e valore, stringa modificabile, alfanumerici case sensitive. No spazi prima o dopo "=". Sono d'ambiente o ex-novo locali. Solo la bash in cui sono create le variabili le può usare. I programmi lanciati dalla bash hanno una pseudocopia della bash.
<code>\$variabile</code>	Fa l'espansione della variabile, ovvero la sostituisce con il suo contenuto.
<code>#variabile</code>	Restituisce il numero di caratteri del contenuto della variabile.
<code>\${!variabile}</code>	Fa l'espansione della variabile che contiene il nome d'un'altra variabile con il valore di quest'ultima (riferimento indiretto). Dalla versione 2 di bash.
<code>export nomevar  nomevar=valore</code>	Variabile d'ambiente. Un shell figlia riceve una copia modificabile che non influenza variabile d'ambiente del padre.
<code>unset <i>nomevariabile</i></code>	Elimina una variabile esistente (vuota o no). Quotare ("...") sempre variabili per evitare errori con variabili vuote o inesistenti.
<code>\${nomeVar}</code>	Fa sostituire il nome della variabile con il valore. graffe opzionali se nome variabile seguito da uno spazio.
<code>#...</code>	Commeto.
<code>#!...</code>	Se indicato nella prima riga indica quale interprete deve eseguire lo script. Se non specificato usato quello corrente.
<code><i>comando1</i>  <i>comando2</i></code>	pipe (pseudo-file temporaneo): collega automaticamente l'output di un comando all'input di un altro. Unidirezionale sinDes.
<code><i>script.sh</i> c1...</code>	Sono un insieme ordinato di caratteri separati da spazi successivi al nome del programma. Sono immutabili dopo la sostituzione dei metacaratteri (*, ?, ecc). Riga di comando = nomeProgramma + parametri. Nella riga di comando gli elementi sono indicizzati da 0 (nomeProgramma). <ul style="list-style-type: none"> <li>• <b>\$#</b> Contiene il numero di parametri passati.</li> <li>• <b>\$n</b> Accede all'n-esimo parametro a partire da indice 0.</li> <li>• <b>\$*</b> Tutti argomenti concatenati e divisi da spazi.</li> <li>• <b>\$@</b> Vettore di argomenti quotati ("...").</li> </ul> I parametri <b>\$*</b> e <b>\$@</b> sono identici se non quotati (concatenazione di argomenti separati da " "). Se quotati <b>\$*</b> quota tutti gli argomenti assieme mentre <b>\$@</b> quota singolarmente ogni argomento. <b>\$@</b> è usato per passare parametri a comandi dentro a degli script.
<code>'comando ./script.sh'</code>	<b>Command substitution.</b> Sostituisce (a run-time) nella riga in cui è specificato il comando o script con l'output (stdout). Comando alternativo <code>\$(./script.sh)</code>
<code>\$?</code>	Modificato alla terminazione di ogni script, contiene l' <i>exit status</i> .

Espressioni aritmetica	Descrizione
<code>((...))</code>	Valuta una stringa come un'espressione aritmetica (+, -, *, /, %, (), !, &&,   ) di soli interi. Racchiude un'espressione più eventualmente un assegnamento. Si possono usare variabili nell'espressione (\$variabile). Exit status 0=true, altro=false. Non contiene espressioni condizionali e comandi. Per le operazioni in virgola mobile usare <b>bc</b> .
<code>\$((...))</code>	Come operatore <code>((...))</code> ma è concatenabile con stringhe tramite " ".

Espressioni condizionali	Descrizione
<code>[[ ... ]]</code>	<p>Restituisce exit status 0=true, altro=false.</p> <ul style="list-style-type: none"> <li>Non può contenere comandi, word splitting, brace expansion, pathname expansion.</li> <li>Non esegue assegnamenti a variabile, annidamenti di espressioni condizionali.</li> <li>Può contenere variable expansion, solo <code>\$()</code>, command substitution (se non genera comandi), process substitution, quote removal. Questi solo per gli operandi.</li> <li>Può andare a capo.</li> <li>Si possono usare gli operatori logici <code>!</code>, <code>&amp;&amp;</code>, <code>  </code>, <code>()</code>.</li> </ul> <p>Operatori unari/binari non quotabili ne generaibili da command substitution:</p> <ul style="list-style-type: none"> <li>Operazioni sui file: <code>-e</code> (esistenza), <code>-d</code> (cartelle), <code>-f</code> (file), <code>-h</code> (link), <code>-r</code> (leggibile), <code>-s</code> (size&gt;0), <code>-t</code> (fd open e riferisce un terminale), <code>-w</code> (scrivibile), <code>-x</code> (eseguibile), <code>-O</code> (possesso effettivo utente), <code>-G</code> (possesso effettivo gruppo), <code>-L</code> (=h).</li> <li>Confronto date ultima modifica file: <code>f1 -nt f2</code> (f1 nuovo o f2 inesistente), <code>f1 -ot f2</code> (f1 vecchio o inesistente).</li> <li>Operatori aritmetici: <code>-eq</code> (==), <code>-ne</code> (!=), <code>-le</code> (&lt;=), <code>-lt</code> (&lt;), <code>-ge</code> (&gt;=), <code>-gt</code> (&gt;).</li> <li>Operatori lessicografici: <code>==</code>, <code>=</code>, <code>!=</code>, <code>i</code>, <code>j</code>, <code>l</code>, <code>z</code> (size==0), <code>-n</code> (size!=0).</li> <li>-o da vero se opzione è abilitata per la shell.</li> </ul> <p>Versioni vecchie: <code>[ .. ]</code>, test ... no a capo (semmai <code>\</code>), <code>-a</code>, <code>-o</code>, <code>no</code> <code>()</code>. Mettere sempre spazi prima e dopo.</p>

Tilde Expansion	Descrizione
<code>/...</code>	Tilde espansa con il percorso assoluto della home directory dell'effective user. Valido caso con solo <code>~</code> e solo <code>/. </code>
<code>userName/...</code>	Tilde e userName espansi con il percorso assoluto della home directory dell'utente specificato. Valido caso con solo <code>~userName/</code> .

Privilegi	Descrizione
<code>chmod u+x script.sh</code>	Modifica permessi file mediante formato numerico: u+x terna 0-7. Ogni numero è la somma dei valori associati ai permessi di r(4), w(2), x/s(1). Ordine: proprietario, gruppo, altri utenti. Può diventare un quartetto aggiungendo per primo l'identificatore numero dei privilegi di <i>setuid</i> , <i>setgid</i> , <i>sticky bit</i> .
<code>chgrp ???</code>	Modifica il gruppo di appartenenza di un file.
<code>chown newOwner nameFile</code>	Modifica il proprietario (e anche gruppo) di un file.
<code>ls -al nomeFile.estensione</code>	Mostra permessi, anche dei file nascosti. Interpretazione: 1°carattere tipo file (- file, d directory, 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.
<code>whoami</code>	Dice all'utente corrente le sue informazioni.
<code>sudo comando</code>	fa eseguire il comando come amministratore, può essere chiesta userPass. Solo utenti gruppo sudo (gestito dall'admin) possono usarlo.

Subshell	Descrizione
bash	<p>Crea una shell figlia (interattiva non di login).  Eredita dal padre: dir. corrente, copia variabili d'ambiente. Non sono ereditate le variabili locali.  Creata in automatico per comandi raggruppati, script, processi in background. I comandi built-in sono eseguiti in shell corrente/padre.</p> <ul style="list-style-type: none"> <li>• <b>-c <i>script.sh</i></b> non interattiva.</li> <li>• <b>-l</b>   <b>-login</b> interattiva di login.</li> </ul>
var=val comando	<p>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.</p>
.  source <i>script.sh</i>	<p>Esegue lo script nella shell corrente.  Utile a impostare/modificare variabili shell.  Ignorata prima riga opzionale e eseguito con interprete corrente.</p>
exit	<p>Termina bash corrente, elimina l'ambiente e sale alla padre.</p>
exit <i>exitStatus</i>	<p>Termina lo script restituendo un valore intero [0-255] per indicarne l'esito di terminazione. 0 indica esecuzione terminata senza errori, qualcosaltro indica un'errore. Viene restituito alla shell esecutrice.</p>
top	<p>Mostra in tempo reale processi in esecuzione e risorse di sistema usate.</p>
ps	<p>Mostra i processi in esecuzione.  Con l'opzione <i>-all</i> vedo più informazioni (PID, PPID,ecc).</p>
set	<p>Visualizza sia variabili locali che d'ambiente della shell corrente (anche funzioni di shell). I parametri [re]settano dei comportamenti della shell:</p> <ul style="list-style-type: none"> <li>• <b>+o <i>comando</i></b> Disabilita il comando (tipo history).</li> <li>• <b>-o <i>comando</i></b> Abilita il comando (tipo history).</li> <li>• <b>-a</b> Successive variabili create/modificate diverranno d'ambiente e ereditate da shell figlie. Per [ri]definire variabili locali usare <b>export -n <i>variabile</i></b>.</li> <li>• <b>+a</b> Successive variabili create/modificate diverranno locali e non ereditabili da shell figlie. (default).</li> </ul>

Uso File Descriptor	Descrizione
\$\$	Variabile con PID della shell corrente. Utile per esplofare /proc/ e fd vari processi.
exec {n}modalità file.estensione	Apri file e l'associa al fd scelto dall'utente (n). Modalità: < (lettura), > (scrittura), >> (aggiungi in coda), <> (lettura e scrittura).
exec {var}modalità file.estensione	Apri file e l'associa al fd scelto dall'os inserito in var.
exec {n}>&-	Chiusura di file con fd=n.
exec {var}>&-	Chiusura di file con fd contenuto in var.
program modalità file prog2 ...	Ridirezionamento input/output. Modalità: < (strin da file), > (strout sovrascrive file), >> (srtout accoda a file),  (output prog1 è input prog2) Per ridirezionare su uno specifico fd del programma si fa: N >, N >>, < N. Quando si omette si sottintende strin, strout, strerr. Nota: Il fine file da tastiera si fa con "Ctrl+D". Si possono fare contemporaneamente i ridirezionamenti I/O. L'ordine di specificazione non conta.
program 0/1/2& > file  &0/1/2	Ridireziona lo strin, strout e strerr del programma nel file indicato sovrascrivendo o in un altro str. Ridirezionare simultanea in file separati: prog 2> fileErr > fileOut
program N> &M ...	Fa puntare lo stream di N allo stesso stream di M (N e M sono fd). Si possono fare più ridirezionamenti (fd e file, fd e fd) in simultanea. Sono eseguiti da sinistra a destra.
prog1  & prog2	Ridireziona lo stderr nello stdout di prog1 e lo passa allo strdin di prog2.
costrutto modalità file	Applica il ridirezionamento a tutti i comandi contenuti nel costrutto e alla condizione.
program << word	Word specificata è delimitatore di fine input da passare al comando. Nell'input la word deve essere a inizio di una riga con solo essa.
program <<< word	Ridireziona in input la word. Se si qouta ("...") si possono passare più dati.
(cmd1;...)	Comandi eseguiti in subshell con stdin/out/err condivisi e concatenati. L'exit status è quello dell'ultimo comando eseguito. Permette ridirigere più comandi simultaneamente.

Standard input e output	Descrizione
read var	<p>Legge dallo stdin o file una sequenze di caratteri (fino all'INVIO o a capo) e la inserisce nella variabile passata. Ignora spazi e tabulazioni inizio e fine riga (amenocché si setti IFS="").</p> <p>Exit status 0=lettura eseguita, &gt;0=eof (var="") o ultima riga senza \n (var="ultimi caratteri"). Necessario controllo con #var.</p> <p>Parametri che considerano spazi e tabulazioni:</p> <ul style="list-style-type: none"> <li>• <b>-n numero</b> Numero massimo di caratteri da leggere.</li> <li>• <b>-N numero</b> Numero esatto di caratteri da leggere.</li> </ul> <p>Considera \n come semplice carattere. Usa prossime righe se attuale ha caratteri insufficiente.</p> <p>Una read parte dalla riga precedente, se non era stata finita di leggere (fino a \n). Il parametro <b>-u \${var}</b> fa leggere dal file di file descriptor in var.</p>
read var1 ...	<p>Alle variabili viene assegnata i-esima parola della riga tranne per l'ultima variabile che riceve la stringa rimanente.</p> <p>Alle variabili in eccesso viene assegnato valore vuoto.</p>

Utilities	Descrizione
head -n <i>n file</i>	Mostra le prime n righe del file. Default 10.
tail -n <i>n file</i>	Mostra le ultime n righe del file. Default 10.
sed 's/ <b>originale</b> / <b>nuovo</b> /[g n];...' <i>file</i>	<p>sostituisce il testo originale (o più testi originali con [o1, o2, ...]) con quello nuovo, solo per la prima occorrenza. Può richiere l'opzione <b>-r</b>.</p> <ul style="list-style-type: none"> <li>• Con l'opzione g si applica per tutte le occorrenze in una riga oppure a "n" occorrenze.</li> </ul> <p>Si possono concatenare più sostituzioni applicare sulla stessa riga.</p> <ul style="list-style-type: none"> <li>• ^ significa inizio linea. Va all'inizio.</li> <li>• \$ significa fine riga. Va alla fine.</li> <li>• . significa un carattere qualsiasi. <b>.{n}</b> significa n. caratteri qualsiasi. .* significa 0 o più caratteri qualsiasi.</li> <li>• [...] caratteri o insiemi di caratteri (a1-an) da sostituire.</li> </ul> <p>backreference: In "originale" si specificano tra () stringhe riusabili in "nuovo" tramite</p>
i-esima backreference.	
cut <i>modalità file</i>	<p>Estrae colonne o intervalli di caratteri dalle righe. Modalità:</p> <ul style="list-style-type: none"> <li>• <b>-c S-D</b> Prende i caratteri dall'indice S all'indice D (inclusi). S o D si possono omettere e si prende dall'inizio o dalla fine fino a S o D. L'indice parte da 1.</li> </ul> <p>Posso concatenare più intervalli dividendoli con ",".</p> <ul style="list-style-type: none"> <li>• <b>-d '...' -fN</b> Divide la stringa in parole secondo il carattere delimitatore indicato poi seleziona la parola con l'indice f-ennesimo.</li> </ul>
cat <i>nomeFile.estensione</i>	Visualizza il contenuto del file.
grep <i>stringa [file]</i>	<p>Filtra le righe contenenti la stringa passata (che sarà evidenziata), anche se contenuta in altre parole, dal testo che sarà successivamente scritto (termina con "Ctrl+d") o dal file passato opzionalmente.</p> <ul style="list-style-type: none"> <li>• <b>-v</b> Filtra righe senza parola da evidenziare.</li> <li>• <b>-i</b> Disabilita il caseSensitive.</li> <li>• <b>-h o -H</b> Disabilita la visualizzazione del nome del file dove è avvenuta la ricerca (nel caso si cerchi in più file).</li> </ul> <p>Spesso usato come secondo comando di  per filtrare output di altri comandi.</p>
tee	...



Process Identifier	Descrizione
<code>\$\$</code>	Variabile espansa col PID (identificatore numerico univoco di processo) della shell corrente. Col raggruppamento di comandi (...; ...) viene espanso col PID della shell più esterna.
<code>\$BASHPID</code>	Da in ogni caso il PID della shell corrente. Comando poco portabile.
<code>script &amp;</code>	Lancia processo in background. Si omette ; (& già terminatore comando). \$! ha il pid del processo lanciato in bg (anche se sganciato). Non è aggiornata se faccio bg di un processo sospeso da <code>^Z</code> .
<code>jobs</code>	Mostra lista processi background e sospesi con rispettivo indice di job. Con <code>%indiceJob</code> si gestisce il job.
<code>^Z</code>	Sospende processo in foreground. Lancia il segnale SIGTSTP
<code>^C</code>	Termina processo in foreground. Lancia il segnale SIGINT.
<code>bg</code>	Riprende l'esecuzione di un processo sospeso ma in background. Lancia il segnale SIGCONT.
<code>fg %n</code>	Riprende l'esecuzione in foreground il Job specificato. Lancia il segnale SIGCONT.
<code>kill [-valore] pid %n [-1]</code>	Termina processo di PID o JobId indicato. Manda un segnale (interruzione software per notificare evento asincrono), default SIGTERM=15. In generale un segnale è accettato solo se proviene dall'effective user e poi viene gestito/ignorato dal processo interessato. Segnale SIGKILL=9 non può essere ignorato. -l da tutti segnali lanciabili. Con -1 si fanno terminare tutti i figli diretti della bash.
<code>disown pid %n, ...</code>	Sgancia jobs dalla shell. Viene sganciato ultimo job messo in bg se non si specifica nulla. <ul style="list-style-type: none"><li>• <b>-r</b> Sgancia tutti job in running.</li><li>• <b>-a</b> Sgancia tutti job in running e suspend.</li></ul>
<code>nohup script &amp;</code>	Lancia in bg e sgancia ultimo processo messo in bg.
<code>%% %+ %</code>	Riferisce job corrente (ultimo interrotto) in foreground o avviato in background.
<code>%-</code>	Riferisce penultimo job interrotto in foreground o avviato in background.
<code>trap "funzione/listaComandi" segnale1,...</code>	Azione (funzione o lista comandi) da svolge alla ricezione di certi segnali predefiniti. Funzione: <code>nomeFunzione(){...}</code>
<code>wait pid %n, ...</code>	Fa attendere la shell la terminazione del processo figlio di pid o JobId indicato (o un insieme di processi). Ha exit status l'exit status dell'unico o dell'ultimo processo terminato da attendere. Solo wait fa attendere tutti figli e da exit status 0.

## Terminologia:

- **Processo:** Minima unità completa ed'autonoma d'elaborazione. Composto da più thread operanti in un contesto = spazio d'indirizzamento + tabella dei descrittori di file aperti.
- **Gruppo di processi:** Insieme variabile di processi avviati da un processo padre.
- **Terminale di controllo:** astrazione della console=video+tastiera (stdin, stdout, stderr) opzionale per un processo.
- **T.c. del gruppo di processi:** terminale controllo padre ereditato dai figli, ameno di sganciamenti volontari.
- **Sessione:** Creata e legata a un terminale di controllo da una shell (leader sessione). `sessionId=PID` leader sessione. Shell prende dal terminale di controllo stdin, stdout, stderr.  
Processi figli appartengono alla sessione, ameno di distacchi, e condividono terminale di controllo.  
Un terminale è t.c. per una sola sessione alla volta. La chiusura del t.c. comporta la terminazione dei processi del gruppo (SIGHUP).
- **Processo foreground:** Controlla terminale da cui è stato lanciato (stdin collegato e impedisce esecuzione fino alla sua fine ad'altri programmi).  
Uno processo foreground alla volta.

- **Processo background:** Eseguito in parallelo rispetto bash madre (bash può leggere e eseguire indipendentemente). Hanno una copia fd della bash.  
Terminati, ameno di sganciamento, alla chiusura del terminale della bash madre (stdin condiviso).  
Più processi background alla volta.
- **Job control:** Componente che cambia stato processi da background a foreground e viceversa.
- **Jobs:** Processi in background o figli sospesi di una shell
- **Processo zombie:** Processo figlio terminato (risorse state rilasciate dall'os) di cui l'os mantiene il pcb (pid, exit status, ecc).  
Del tutto eliminato quando padre fa la wait (rilascio pcb).
- **Processo orfano:** Processo figlio di cui il padre è terminato senza fare wait. Viene adottato dal processo `init` (PID=1) di cui farà la wait periodicamente (rilascio pcb).

Vari	Descrizione
<code>./ comando</code>	Esegue comando presente nella directory corrente (percorso relativo). Alternativa è inserire il percorso relativo o aggiungere il percorso assoluto alla variabile PATH.
<code>strace comando</code>	Dice la sistem-call usata.
<code>clear</code>	Pulisce la CLI. Non modifica variabili create.
<code>which comando</code>	Cerca in PATH il comando e se lo trova mostra il path in cui si trova.
<code>lsmod</code>	Elenca tutti i moduli attivi.
<code>modinfo nomeModulo</code>	Dice le informazioni sul modulo specificato.
<code>sudo modprobe nomeModulo</code>	Carica il modulo specificato.
<code>history</code>	Visualizza comandi, numerati, precedentemente eseguiti, anche da shell precedentemente chiuse. Con <code>!NUMERO</code> lancio il comando corrispondente nell'elenco di history. Con <code>!STRINGA</code> lancio il comando più recente che corrisponde alla stringa.
<code>find file</code>	Trova percorsi di file in sotto alberi della memoria. <ul style="list-style-type: none"> <li>• <b>-type d</b> Trova solo directory.</li> <li>• <b>-type f</b> Trova solo file.</li> <li>• <b>-maxdepth n</b> Limita la profondità di ricerca il numero di livelli indicato.</li> <li>• <b>-mindepth</b> Indica il livello minimo di ricerca.</li> <li>• <b>-name "..."</b> Definisce un vincolo sul nome del file da trovare.</li> <li>• <b>-iname</b> Disabilita il caseSensitive.</li> <li>• <b>-exec ... "</b> ... <b>\";</b> Istruzioni da fare quando viene trovato un elemento. L'output di find viene messo al posto di <code>"</code>.</li> </ul>
<code>wc</code>	Dice il testo passatogli il numero di righe, parole e caratteri.

Directory `/proc` : info sui processi, con una dir. per ogni processo attivo, create e cancellate continuamente.  
Directory sulla RAM `/proc/` : contiene i processi in esecuzione e nella sottodirectory `fd` i file descriptor aperti.

Costrutti controllo flusso	Descrizione
<code>for varName in elencoParole ; do listCommand ; done</code>	Dopo il for variabile è usabile.
<code>for ((exp1 ; exp2 ; exp3)) ; do listCommand ; done</code>	Le exp sono espressioni valutate aritmeticamente.
<code>if listA ; then listB ; [elif listC ; then listD ; ...] ... [else listZ ;] fi</code>	...
<code>while list ; do list ; done</code>	...

In tutti i costrutti l'exit value è o quello della lista do comandi o 0 se nessun comando viene eseguito.

Espressioni condizionate su file o variabile: `[ condizione di un file ]`. Valutazione di un'espressione matematica applicata a variabili d'ambiente: `(( istruzioni con espressione ))`

## 2 Crittografia

Algoritmo pubblico/noto, robusto e chiavi segrete (o note in parte) di varie dimensioni.

- Codificare (E):  $testoOriginale + chiaveCodifica \xrightarrow{codifica} testoCodificato$ .
- Decodificare (D):  $testoCodificato + chiaveDecodifica \xrightarrow{decodifica} testoOriginale$ .

Impossibile ottenere testo originale dal testo cifrato senza conoscere la chiave.

## 2.1 Crittografia Simmetrica (chiave segreta)

Stessa chiave  $K$  per codificare e decodificare per  $A$  e  $B$ .

+ Veloce, +Efficente, -distribuzione chiavi fase rischiosa, -tante chiavi ( $n^*(n-1)/2$ ).

$$A_K \xrightarrow{testo} E_K \xrightarrow{E_K(testo)} D_K \xrightarrow{testo} B_K$$

Algoritmo noto a tutti, chiave nota solo a  $A$  e  $B$ .

- SI: segretezza, autenticazione, integrità
- NO: Non repudiabilità

Adottato da Des, Triple-DES, AES, ecc.

## 2.2 Crittografia Asimmetrica (chiave pubblica)

Una chiave pubblica/nota ( $K^+$ ) e una chiave privata/segreta ( $K^-$ ) sia per  $A$  che per  $B$  **correlate**<sup>1</sup>.  $A$  e  $B$  non codivono chiavi segrete.

Due chiavi private-pubbliche (correlate) criptano e decriptano in modo reciproco. La chiave privata è incalcolabile dalla chiave pubblica.

+Sicurezza (non scambio completo chiavi), -Lento, -assenza chiavi pubbliche proprietà.

Configurazioni:

- $A$  sa a chi spedisce ma  $B$  non sa chi gli scrive.

$$A_{K_A}^{K_A^+} \xrightarrow{testo} E(K_B^+) \xrightarrow{K_B^+(testo)} D(K_B^-) \xrightarrow{testo} B_{K_B}^{K_B^+}$$

SI: segretezza.

- $A$  non sa chi capisce cosa scrive ma  $B$  sa chi scrive.

$$A_{K_A}^{K_A^+} \xrightarrow{testo} E(K_A^-) \xrightarrow{K_A^-(testo)} D(K_A^+) \xrightarrow{testo} B_{K_B}^{K_B^+}$$

SI: Autenticazione, Integrità, Non repudiabilità.

- $A$  sa chi capisce cosa scrive e  $B$  sa chi scrive.

$$\begin{array}{ccc} A_{K_A}^{K_A^+} & & B_{K_B}^{K_B^+} \\ \downarrow \text{testo} & & \uparrow \text{testo} \\ E_1(K_A^-) & & D_2(K_A^+) \\ \downarrow K_A^-(testo) & & \uparrow K_A^-(testo) \\ E_2(K_B^+) & \xrightarrow{K_B^+(K_A^-(testo))} & D_1(K_B^-) \end{array}$$

SI: Segretezza, Autenticazione, Integrità, Non repudiabilità.

Adottato da RSA, ecc.

<sup>1</sup>Un testo cifrato con una chiave può essere decifrato solo dalla chiave a cui è correlata (legata).

### 2.2.1 Funzioni Hash Crittografiche

Una **Funzione Hash crittografiche** ( $MD()$ ) (one-way hash, funzioni digest) prende in input un messaggio ( $m$ ) lungo a piacere e da in output un digest (stringa) a lunghezza fissa ( $MD(m)$ ).

Il digest non è reversibile, ossia non esiste  $MD(x) = MD(y)$  oppure dato  $y = MD(x)$ ,  $x$  è indeterminabile.

Più due  $m$  sono simili più i digest sono diversi. Più due  $m$  sono diversi non è detto che i digest siano molto diversi.

Le hash non sono decodificabili, non servono chiavi di cifratura. Le funzioni di codifica/decodifica non restituiscono output a lunghezza fissa.

Hash è diverso per quanto simili dal checksum (perchè è reversibile).

Funzioni hash: MD5, SHA-1/2(x, 256, 512, ecc)/3, ecc.

## 2.3 MAC e Firma digitale

MAC (Message Authentication Code) è una tecnica, veloce, di autenticazione parziale dell'origine di un messaggio. Inoltre garantisce l'integrità del messaggio.

**MAC:** Si attua combinando crittografia simmetrica e funzioni Hash crittografico. Attori devono conoscere la chiave segreta. Assente autenticazione di chi spedisce (potrebbe essere l'altro o se stesso). SI: Integrità, autenticazione.

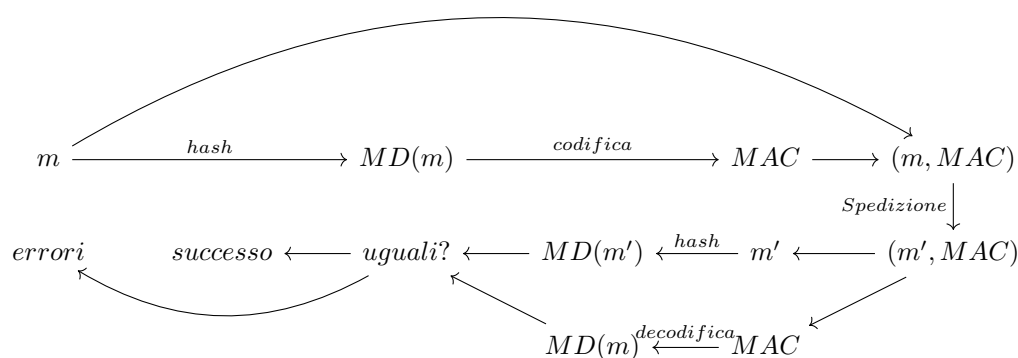
**Firma Digitale:** Tecnica di MAC avanzata. Si attua combinando crittografia asimmetrica e funzioni Hash crittografico. Mit. deve avere chiavi asimmetriche. Chiunque può leggere e verificare l'integrità. SI: Integrità, Autenticazione, Non repudiabilità.

**Spedizione:**

1. Applico Funzione hash crittografico a  $m$  ottenendo  $MD(m)$ .
2. Codifico il diget simmetricamente o asimmetricamente (con la chiave privata del mittente) ottenendo il MAC.
3. Concatenazione MAC al  $m$ .
4. Spedizione ( $m$ , MAC) che può modificare  $m$  in ( $m'$ , MAC).

**Ricezione:**

1. Separazione di ( $m'$ , MAC).
2. Decodifica simmetrica o asimmetrica (usa chiave pubblica mittente) del MAC ottenendo il digest ( $MD(m)$ ).  
Terminazione improvvisa se decodifica non da successo. Implica che mittente non è quello atteso.
3. Applico Funzione hash crittografico a  $m'$  ottenendo  $MD(m')$ .
4. Confronto i due digest. Se sono uguali  $m'$  è l'originale spedito dal mittente dichiarato.



## 2.4 GPG e PGP

GPG meccanismo di firma digitale gratuito basato su PGP proprietario. Garantisce autenticità e integrità di un file/package.

Il **maintainer**<sup>2</sup> della risorsa possiede chiavi asimmetriche e per ogni risorsa da pubblicare crea la firma digitale e la distribuisce con la risorsa. Oggi firmato solo file indice di risorse (Release.gpg) che ha versione e checksum (digest) di ogni risorsa.

L'utente, scarica se non l'ha la chiave pubblica, scarica il package+firma verifica firma con la chiave pubblica del maintainer.

Le chiavi pubbliche si possono trovare di default nell'OS o si scaricano da canali ufficiali verificando a modo.

<sup>2</sup>Qualcuno che distribuisce, assicura il funzionamento e aggiornamento di una risorsa online.

### 3 Programmazione Concorrente

Inclusioni necessarie:

- **#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.

Multi-processo	Descrizione
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 ( $\neq 0$ ).
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, statoAtteso)	Fa aspettare un thread specifico con uno stato specifico.
exec	Sostituisce codice e dati a un processo. Non crea processi figlio.

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

Struttura base programma multi-processo:

```
int main() {
    pid_t pid;

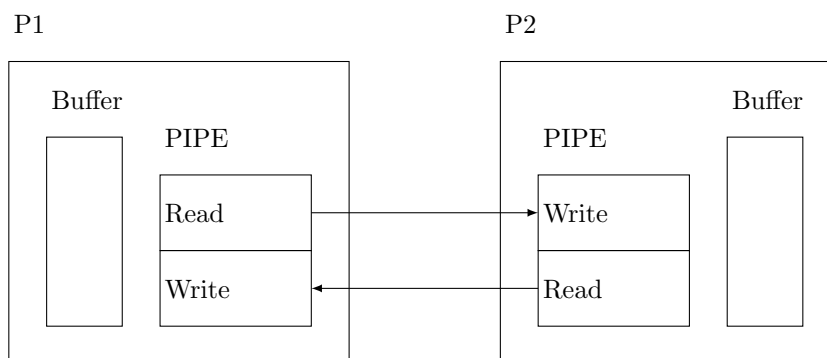
    pid = fork();

    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.

PIPE	Descrizione
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)	Scriva nella pipe.

Permette di comunicare fra processi correlati usando sistem-call e i descrittori di file. Il collegamento esiste 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 contenuti nel buffer.

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

Multi-thread	Descrizione
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: <ol style="list-style-type: none"> <li>1. Variabile tipo thread.</li> <li>2. Puntatore a struttura di attributi del thread. Default è NULL.</li> <li>3. Funzione che sarà eseguita dal thread.</li> <li>4. Puntatore a struttura contenente parametri usati dalla funzione.</li> </ol>
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 = ...;

    pthread_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.

### 3.1 SJF

Calcolo approssimato CPU Bust:  $T_{n+1} = \alpha t_n + (1 - \alpha)T_n$ .

- $t_n$  tempo n-esimo CPU burst. Storia recente.
- $T_n$  previsione prevista. Storia passata.
- $\alpha$  peso storia recente e passata.

Calcolo Media esponenziale:  $T_{n+1} = \sum_{j=0}^n \alpha(1 - \alpha)^j t_{n-j} + (1 - \alpha)^{n+1} T_0$ .

### 3.2 Produttore e consumatore

$[producer] \xrightarrow{\text{genera/trasferisce}} <variable> \xleftarrow{\text{consuma/preleva}} [consumer]$

```
shared Object buffer;

Semaphore empty =
    new Semaphore(1);

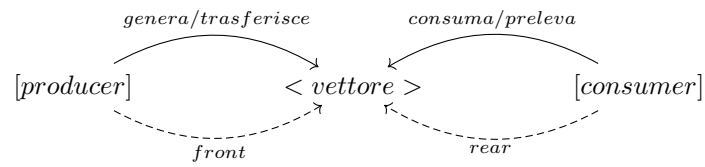
Semaphore full =
    new Semaphore(0);

cobegin
    Producer
//
    Consumer
coend

process Producer {
    while (true) {
        Object val = produce();
        empty.P();
        buffer = val;
        full.V();
    }
}

process Consumer {
    while (true) {
        full.P();
        Object val = buffer;
        empty.V();
        consume(val);
    }
}
```

### 3.3 Buffer limitato



```
Object buffer[SIZE];  
int front = 0;  
int rear = 0;  
Semaphore empty =  
    new Semaphore(SIZE);  
Semaphore full =  
    new Semaphore(0);  
  
process Producer {  
    while (true) {  
        Object val = produce();  
        empty.P();  
        buf[front] = val;  
        front = (front + 1) % SIZE;  
        full.V();  
    }  
}  
process Consumer {  
    while (true) {  
        full.P();  
        Object val = buf[rear];  
        rear = (rear + 1) % SIZE;  
        empty.V();  
        consume(val);  
    }  
}
```

### 3.4 Filosofi orientali a cena

$pensa \xrightarrow{\text{-----}} [Filosofi] \xrightarrow{\text{mangia}} \langle chopsticks \rangle$

```
Semaphore chopsticks =
  { new Semaphore(1), ..., new Semaphore(1) };

process Philo[i] { /* i = 0...4 */
  while (true) {
    think
    chopstick[i].P();
    chopstick[(i+1)%5].P();
    eat
    chopstick[i].V();
    chopstick[(i+1)%5].V();
  }
}
```

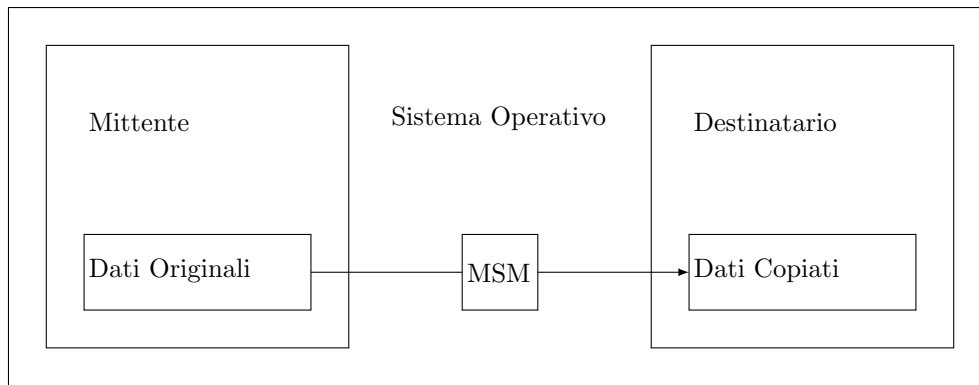
Le battecche sono un vettore lungo 5 (0...4) e i filosofi accedono alle battecche  $i$  e  $i+1$  in %5.

Rischio di deadklock!!! Esistono varie soluzioni, avere un filosofo mancino è la più semplice:

```
process Philo[0] {
  while (true) {
    think
    chopstick[1].P();
    chopstick[0].P();
    eat
    chopstick[1].V();
    chopstick[0].V();
  }
}
```



### 3.5 Message Passing



Legenda: p=mittente, m=messaggio, q=destinatario.

#### Sincrono:

- Mittente attende l'ACK (Acknowledgement) del destinatario.  
`ssend(m,q)`
- Destinatario attende la spedizione di m da p o da un qualsiasi mittente (\*). Poi manda l'ACK.  
`m=sreceive(p|*)`

#### Asincrono:

- Mittente non attende l'ACK del destinatario.  
`asend(m,q)`
- Destinatario attende la spedizione di m da p o da un qualsiasi mittente (\*). Non manda l'ACK.  
`m = areceive(p|*)`

#### Completamente asincrono:

- Mittente non attende l'ACK del destinatario.  
`asend(m,q)`
- Destinatario non attende la spedizione di m da p o da un qualsiasi mittente (\*). Non manda l'ACK.  
`m = nb-receive(p|*)`  
Se non si riceve nulla restituisce `null`.

Nella pratica: lo scambio di messaggi avviene tra canali di comunicazioni su cui mandare e ricevere. L'istruzione `replay(Thoth)` sblocca il mittente (e non la receive).

### 3.6 Concorrenza in C

Usata la libreria POSIX **semaphore.h** (API POSIX con semafori, lock mutex e variabili condizionali).

Tipi di semaforo:

- Con nome (**named**) utile a sincronizzare processi non correlati.
- Senza nome (**unnamed**).

Funzioni semafori	Descrizione
<code>sem_t * &lt; variabile &gt;</code>	Definisce una variabile di tipo puntatore a struttura Semaforo.
<code>sem_init(&amp;&lt; varSem &gt;, 0/altro, jvaloreIniziale<sub>i</sub>)</code>	Crea un semaforo unnamed inizializzato al valore passato e lo assegna alla variabile d'indirizzo indicata. Secondo parametro a 0 semaforo condiviso tra thread stesso processo, a $\neq 0$ conviso tra processi.
<code>sem_open("nomeSem", 0_CREAT,0666, jvaloreIniziale<sub>i</sub>)</code>	Crea e restituisce un semaforo named in R/W (3° parametro) inizializzato al valore passato.
<code>sem_wait(&amp;varSem)</code>	Esegue l'operazione P (decrementa).
<code>sem_post(&amp;varSem)</code>	Esegue l'operazione V (incrementa).
<code>sem_destroy(&amp;varSem)</code>	Distrugge un semaforo.

Le CS saranno contenute tra un wait e un post.

## 4 Esercizi Scheduling

Procedura risolutiva di un esercizio di scheduling data la **durata del quanto di tempo** e una tabella dei processi del tipo:

Processi	Tempo di Arrivo	Tempo d'Esecuzione
...	...	...

### \*Passo 1\*

Fare una tabella di scheduling come la seguente:

Ready Queue				
Running				
Quanto d'Inizio/Fine				
Tempo	0	1	...	n
T.R. P1				
...				
T.R. PN				

Spiegazione delle righe della tabella:

1. *Ready Queue* specifica per ogni quanto i processi pronti per essere eseguiti, tipicamente è FIFO.
2. *Running* indica per ogni quanto di tempo il processo in esecuzione.
3. *Quanto d'Inizio/Fine* evidenzia i quanti d'arrivo (A) e di termine (F) dei processi.
4. *Tempo* sono i quanti del processore. Ogni colonna è un quanto.
5. *Righe T.R.* tempo d'esecuzione rimanente per ogni processo a ogni quanto di tempo dalla sua partenza.

### \*Passo 2\*

Per rispondere alle statistiche richieste fare una tabella come la seguente:

Processi	T. Turnaround	Intervalli di Ready	T. Attesa
...	...	[x1-x2], ...	...
Medie:	...	...	...

Spiegazione delle colonne della tabella:

1. *Turnaround* = T. fine - T. arrivo.
2. *Intervalli di Ready* sono i quanti in cui il processo è stato nella coda di ready (dal quanto in cui vi è entrato al quanto in cui vi è uscito).
3. *T. Attesa* somma degli intervalli di ready.

## 5 File Descriptors

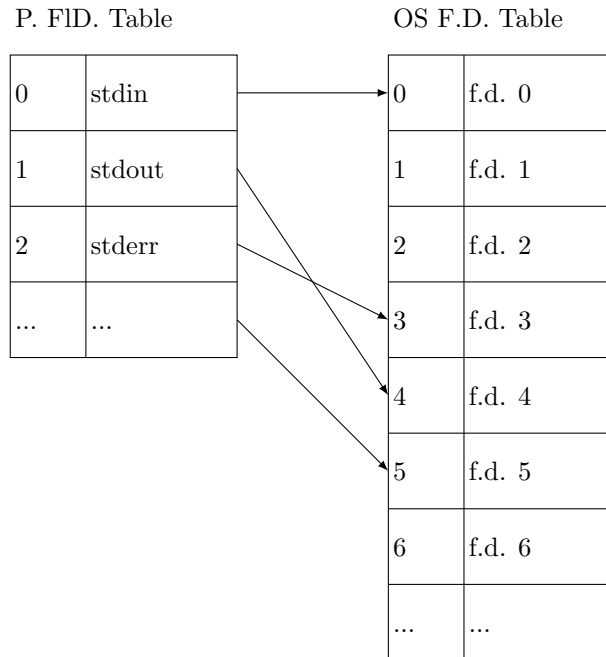
Il file descriptor è un'astrazione (integer, inizio buffer) dell'os, e quindi multi linguaggio, per permettere l'accesso ai file.

Ogni processo ha la **file descriptor table** di record indiretti delle informazioni (file descriptor) sui file aperti dal processo stesso. I record sono indiretti perchè puntano ai record della **tabella di sistema**. La tabella di sistema contiene file descriptor dei file attualmente aperti.

Un processo/shell figlia eredita una copia della f.d. table, successivamente modificabile. Padre e figlio si devono sincronizzare sull'accesso ai file. Ciò permette un modo di comunicazione tra i processi.

Rindirizzamenti:

- **Processi figlio:** Padre può cambiare le associazioni fd/stream passate al figlio.
- **Auto-ridirizionamento:** fd viene associato a stream diverso.



In POSIX per ogni processo i file descriptor standard sono: stdin=0, stdout=1, stderr=2.

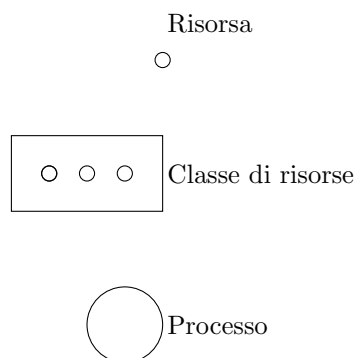
Ricorda che stdin è lo stream input da tastiera e stdout e stderr sono gli stream a video dell'output e dei messaggi d'errore.

Nota: Processi diversi possono avere file descriptor diversi, ma con stesso valore, e che referenziano file diversi.

In C per scrivere si usa `jprintf(stdout/stderr, "...", [parametri])` e per leggere si usa `jfgets(buffer, size, stdin)`, per il fine file usare `jfeof(stdin)`.

## 6 Risorse

### 6.1 Grafo di Holt Generale (classi e processi)

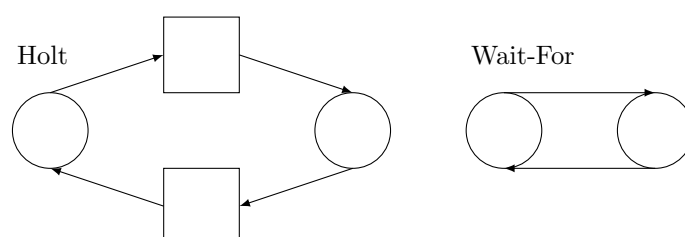


Per gli archi si può segnare la molteplicità, e nelle classi il numero di risorse non assegnate.

### 6.2 Detaction and recovery - Caaso 1

Teorema: Siano le risorse ad'accesso mutualmente esclusivo, seriali e non prerilasciabili. Con una risorsa per classe si ha deadlock  $\iff$  Holt contiene un ciclo (attesa circolare).

Dimostrazione: In un grafo di Holt si ha attesa circolare se si ha attesa circolare nella rispettiva variante di grafo Wait-For (elimina le risorse collassando gl'archi appropriati).



### 6.3 Detaction and recovery - Caaso 2

Teorema: Siano le risorse ad'accesso mutualmente esclusivo, seriali e non prerilasciabili. Con più risorse per classe si ha deadlock  $\iff$  Holt non è completamente riducibile (eliminazione tutti archi).

### 6.4 Detaction and recovery - Knot

Teorema: Siano le risorse ad'accesso mutualmente esclusivo, seriali e non prerilasciabili. Con una richiesta sospespa per processo si ha deadlock  $\iff$  esiste un knot.

### 6.5 Stato SAFE - algoritmo del banchiere

Sia  $s$  una permutazione dei valori  $1, \dots, N$  e  $s(i)$  l'i-esima posizione nella sequenza. Si calcola il vettore **avail** così:

1.  $avail[1]=SC$
2.  $avail[1+j]=avail[j]+p_{s(j)}$ , con  $i=1, \dots, N-1$

Uno stato del sistema è safe se vale:  $n_{s(j)}i=avail[j]$ , con  $j=1, \dots, N$ .

Con il banchiere a singola valuta basta ordinare in modo crescente gli  $n_i$  per assicurare lo stato SAFE.

Nel caso multi valuta si aggiunge il pedice k della valuta.

Con il banchiere multi valuta ogni vlauta ha un suo ordine.

## 7 Gestione Pacchetti in Linux

### 7.1 Concetti chiave

- **Package (.deb)**: archivio compresso di file descrivono programma e metadati del package.
- **Dipendenze**: Librerie necessarie funzionamento programma.
- **Repository**: Collezioni locali o remote di package in server centralizzati.
- **Database locale di package**: Elenco locale dei metadati di package nei repository.

### 7.2 Installazione Package

Interfaccia a CLI per download-installazione (risolve dipendenze e gestisce fonti), rimozione, aggiornamento, gestione di package. Usa **dpkg** per l'installazione vera e propria (disassembla package e installa in directory giuste) e **apt-cache** per la ricerca (usabili anche dall'utente).

Assente di descrizione esecuzioni (ideale per script).

**apt-get**

**apt** fornisce una descrione testuale della procedura eseguita (user freandly).

Comandi apt-get	Descrizione
update	Aggiorna database locale di package. Da eseguire per primo.
upgade	Installa le nuove versioni dei package già installati.
install	Installa un package nuovo e le dipendenze.
remove	Rimuove package lasciando file configurazione.
purge	Rimuove package e file configurazione.
autoremove	Rimuove package, non più necessari, autoinstallati soddisfare dipendenze altri package.
dist-upgrade	Aggiorna la distribuzione (package e dipendenze).
clean	Cancella package dalla cache locale.

Formato generico: `sudo apt-get "comando" "package0pzionale"`

Procedura Installazione Package:

1. `sudo apt-get update` Aggiorna l'indice/database locale scaricando l'elenco aggiornato dei package e versioni dai repository.
2. `apt-get intall "package"` Installa il package cercandolo nel database locale e controllando le dipendenze.

### 7.2.1 Posizione Repository

I database locali di package sono contenuti in: `/etc/apt/sources.list` (principale), `/etc/apt/sources.list.d/` (aggiunte repository individuali).

**Formato one-line:** `deb [opzioni] URI distribuzione comp1 comp2 ...`

- **Tipo:** Indica il tipo d'archivio.  
`deb` indica package binari precompilati.  
`deb-src` indica package di sorgenti.
- **Opzioni:** La firma GPG si attua col campo dedicato che accettà il **keyring** con la public-key del maintainer.  
`signed-by="..."`
- **URI:** Indirizzo del repository.
- **Distribuzione:** Nome in codice della versione del OS.
- **Componenti:** Definiscono il tipo di licenza (main, restricted, universe, multiverse).

Caratteristiche: chiarezza, leggibilità, facile da gestire e sicurezza esplicità.  
C'è anche un formato `deb822` più leggibile.

Directory public-key: `/usr/share/keyrings/` (key di sistema), `/etc/apt/keyrings/` (key d'utente).

Il `keyrings`, `sources.list`, `sources.list.d/` sono editabili con editor con privilegio `sudo` o GUI specifiche.  
Aggiunta Manuale:

1. Verificare esistenza directory, scarica GPG-key (sopprime output e reindirizza allo stdout), comprimerla (ASCII in binario `.gpg`) e salvataggio nel keyring.  

```
sudo mkdir -p /etc/apt/keyrings  
wget -qO- URI | sudo gpg --dearmor -o directory.gpg
```
2. Aggiungere riferimento one-line con `signed-by` del repository alla directory.  

```
echo "FormatoOneLine" | sudo tee directory.list
```
3. Aggiornamento e installazione: `sudo apt update`, `sudo apt install nomePackage`.