

RIPASSO

SISTEMA ELABORAZIONE

HARDWARE
SO
APPLICATIVI
UTENTE

• SERVIZI DI UN SO:

- INTERPRETE DEI COMANDI: L'UTENTE SCRIVE I COMANDI, LA SMELL LI INTERPRETA
- GESTIONE DEI PROCESSI: GESTIONE DEI PROGRAMMI IN ESECUZIONE
- GESTIONE UTENTI PRINCIPALE
- GESTIONE GESTIONE SECONDARIA
- GESTIONE DISPOSITIVI DI I/O
- GESTIONE FILE E FILE SYSTEM
- IMPLEMNTAZIONE NECESSARI DI PROTEZIONE: CONTROLLA SULLE ACCESSI REGOLI UTENTI A PROCESSI AL SISTEMA
- GESTIONE RETI E SISTEMI DISTRIBUTI:

• KERNEL:

- NUOCIO DEL SO, PROGRAMMA
- GESTISCE LE RISORSE \rightsquigarrow SISTEMA DI LIBERAZIONE DI LIBERE KERNELE PIÙ MOLTI TUTTI I PROGRAMMI
- AGLIEGLI: IL PIÙ BASICO CARATTERIZZATO DAL'INDIVIDUALE
- \backslash MICRO-KERNEL: FUNZ. DI BASE, SEPARAZIONE SERVIZI / STRUTTURE SO
- MONOLITICO: IL PIÙ DIFUSO, SERVIZI TRAMITE SYSTEM CALL

• BOOTSTRAP:

- PROCEDURA DI INIZIALIZZAZIONE: CARICA IL KERNEL IN MEM. CONFERMA ALL'ALLEGGERIMENTO, SALVATO IN ROM
- SYSTEM CALL: PERMETTE L'INTERAZIONE AI SERVIZI FORNITI DAL SO, FORMA PUBBL. DI CHIAMA
- IL SO ENTRA IN DUAL MODE: L'UTENTE ENTRA IN MODELO UTENTE \rightarrow BIT DI MODE = 1
IL KERNEL ENTRA IN MUO PREFERITO \rightarrow BIT DI MODE = 0

• LOGIN:

PROCEDURA DI ACCESO AD UN SISTEMA

- SMELL: INTERPRETE DEI COMANDI LINUX, \neq SO / BOUNDARY SHELL (SH)
- FILE SYSTEM: STRUTTURA DI DATI IN CUI È ORGANIZZO I FILE E DIRIGATORI

BOUNDARY AGAIN SHELL (BASH)

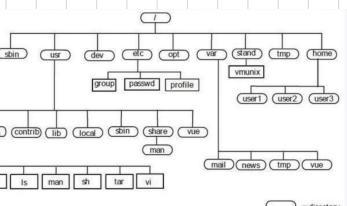
• HOME DIRECTORY: ACCESSO INIZIALE DOPO IL LOGIN

• ROOT DIRECTORY: DIRECTORY PRINCIPALE \rightsquigarrow CONFERMA CHEME I VARI UTENTI

• WORKING DIRECTORY

• THREAD: UN PROCESSO \rightarrow 3 OPI FLUSSI DI ESECUZIONE, 1 THREAD

• PIPE: FLUSSO DATI TRA 2 PROCESSI, $P_1 \rightarrow \square \rightarrow P_2$

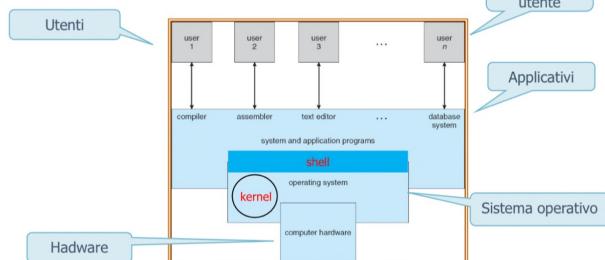


\rightsquigarrow DEADLOCK \leftrightarrow STARVATION

\hookrightarrow LIVELLOK: ANNOGLA, MA NON C'E' NEGAN INGRESSO A COMMUNE \rightarrow SOLO NON L'E' AUN PROMESSO

\hookrightarrow STARVATION: AG UN UTENTE VITNE NEGATO L'ACCESSO A UNA RISORSE UTILE A COMMUNE

SEGUITO COMPONENTI



• U03 - STRUMENTI PER LA PROGRAMMAZIONE C :

• MAKEFILE:

• > AUTOMATIZZA LE FASI DI COMPIAZIONE E LINK, PROVVEDE ALLA VERSICELLA DELLE DIPENDENZE

• SI SCRIVE UN FILE CON LA SINTASSI DEL MAKEFILE, DA SHELL LO SI RICHIESTA CON make mymakefile

• FORMATO:

target : dependency
2 tab > command

→ si indica nome di un file

target : identifica l'oggetto che può far sì che questo dependency. Se c'è più di uno in ESSE
→ US ESEGUITO

• SE \nexists target : PRIMO TARCHET → ESEGUITO SEMPRE

• U04 - FILESYSTEM:

→ GESTIONE DEI FILE ALL'INTERNO DEL SO $\xrightarrow{z^2}$ ZFS IN BLOCCHE ASCII

• ASCII : caratteri su 7 bit, codifica di 128 caratteri \rightsquigarrow sono meno

16 BIT
?

• UNICODE : UCS o UTF → +110 000 caratteri, 170 inserimenti di simboli → UTF-8, UTF-16, UTF-32

• FILE $\begin{cases} \text{BINARIO} \\ \text{TESTUALE} \end{cases}$: V: COMPATIBILE, Facilità modifica, Facilità posiz. su file, S: persistenza garantita, in stocca

• SERIALIZZAZIONE : PROCESSO DI TRASFORMAZIONE DI UNA STRUTTURA, VISUALIZZABILE COME UNA BUSTA

• I/O ANSI C : funz. di <stdio.h> per I/O

• I/O UNIX : uso di sole 5 funzioni → open, read, write, lseek, close

→ UNBUFFERED I/O : 1 operazione di I/O → 1 chiamata al KERNAL

• open():

• path: PERCORSO DEL FILE DA APRIRE

• flags: modalità APERTURA FILESS

- O_RDONLY
- O_WRONLY
- O_RDWR

{ CONTINUA IN fcntl.h
↳ VIENE FATTO UN OR tra diverse costanti

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open (const char *path, int flags);
```

```
int open (const char *path, int flags, mode_t mode);
```

S_I[RWX]USR rwx --- ---

S_I[RWX]GRP --- rwx ---

S_I[RWX]OTH --- --- rwx

• read():

• fd: FILE

```
#include <unistd.h>
```

```
int read (int fd, void *buf, size_t nbytes);
```

• nbytes: n° di byte da leggere in fd

• buf: VET. IN cui ritornare ciò che fu letto

→ return: n° byte letti \rightarrow deve essere < nbytes / -1 se FINIRE / 0 se EOF

• write () :

- fd : FILE OR DOPPIO
- nbytes : n° byte in buf
- buf : VETR. CONTENENTE LE INFO DA SCRIVERE
- return : n° bytes scritti → cioè nbytes di read / -1 se ERRORE

```
#include <unistd.h>
int write (int fd, void *buf, size_t nbytes);
```

• lseek () :

- RESTAURA UN NUOVO VETRONE ALLO POSIZIONAMENTO DI INIZIARSI DI R/W DEL FILE (OFFSET)
- whence : SPECIFICA L'INIZIARSIATORE DEL OFFSET
 - SEEK_SET : OFFSET VANTATO DA INIZIO FILE
 - SEEK_CUR : " " " " POS. CORRENTE
 - SEEK_END : " " " " FINE FILE

→ return: NUOVO OFFSET / -1 se ERRORE

```
#include <unistd.h>
off_t lseek (int fd, off_t offset, int whence);
```

• close ()

- CHIODE IL FILE DESCRITTO fd
- return 0 se SUCCESSO / -1 se ERRORE

```
#include <unistd.h>
int close (int fd);
```

• DIRETTORI :

MODO IN CUI È OGNISSAZIONE IL FILE SYSTEM: AGLI OGGETTI È UN NODO DI UN ALbero

• AD UN NODO: TUTTI I FILE NEL'INTERO DI UN'UNICA OGGETTIVITÀ

V: EFFICIENZA ; S: MULT. UTENZE, NAMING

• A UN NODO: LEGGE PRINCIPALE REGOLE VETRONE, A VETRONE → DIR, A 1 NODO

V: EFFICIENZA USER-ORIENTATO, NAMING ; S: GROUPING COME > SO PROTEZIONE SIMILE

• AD ALBERO: HIERARCHIZZAZIONE DEI POSSESSI

V: EFFICIENZA, NAMING, GROUPING.

• IN CASO DI NECESSITÀ DI CONDIVIDERE UN FILE:

• A VEDERE ACCORDO: LA CONDIVISIONE DI UN FILE, IN SO UNIX-LIKE, È REALEZZATA

TRAMITE CREAZIONE DI LNK

• VISITA A NODI: "PISANO IL LINK" PER RENDERE UN'OGGETTO IMMAGGIO

• CANCELLAZIONE (SE CANCELLA OBE. CINTUA?):

• SOFT-UNIX UNIX: CANCELLA L'OGGETTO, INTERRAUNTE, CANCELLA LINK PRIMA (DARKLINK)

CANCELLA I
DIRI MA
CANCELLAZIONE
DEGLI UNICI LINK

- HARD-LINK UNIX: TENNE TRACCIA DI NOME I LINK PRESENTI CONTRO M'OGGETTO
- MANTIENE SEMPRE IL CONTO DEI NUMERI DI UNICI → comando ls -i
- IN UNIX, PRESENTI NERI I-NODES

- DIRETTOREI C (CLC) : possono presentare dei problemi (DGL. AUTOREPRESENTANTI)
 - PRESENZA DI CLC
- ALLOCAZIONE:

• CONSIGLIA:

✓ FUE → lo memorizza in blocchi contigui di memoria

→ NE memorizza l'indirizzo 1° blocco e la sua lunghezza

V: semplicità (pochi info memorizzare), accesso sequenziale immediato, accesso diretto formula ($b = b + i \cdot s$)

S: risulta spazio libero sufficiente, spesso spazio (pratica esterna), problemi alloc. dinamica

• CONTINUA:

✓ FUE → memorizza in blocchi separati in memoria, discontigui tra loro

→ NE memorizza i puntatori al 1° e ultimo blocco \rightarrow il blocco → puntatore all'indirizzo

V: alloc. dinamica, no pratica esterna, no algoritmi complessi di allocazione.

S: memorizzazione puntatori, struttura e accesso (per via dell'accesso sequenziale)

• ACCESIONE FAT (FILE ACCESS TABLE):

→ varianti della costruzione, in cui i puntatori successivi non sono memorizzati nei blocchi stessi, ma in un'unica tabella

S: dopo accesso (FAT + b), accesso (dato, ripetitiva), semplicità delle operazioni da eseguire

• INICIIZZATA:

✓ FUE → è blocco in cui sono memorizzati tutti i suoi puntatori in sequenza \rightarrow tabella: i-node

↳ blocco (inode)

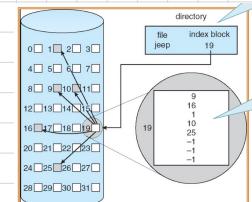
→ memorizza solo l'indice del blocco contenente i puntatori

UNIX-like

• in caso lo spazio riservato per i-node non basta \longrightarrow sovrapposizione

• schema combinato: ✓ i-node: 15 puntatori + 12 indiritti al blocco + 3 indiritti

→ 13° puntatore: simbolo \rightarrow 1 solo link di puntatore
 14/15° \rightarrow 00000/numero \rightarrow 2/3 indiritti di puntatori



NO
 ?
 SEZIONE: • IL LINK DA UN OGGETTO AL PROPRIO i-NODE È UN HARO LINK \rightarrow VERSO oltrerrail (CLC)

• MANIPOLAZIONE FUE SYSTEM: \rightarrow USO STANDARD POSIX

• STRUCT stat: tipologia di struttura in cui le informazioni del tipo stat() scambiano le informazioni

• stat(): richiede un puntatore sb di tipo struct in cui si salvano le info

• path è il percorso di cui tornano le info

→ permette di capire di che tipo di oggetto si tratta
 return 0 successo / -1 errore

• lstat(): restituisce info sul link singolo, nel caso in cui il path invoca un link

• fstat(): .. " .. su un FUE già aperto

```
#include <sys/types.h>
#include <sys/stat.h>

int stat (const char *path, struct stat *sb);
int lstat (const char *path, struct stat *sb);
int fstat (int fd, struct stat *sb);

struct stat {
    mode_t st_mode; /* file type & mode */
    ino_t st_ino; /* i-node number */
    dev_t st_dev; /* device number */
    dev_t st_rdev; /* device number */
    ...
};

Tipi primitivi
```

getcwd() : ~) GET CURRENT WORKING DIRECTORY

-> riceve UN char* e la ritorna. Dessa struttura

-> OTTENNE IL PATH DEL DIRITTORE DI LAVORO

return buf SE SUCCESSO / NULL SE ERRORE

chdir():

-> RICEVE IL NUOVO PATH DI LAVORO

, return 0 SE SUCCESSO / -1 ERRORE

-> CREA IL PATH DI LAVORO

mkdirm (const char *path, mode_t mode)

-> CREA UN NUOVO DIRITTORE

return 0 SE SUCCESSO / -1 ERRORE

rmkdir (const char *path)

-> CANCELLA UN DIR, SE VUOTO

return 0 SE SUCCESSO / -1 ERRORE

```
#include <dirent.h>

DIR *opendir (
    const char *filename
);

struct dirent *readdir (
    DIR *dp
);

int closedir (
    DIR *dp
);
```

) APRE UN DIR IN LETTURA

return DIR* corretta / NULL ERRORE

) CREA UN DIRITTORE

return DIR* corretta / NULL ERRORE O TERMINA

) CHIAMA UN DIR

return 0 corretto / -1 ERRORE

```
struct dirent {
    ino_t d_ino;
    char d_name[NAM_MAX+1];
    ...
}
```

UOS - PROCESSI:

S01 • POSSIBILITÀ DI CREARE, IDENTIFICARE, CONTINUARE E TERMINARE PROCESSI

• A PROCESSO -> 3! PID : PROCESS IDENTIFIER, È UN INT32 > 0 UNIVOCO

unistd.h

```
pid_t getpid();    RESTRUISCE IL PID DEL PROC. CORRENTE
pid_t getppid();   "    "    "    " PADRE
uid_t getuid();    "    10. IDL' UTENTE CHIAMANTE
gid_t getgid();   "    "    "    " GRUPPO
```

CREAZIONE DI UN PROCESSO:

• fork(): CREA UN PROCESSO FIGLIO, ANALOGO AL PAREN MA CON \neq PID

return IL PADRE RICEVE PID DEL FIGLIO, IL FIGLIO RICEVE 0, -1 SE NUOVO PR. NON ALLOCATO

PID = X
 F
 P = 0
 P = 1

• CFG (CONTROL FLOW GRAPH): RAPP. IL FLOW DI CONTROLLO

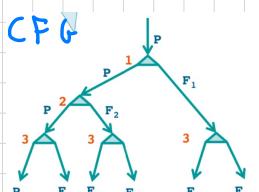
• ATP (ALTERNATE THREADING PROCESS)

ES.

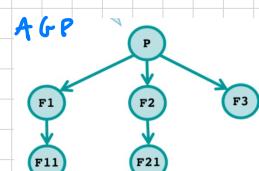
```
pid = fork (); // 1
if (pid != 0)
    fork (); // 2
    fork (); // 3
```



CFG



AGP



ESEGUITO A NIVELLO
 ~) KERNEL

{ 0: SCHEDULATORE DI PROCESSO

1: init(), ESEG. DOPO
 BOOTSTRAP, PADRE DI UN
 PROCESSO DIFANO

TERMINAZIONE DI UN PROCESSO:

- 5 modi standard:

- return 0/1 & principale
- exit()
- exit V-Exit, return simile a exit()
- return da main dell'ultimo thread nel processo
- pthread_exit() dell'ultimo thread nel processo

• avendo un processo terminato, il mentre manca SIGCHLD se parso,

IL SISTEMA: { gestire la sua terminazione → } ASINCRONICO: invia segnale di SIGCHLD
 ignorare la sua → (DEFOLY) SINCRONICO: wait() o waitpid()

wait():

BLOCCA IL PROCESSO IN ATTESA DELLA VARIANTE DI RITORNO DI UNO DEI FIGLI

return PID del figlio avendo terminato / -1 se errore o ≠ Figlio

- *statLoc: indica lo stato di TERMINAZIONE DEL FIGLIO
- uso WIFEXITED(statLoc) per verificare che la TERMINAZIONE sia corretta
- uso WEXITSTATUS(statLoc) per estrarre gli 8 LSBs del parametra presenti nel figlio non exit

es. SE IN FIGLIO: ... exit(2) → IN PARENTE: WEXITSTATUS(statLoc) := 2

F, P • ZOMBIE: PROCESSO TERMINATO PER CHI IL PADRE NON HA ANCORA ESEGUITO wait()

P, F • ORFANO: PROCESSO DI CHI IL PADRE È TERMINATO PRIMA DI PARTE wait() → diventa figlio di init / pid(imib):=1

waitpid():

→ SEMPRE PER ASpettare UN FIGLIO CON UN PID SPECIFICATO

→ NON BLOCCANTE

- .pid:
 - = -1 : ATTENDI UN UNIPROCESSO FIGLIO CON UN PID SPECIFICATO
 - > 0 : " FIGLIO CON pid
 - 0 : " uniprocessor FIGLIO CON groupID = A unico DEL GRUPPO
 - < -1 : " FIGLIO / SUB FIGLIO DA "

#include <sys/wait.h>

pid_t waitpid(pid_t pid, int *statLoc, int options);

*statLoc: (ANACRONICO)

*options: 0 oppure OR di costanti, permette controlli di terminazione

• WNORM: VERSIONE NON BLOCCANTE DELLA wait() → si eseguisce non si parla se il figlio di PID specificato non esiste

• WCONTINUED, WUNTRACED: permettono di conoscere lo stato di UN FIGLIO IN CONCERN. PROBLEMI

3 modi non standard:

- abort(): quando il segnale SIGABRT
- SIGKILL: DI UN SEGNALE
- cancellare l'ultimo thread nel processo

SIGKILL

#include <sys/wait.h>

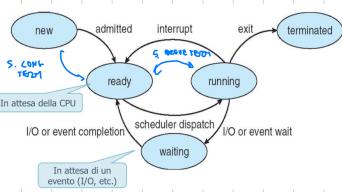
pid_t wait(int *statLoc);

• S02 - ASPECTI TEORICI:

• PCB (PROCESS CONTROL BLOCK):

✓ A PROCESSI, IL SO TIENE TRACCE DI MTCHE UE INFO DI TESSO, TIPO:

- STATO DEL PROCESSO (NEW, READY, RUNNING, WAITING, TERMINATED)
- PRIORITY COUNTING: INIZIAZIONE DELL'ESECUTIVA ISR. DA ESEGUIRE
- REGISTRI DEI CPU, E INFO UTILI PER LO SCHEDULING
- INFO UTILI PER LA GESTIONE MEMORIA
- TABEINA ESTATISTI
- INFO AMMINISTRATIVE VARIE (E UTILIZZO CPU, UTM, ecc.)
- INFO SU STATO I/O



• CONTEXT SWITCHING: PROCEDIMENTO IN CUI IL KERNEL SI OCCUPA DI SALVARE LO STATO DEL P. RUMINANTE E CARICA IL NUOVO P., RIPRISTINANDONE LO STATO PRECEDENTE, NEI MONDANI IN CUI UNA CPU VENNE ASSIGNATA A UNO NUOVO PROCESSO

EXEC OR

? CONCUPERLO DELL'CPU:

- OBIETTIVO DI MASSIMIZZARE L'UTILIZZO DELLA CPU
- DETERMINANTE: UNICO P. CORRENTE OGNI TERMINE DI UNA ESECUSIONE E SCELZIONARE IL SUCCESSIVO PROCESSO PER ESEGUIRE
- SOLUZIONI:

P. RUMN → RAM • A BREVE TERMINE: SPARGE P. TRA SEVERI IN READY, LAVORA IN RAM, MORTE VOLTI

RAM-MUT. SEC.: A MEDIO TERMINE: SPARGE P. RAM → MUT. SEC., SEGUENDO P. INIZIALE NELA RETE LIST → CONTINUOUS IC M° DI P. IN RAM

• A LUNGO TERMINE: SEGU' B.I. PROGRAMMI DA ESEGUIRE IN BASE ALLE RISORSE → NUOVA PRL LO SCELGERE A MEDIO-TERMINE

• A DISPOSIZIONE → 3 CODE DI PROCESSI: USO LISTA CONCATINATA;

• OIAL. DI ACCORDAMENTO: GESTIONE DEI PROCESSI NEMO VADE CODICE.

• S03 - CONTROLLO AVANZATO

• exec(): SOGGETTO AL PROCESSO CON UN NUOVO PROGRAMMA, MA CONTRA IL PDL

→ NE ESISTONO 6 VERSIONI: exec(), execvp(), execle(), execv(), execvp(), execve()

/ l (list): se contiene UNA LISTA CORTE (separata), V (version): ... RIVIVE UN VECCHIO CODE

p (param): INDICA IL NOME DEL FILE, e (environment): SPECIFICA LE VAR. D'AMBIENTI

• RETURNA NULL SE SUCCESSO / -1 ERRORE

• path: SPECIFICA IL PATH DEL FILE DA ESEGUIRE. (O NUOVO " -p " IL NOME (*name))

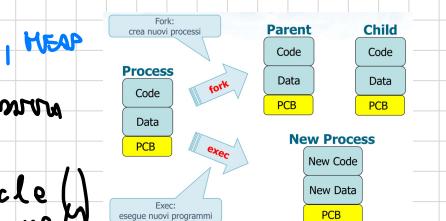
• argv / argc: SPECIFICA I PARAMETRI DA PASSARE

• envp: VAR. DI AMBIENTI

GRUPPO fork(), exec(), wait()

• system(char *string): permette di ESEGUIRE UN COMANDO IN UN PROGRAMMA

RETURNA -1 SE ERRORE system() o sem-blocca / -17 se esecut. exec()



```
#include <unistd.h>
int execl (char *path, char *arg0, ..., (char *)0);
int execcl (char *name, char *arg0, ..., (char *)0);
int execle (char *path, char *arg0, ..., (char *)0,
            char *envp[]);
int execv (char *path, char *argv[]);
int execvp (char *name, char *argv[]);
int execve (char *path, char *argv[], char *envp[]);
```

```
int execc (char *path, char *arg0, ..., (char *)0);
int execcp (char *name, char *arg0, ..., (char *)0);
```

```
int execcv (char *path, char *argv[]);
```

```
int execcvp (char *name, char *argv[]);
```

```
int execce (char *path, char *argv[], char *envp[]);
```

L'ESECUZIONE
IN BACKGROUND

• S04 - SEGNALI :

DEF.: INTERRUPT SOFTWARE INVIA AL PROCESSO UN SEGNALE PER NOTIFICARE IL VERIFICARSI DI UN EVENTO

-> GESTIONE DI EVENTI ASINCRONI

↳ KERNEL / P → P

-> COMUNICAZIONE TRA PROCESSI

• SIGCHLD: INVIA AL PADRE DAVANTI AL FINITO TERMINA, AZIONE DEFAULT: IGNORE

• SIGINT: INVIA AL PADRE AL P IN ESEC CONATO CTR+C, || : TERMINA PROCESSO

• SIGTSTP: INVIA AL PROCESSO DAVANTI PROCESSO IN NORMALE ESEGUITURA, || : SOSPENSIONE ESEGUITURA

• SIGALARM: INVIA AL P DOPO I T SECONDI DELL'SYSCALL SLEEP (Y), ||: FAR RIPARTIRE IL P

-> DEFINITI IN #include <signal.h>

• command: kill -l ↗ -> FORMAUSC ELENCO DEI SEGNALI

• GESTIONE: GESTIONE -> CONSEGNA -> GESTIONE
↳ CON SEGNALI

-> AVVIRE TRAMITE US SYSCALL:

• signal():

INSTANZA UN GESTORE DI SEGNALI
OSS SERVIZIO DI GESTIONE UNO SPECIFICO

• signum: SEGNALE DA GESTIRE

• handler: ↗ ARRIVA DA SIGNUM PER GESTIRE

↳ RETURNA VOID* SE SUCCESSO / SIG_ERR SE ERRORE

• SI POSSONO IMPOSTARE 3 COMPONENTI INSERISCI:

• SIG_DFL: COMP. DI DEFAULT

• SIG_IGN: IGNORA IL SEGNALE -> IN QUESTO CASO NON REAZIONARE (es. SIGHUP, SIGSTOP)

• signal Handler Function: ↗ UTENTE DI GESTIRE REC SEGNALI

• kill():

SPECIFICO SEGNALE A P DEMANDA ↗ NON TERMINA P!

• sig: SEGNALE DA INVIARE AL P CON PID = pid

→ SI POSSONO INVIARE SEGNALI SOLO A QUELLO CHE HA APERTO:

P DEMON ANCHE LO STESSO PID

RETURNA 0 SE SUCCESSO / -1 SE ERRORE ↗ PID ≠ pid

• raise():

↗ US STESSO

INVIA UN SEGNALE AL CHIAMANTE STESSO

-> ANALOGO kill (getpid(), sig)

#include <signal.h>

void (*signal(int signum, void (*handler)(int))) (int);

SEGNALE DA GESTIRE Dopo WHATEVER

so.

```
void manager (int sig) {
    printf ("Ricevuto il segnale %d\n", sig);
    // (void) signal (SIGINT, manager);
    return;
}
int main() {
    signal (SIGINT, manager);
    while (1) {
        printf ("main: Hello!\n");
        sleep (1);
    }
}
```



(es. SIGHUP, SIGSTOP)

#include <signal.h>

int kill (pid_t pid, int sig);

Se pid è Si invia il segnale sig ...

>0	al processo di PID uguale a pid
==0	a tutti i processi con group id uguale al suo (a cui lo può inviare)
<0	a tutti i processi di group id uguale al valore assoluto di pid (a cui lo può inviare)
== -1	a tutti i processi del sistema (a cui lo può inviare)

#include <signal.h>

int raise (int sig);

• pause():

SOSPENDE IL PROCESSO PER UN TIEMPO INDEFINITO FINO ALL'ARRIVO DI UN SEGNALE
 (return -1 => ESEGUE UN'ESTONTE DI SEGNALI Dopo TERMINE LA SUA ESECUZIONE)

```
#include <unistd.h>
```

```
int pause (void);
```

• alarm():

ARMIA UN COUNTDOWN DI SECONDI(s)

```
#include <unistd.h>
```

```
unsigned int alarm (unsigned int seconds);
```

-> AL TERMINE URGENTE GENERA SIGALRM

-> se seconds = 0 -> si esegue la funzione omologa di alarm()

return n° sec rimasti fino al SIGALRM in caso di esecuzione un'interruzione /) segnalazione

• LIMITI DEI SEGNALI:

MIGLIORIA INTEGRATA: al max 1 SEGNALE IN PENDING IF TIPO DI SEGNALE
 -> SEGNALI DELL'OGGI STESSO TIPO SUCCESSIVI SONO RITORNATI ~> POSSONOESSERE PREMESSI

ALCUNI SEGNALI POSSONO ESSERE BLOCCATI ~> POSSO ENTARE DI NUOVO

FUNZIONI RICONTROLLANTI: alcune di se interrappi da un segnale o svil. viene eseguita
 se alcun segnale ricevuto non compare con gli chiamanti interrotti
 -> ERRORE o CORRUZIONI ~> DEF. IN RICONTROLLO

-> VENUTO DEFINITE: di RICONTROLLO -> di che possano essere interrotte senza program.

-> A MAIOR PARTE NON F. DI I/O ANSI C NON SONO RICONTROLLO

RAZIE CONDIZIONI: ricevuti IN CASO DI PROGRAMMI CORRUZIONI O PROBLEMI UNICO IN DATI CORRUTTI

• VDS - comando p :

- ps : MOSTRA I P IN RUN, IN FORMATO ESTENDO
- top: , AGGIORNAMENTI RUN-TIME
- kill [-sig] pid : TERMINA UN P CON pid, -sig: MANDA UN SEGNALE SPECIALE
 L, es. kill -SIGKILL q10

• SDF - COMUNICAZIONE TRA PROCESSI:

• IPC : INTER PROCESS COMMUNICATION

• SO VORTEMENTE IMPOSTE DI ACCEDERE ALL'AREA DI MM. DI UN ACTIVO PROCESSO

→ PER CONDIVIDERE USO DI:

- MIGLORIA CONDIVISIONE: FILE (percorso): prima di fork/exec, si associa il punt. di file

- FILE MAPPA: $\forall P_1 \rightarrow$ NELLO SPACIO MM. PUÒ QUINDI ACCEDERE allo stesso file

- SCAMBIO MESSAGGI: COMMUTAZIONE. TRAMITE IL KERNEL
→ USO NUOVO system call

- COMUNICAZIONE: DIRETTA: DESYNCRONIZZARE E MIGLIORARE ESPANSIONE
INDIRETTA: TRAMITE MAILBOX

$S \leftarrow D$

$S \leftarrow \text{MAILBOX} \leftarrow D$

- SINCRONIZZAZIONE: SINCRONIA { BLOCCANTE }
ASINCRONIA { " " } → DO. PIPES

- CAPACITÀ: 0 / LIMITATA / INFINTA

↓ ↓ ↓
NO ATTESA BLOCCO SEPIENA NO BLOCCO

COME DI COMUNICAZ.: DIRETTO, ASINCRONI, CAPACITÀ LIMITATA

• PROCESSO DI DAM TRA PROCESSI



→ \forall PIPE \rightarrow 2 DESCRIPTORI, 1 ESTREMO: 1 P SOGLIE AD UN'ESTREMA, L'ALTO P CONCERNTE ALL'ALTRA ESTREMA

- MACRO-DUPLEX: $P_1 \sim P_2 / P_2 \sim P_1$, MA NON CONTEMPORANEA

• pipe():

CREA UNA PIPE.

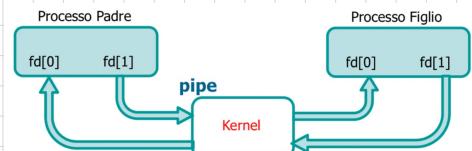
→ SONO 2 DESCRIPTORI DI FILE IN fileDescriptor[2]

- fd[0]: APERTO PER L'USO PIPE { OUTPUT SU fd[1] }
- fd[1]: " " " " " { INPUT SU fd[0] }

return 0 SUCCESSO / -1 ERRORE

```
#include <unistd.h>
int pipe(int fileDescriptor[2]);
```

→ FUNZIONE DI DAM ATTENZIONE UNA PIPE POSSA PERDERE IL KERNEL



→ P CHIAMA pipe() → fork() → P FIGLIO ESEGUE IFD DEL PADRE
L \rightarrow fd NON VERRÀ MAI PUÒ ESSERE UTILIZZATO → SIMPLEX

• SE UTILIZZANO read() E write()

- read(): SI GUADAGNA SE PIPE VUOTA, RETURNA 0 SE \nexists writer \rightarrow BLOCCANTE

- write(): " " " " " PIENA, RETURNA -1 SE PIPE SE L'ULTIMA SCRITTURA È CRISENTE \rightarrow BLOCCANTE

L \rightarrow OPERAZIONE ATOMICA (OASI SERVIZIO)

L \rightarrow DIM. MAX IN PIPE_BUF = L / 128 kB

• U06 - THREAD

→ HP → 1 SP. DI INDIRIZZAMENTO

P → RAGIONPPIA RISORSE
L → T → sincron. CPU

• S01 - THREAD : → HANNO UN UNICO SPAZIO DI INDIRIZZAMENTO, UNICA OI SCHEDULAZIONE DELL'CPU

• RISOLVONO ALCUNI LIMITI DEL PROCESSI: UTILIZZO REGISTRI, SYNC, LESIONI MULTIPLO DI P

• THREAD: UNITÀ DI SCHEDULAZIONE DELL'CPU, ESEGUITA INDEPENDENTEMENTE DAL PROCESSO

FLUSSO DI DEDIC

- CONDIVIDE UN SPAZIO CON ALTRO THREAD
- È PIÙ VELOCITÀ DI UN P

V:

- È RISPOSTA RAPIDA
- COSTI MINORI PER LETTURA RISORSE
- PROGRAMMAZIONE MULT-THREAD → SICUREZZA

↳ MIGRAZIONE SYNC

S:

- PROTEZIONE TRA I THREAD
- RENDIMENTO HIERARCHICO PAGE-FIBO

• 3 modelli di programmazione MULT-THREAD:

- KERNAL LEVEL THREAD:

LETTERAZIONE FISICA DEL NERVAL

SO È A DISPOSIZIONE SOLO IL PROCESSO, OPERAZIONI TRAMITE SYSCALL

↳ MIGRAZIONE INFO COME PER P: TRAMITE TCB → V T. INFO → TCB (Thread control Block)
INFO COMUNE ALL'INTERNO DEL SO

V: SCHEDULANTE NEI THREADS

SUDDIVISIONE DI TUTTI I VARI PROCESSI

LETTERAZIONE DELL'PROGRAMMA CHE SI ESEGUE SPETTACOLO

S: CENTRALIZZAZIONE IN INFRASTRUCTURE

LIMITAZIONE N° MAX THREAD

SO DEVE MIGRARE INFO CONTRO

- USER LEVEL THREAD:

T CREA IL PROPRIO SPAZIO DI INDIRIZZAMENTO

→ KERNAL NON È A CONOSCENZA DEI T, POSSIEDE SOLO P

→ LEGGERE PAGE-MAP OR UNA LIRETTA → USA LIB. DI SYST, NAN INTERRUZIONE HEMICO

• V P → TADOS CONTENUTO I MIGRAZIONI IN EXEC → MA OLM. POSSIBILI PROBLEMI DI CONCERNIRE NEL LIBRARY

V.

• IMPLEMENTAZIONE IN NFM E KERNAL

• NO MODIFICHE SO

• EFFICIENZA → VELOCITÀ CONTEXT SWITCH

• NO N° MAX

VERSATILITÀ

S:

• SO NON AVE CONSCIE I T

• POSSIBILITÀ DI LETTURA INFRACCIAZIONE

• COMUNICAZIONE KERNAL - U → POSSIBILE MAGGIORE NESS. PROBL. MULT. THREAD

NON BLOCCANTE IN CASO DI ERRORE



SE TOME BLOCK → NFM T UGG BLOCCATI

• BZDA:

IMPLEMENTA I VANTAGGI DI ENTREI: U DECIDE QUANTO T COME E COME REAGIRE SU T KERNALE

SD2 - C182521A pthread()

pthread_t tid;

- POSIX thread \rightarrow <pthread.h>
- If there \rightarrow $\exists!$ TID : pid_t, means identifier, or zero VOID
- TID has significance soon as internal desc T (in P PID 824 680000)
- **int pthread_equal(pthread_t tid1, pthread_t tid2);**
compara 2 TID \rightarrow return 0 se diversi / f0 se uguali
- **pthread_t pthread_self(void);** :

return TID del T corrente

- **int pthread_create();** :

- tid : lo deg T viene
- attr : attributi rec T
- start routine : funzione che eseguirà da T
- err : avviando passo sua routine
return 0 successo / COD. ERRORE

- **void pthread_exit(void *valuePtr);** :

- termina T, restituisce stato di TERMINAZIONE
- valuePtr : sono in caso di JOIN

- UN THREAD PUÒ ESSERE DI CUISENDO : \rightarrow si può usare un pthread_join()

- JOINABLE : UN SIMO T PUÒ FARE wait() SU DI SE E RICEVERE IL SUO exit_status
- DETACHED : NON JOINABLE

- **int pthread_join(pthread_t tid, void **valuePtr);** :

- tid : T da ATTENDERE
- valuePtr : corrisponde allo stato di ritorno della T su cui ha fatto la join
 \hookrightarrow PTHREAD_CANCELLED se T è stato cancellato

return 0 successo / COD. ERRORE SE ERRORE (es. caso T IN DETACH)

- **int pthread_cancel(pthread_t tid);**

return 0 successo / COD. ERRORE ERRORE

- **int pthread_detach(pthread_t tid);**

return 0 successo / COD. ERRORE ERRORE

• TERMINAZIONE T

- Un intero processo (con tutti i suoi thread) termina se
 - Un suo thread effettua una exit (_exit o _Exit)
 - Il main effettua una return
 - Un suo thread riceve un segnale la cui azione è terminare
- Un singolo thread può terminare
 - Effettuando un return dalla sua funzione di inizio
 - Eseguendo una pthread_exit
 - Ricevendo una pthread_cancel da un altro thread

```
int pthread_create(
    pthread_t *tid,
    const pthread_attr_t *attr,
    void *(*startRoutine)(void *),
    void *arg
);
```

S03 - REGEXP e find: $\rightarrow 17/30 : yy/mm/aaaa \rightarrow ([0-9])\backslash\backslash([0-9])\backslash\backslash([0-9])\backslash\backslash([0-9])$

• METACARATTERI:

Operatore	Significato
[...]	Specifica un elenco o un intervallo di simboli
(...)	Gestisce la precedenza tra operatori
Raggruppa insieme di simboli in sottoespressioni	
Permette riferimenti a espressioni precedenti (backward reference)	
	Effettua l'OR tra espressioni regolari

Ancore	Significato
\<	Inizio parola
\>	Fine parola
\^	Inizio riga
\\$	Fine riga

Caratteri speciali	Significato
\+ \? \. *	Char '+', '?', '.', '*'
\n	New line
\t	Tabulazione

Quantificatori e Intervalli	Significato
*	Elemento presente $[0, \infty]$ volte
+	Elemento presente $[1, \infty]$
?	Elemento presente $[0, 1]$ volte
$[c_1 c_2 c_3]$	Uno qualsiasi dei caratteri in parentesi
$[c_1 \cdots c_n]$	Uno qualsiasi dei caratteri nel range
$[^c_1 \cdots c_n]$	Uno qualsiasi dei caratteri non nel range
{n}	Elemento presente esattamente n volte
{n ₁ ,n ₂ }	Elemento presente da n ₁ a n ₂ volte

Caratteri	Significato
c	Un qualsiasi simbolo c (Tranne quelli utilizzati a scopi speciali)
.	Un carattere qualsiasi (non '\n')
\s	Uno spazio o una tabulazione
\d	Una cifra 0-9
\D	Non una cifra
\w	Qualsiasi carattere tra 0-9, A-Z, a-z
\W	Qualsiasi carattere non in 0-9, A-Z, a-z

• () . \z \1 : BACKWARD REFERENCE \rightarrow ciò che significa il 1° () sarà u
REFERENCE A \1, ovvero PR2 \2

• find:

COSTRUO UNA PREFERENZA DI RICERCA PER I DIRITTORI LINKE CHE PANNO DAGLI CON UNA REGEXP

• find DIRECTORY [OPZIONI] [AZIONI]

• OPZIONI: SPECIFICO LE REG EXP

nel bin DAYI. CHE FA MATCH
 \rightsquigarrow RITORNA L'INTERO PATH

• OPZIONI:

Opzione	Significato
-name pattern	Match con il nome del file. Il path iniziale è rimosso. In alcune versioni è possibile racchiudere il pattern tra doppi apici per specificare espressioni regolari.
-path pattern	-iname è identica ma case insensitive. Come il precedente ma specifica path + nome -ipath è identico ma case insensitive.
-regex expr	Specifica una espressione regolare che deve avvenire match con il path (completo). -iregex è identica ma case sensitive.
-regextype type	Indica il tipo di RE utilizzato, ovvero: posix-basic, posix-egrep, posix-extended, etc. Occorre specificare il tipo prima della RE (regextype va inserito prima di regex).
-atime [+,-]n	Ultimo access, status o modification time. n=1 specifica da 0 a 24 ore fa.
-ctime [+,-]n	Il valore di n può essere inserito con segno: + indica ≤ - indica ≥
-mtime [+,-]n	

Opzione	Significato
-size [+,-]n[bckwMG]	Dimensione del file. Il segno + indica ≥, quello - indica ≤. Il carattere successivo indica l'unità di misura: • b blocchi (di 512 byte) • c byte • k kByte • w word (2 byte) • M Mbyte • G Gbyte
-type tipo	Tipo di file. Il tipo può essere: f per file regolari (i.e., file di testo, eseguibili, etc.), p per pipe, l per symbolic link, s per socket, d per direttori
-user nome	Definizione del proprietario del file, ovvero identificativo del proprietario (user) oppure del gruppo (group)
-group nome	
-readable	Modalità di accesso, ovvero l'oggetto deve essere leggibile, scrivibile, eseguibile
-writable	
-executable	
-mindepth n	Sezione dell'albero in cui effettuare la ricerca: mindepth indica la profondità minima per la ricerca (nell'albero di direttori) e maxdepth quella massima. Con quit esce dalla ricerca dopo il primo match.
-maxdepth n	
-quit	

Azione	Significato
-print	Azione di default. Stampa un nome per ciascuna riga
-fprint	Come il precedente ma effettua l'output su file
-print0	Come -print ma non va a capo
-execdir command	Esegue il comando
-exec command	Versione sicura POSIX dell'azione precedente. Espande il comando includendo il path e il nome.
-delete	Elimina l'oggetto rintracciato

es. Azione: "exec"
 $\{ \$: SO \text{ SOSTITUISCE} \}$
 $\text{e quindi eseguisce il suo comando}$

```
find directory options -exec comando '{}';'
find directory options -exec comando '{} \';
```

504 · FILTERI :

IN UNIX | LINUX : $I \rightarrow \text{FILTER} \rightarrow O$ / $I \in \text{UN FILTER}$

comandi utilizzabili IN PIPE

SPESO
USATO
SU TESTO

• cut [opzioni] file :

RIMUOVE SEZIONI SPECIFICHE DI UN FILE
→ IL FILE È DIVISO IN SEZIONI (-d : TAB REPORT) (→ TAB REPORT)
DELIMITATORI

• tr [opzioni] set1 [set2] :

EFFECTUA SOSTITUZIONI CONDIZIONALI

· set₁ → set₂ : ASSOCIAZIONE PER POSIZIONE
→ es. tr ab BA → a → B, b → A

• uniq [opzioni] [inFile] [outFile] :

RIPORTA / ELIMINA LE RIGHE RIPETUTE IN FILE I

→ UNICA OPZIONE ELIMINA R. RIPETUTE

→ inFile deve essere originato

• basename nome [estensione] :

NOME · PATH DEL FILE DA UN ELIMINARE IL DIRIZIONALE

→ POSSO RIMUovere L'ESTENSIONE DELL FILE

• sort [opzioni] [file] :

ORDINA IL FILE IN ORDINE ALFABETICO

→ DEFAULT: INFILA CON ASCII

• grep [opzioni] pattern [file] :

CERCA NEL FILE IL MATCH CON UNA REGEXP

→ egrep PER EXTENDED RE

Opzioni			
Formato		Significato	Effetto
Compatto	Esteso		
-c LIST	--characters=LIST		Seleziona solo i caratteri di posizioni indicate
-f LIST	--fields=LIST		Indica la lista dei campi da selezionare (separati da virgola)
-d DELIM	--delimiter=DELIM		Formato: n (=n), -n (<n), n-> (n1 && n2) Esempi: 3, -3, 3, 3-5 Usa DELIM per dividere i campi (il delimitatore di default è la tabulazione)

Opzioni			
Formato		Significato	Effetto
Compatto	Esteso		
-c, -C	--complement	Complema	Utilizza il complemento del set ₁
-d	--delete	Cancella	Cancella i caratteri indicate nel set ₁
-s	--squeeze-repeats	Elimina	Sostituisce ogni sequenza di un carattere ripetuto incluso nel set ₂ con una occorrenza singola dello stesso carattere

Opzioni			
Formato		Significato	Effetto
Compatto	Esteso		
-c	--count		Stampa il numero di ripetizioni prima della riga
-d	--repeated		Visualizza solo le righe ripetute
-f N	--skip-fields=N		Ignora i primi N campi per il confronto
-I	--ignore-case		Case insensitive

Opzioni			
Formato		Significato	Effetto
Compatto	Esteso		
-b	--ignore-leading-blanks		Ignora gli spazi iniziali
-d	--dictionary-order		Considera solo spazi e caratteri alfabetici
-f	--ignore-case		Trasforma caratteri minuscoli in maiuscoli (case insensitive)
-I	--ignore-case		Case insensitive
-n	--numeric-sort		Confronta utilizzando un ordine numerico
-r	--reverse		Ordine inverso

Opzioni			
Formato		Significato	Effetto
Compatto	Esteso		
-e PATTERN	--regexp=PATTERN	Specifica il pattern da ricercare Permette di specificare pattern multipli	Dopo ciascun match stampa ancora N righe (oltre alla riga in cui si è trovato il match). Inserisce un separatore (-) dopo ogni insieme stampato.
-B N	--before-context=N	Prima di ciascun match stampa N righe (oltre alla riga in cui si è trovato il match). Inserisce un separatore (-) dopo ogni insieme stampato.	-A N --after-context=N
-l			-H --with-filename
			-I --ignore-case
			-n --line-number
		OUTPUT ONLY	-r, -R --recursive
		MATCHING FILE	-v --inverse-match

Dopo ciascun match stampa ancora N righe (oltre alla riga in cui si è trovato il match). Inserisce un separatore (-) dopo ogni insieme stampato.
Stampa il nome del file per ogni match
Case insensitive
Stampa il numero di riga del match
Procede in maniera ricorsiva sul sottoalbero
Stampa solo le righe che non fanno match

• 4.07 - SHELL

• 5.07 - SHELL :

• AMBIENTE DI PROGRAMMAZIONE NATIVO DEL SO

• /bin/sh : UNA ALTA SHELL IN USO

• FILE GI' ANTO:

• FILE DI ZERPO : SCRIPT GRAMMATICO (gg. PROPSYS)

• `` `` `` `` : SCRIPT UTENTE

• VARIABILI : \$var = \${var} = ...

• ... : str con vari INTENDO VAR. NON ESPANSE → NO AUMENTAMENTO → VAR → STRINGA

• ... : str ESPANSE → AUMENTATO → VAR → VARIABLE

• \... : ESCAPE → SCRIVONO IL SICURO ZC CHEH PERDENTI ME LO SISUWE

• ESPLORAZIONE OUTPUT DI UN COMANDO: a=\$(command) → echo \$a → STAMPA L'OUTPUT DI COMANDO

• CREAZIONE DI ALIAS PER COMANDI: alias nomealias="command" (unalias nomealias)

• 5.02 - SCRIPT :

• ESECUZIONE .sh (.bash)

• ESECUZIONE :

• ODIRITTA: ESEGUITA IN UN'ALTRA SHELL → SUCCESSO PID Nel comando ESECUZIONE DI UN PROCESSO

• INDIRETTA: ESEGUITA NELLA STESSA SHELL → STESMO PID

(argv) { .\$0 : PASSATO INIZIALMENTE CO SCRIPT ESEGUITO (quindi argv[0]) }

.\$1, \$2, \$3, ecc. PARAMETRI SUCCESSIVI

.\$* : LISTA DEI PARAMETRI (str()) DA 1:N (now argv[0])

(args) { .\$# : n° PARAMETRI (ANAGLI. argv), \$\$: PID DEL PROCESSO }

NO \$0 ← .set : PASSA NELLE LE VAR. DI SCRIPI (GLOBAL E LOCAL) CON IL RISP. VALORE

• VARIABILI:

↑ NO SPAZI

• ASSEGNAZIONE: var="valore"

• UTILIZZO: \$var

→ RENDERE UNA VAR. GLOBALE: export var

↳ VISIBLE DA TUTTE LE SHELL

• LETTURA: read [opzioni] var1 var2 var3 ecc

• OPZIONI:

• -n nchars : LETTURA FINALE DOPO ANCHE MCHARS CONSUMATI

• -t timeout : timeout SECONDI

Variabile	Significato
\$?	Memorizza il valore di ritorno dell'ultimo processo: 0 in caso di successo, valore diverso da 0 (compreso tra 1 e 255) in caso di errore. Nelle shell il valore 0 corrisponde al valore vero (al contrario del linguaggio C).
\$SHELL	Indica la shell in uso corrente
\$LOGNAME	Indica lo username utilizzato per il login
\$HOME	Indica la home directory dell'utente corrente
\$PATH	Memorizza l'elenco dei direttori separati da ':' utilizzato per la ricerca dei comandi (eseguibili)
\$PS1 \$PS2	Specificano il prompt principale e quello ausiliario (di solito '\$' e '>', rispettivamente, # per root)
\$IFS	Elenca i caratteri utilizzati per separare le stringhe lette da input (vedere comando read della shell)

• Scrittura :

echo : opt -n: stampa senza "\n", -e: interpretare i ruote char

printf : come in C

• Espr. aritmetiche : uso **let**, deprecate ((...)), [...]

• if [cond] then
else
fi

→ posso scrivere utilizzando quei paragoni :

if [param1 op param2]

• for () : for var in list; do
...
done

• while () : while [cond]
do
...
done

• " " : usato per istruzioni multiple o cond. = "true" (es. while [:])

• Vars : → 3 nuoveくれる associazioni

richiamabile come in C

\${vett[i]} : riferimento a vett[i]

\${#vett[*]} : n° di elementi nello vett.

\${vett[*]} : riferimento a tutti gli elementi di vett. **\${#vett[i]}** : lunghezza dell'elem. vett[i]

• ELIMINAZIONE di VETTORE / VETT[i] : **unset vett/vett[i]**

• REDIREZIONE I/O : [https://www.andreaminini.com/linux/redirezione-su-linux#:~:text=La%20redirezione%20%C3%A8%20una%20funzione,un%20file%20\(%20standard%20input%20\).](https://www.andreaminini.com/linux/redirezione-su-linux#:~:text=La%20redirezione%20%C3%A8%20una%20funzione,un%20file%20(%20standard%20input%20).)

• STDOUT : fd = 1 (es. IN & xshell redir/scrive)

→ [command] > file.txt : 0 osc corrisponde al file
 >> : " ", 0 apre un file

• STDIN : fd = 0

→ [comand] < file.txt : I passo da file.txt

• STDEERR : fd = 2

I
↑
O
↑

es. sort<fin>fout : il comando di sort viene applicato all'I da fin, una volta eseguito il comando, O messo su sfout

Operatori per numeri

-eq	==
-ne	!=
-gt	>
-ge	>=
-lt	<
-le	<=
!	! (not)

Operatori per file e direttori

-d	L'argomento è una directory
-f	L'argomento è una file regolare
-e	L'argomento esiste
-r	L'argomento ha il permesso di lettura
-w	L'argomento ha il permesso di scrittura
-x	L'argomento ha il permesso di esecuzione
-s	L'argomento ha dimensione non nulla

Operatori logici

=	strcmp
!=	!strcmp
-n string	non NULL string
-z string	NULL (empty) string

!	NOT (in condizione singola)
-a	AND (in condizione singola)
-o	OR (in condizione singola)
&&	AND (in un elenco di condizioni)
	OR (in un elenco di condizioni)

U08 - SINCRONIZZAZIONE:

S01 - SEZIONI CRITICHE:

• PROBLEMI PRRR. PARALLELI ($T \circ P$):

- NECESSITA DI FAMPIRE I DATI CONDIVISI

- CODICE CRITICO: RISULTATO DIPENDE DALL'ORDINE DI EXEC

- MANY DI CODICE NON RIENTRANTI \Rightarrow NON INTEROMPIBILI

\rightarrow NECESSITA DI SYNC TRA P e T

• SC (SEZIONE CRITICA):

È UNA PORZIONE DI CODICE IN CUI PIÙ P_i CONDIVIDONO

PER L'USO DI RISORSE COMUNI (es. VARIABILI CONDIVISE)

• \forall SC: \exists UNA SEZ. DI INGRESSO, \exists UNA SEZ. DI USCITA

• CONCET. PER LA DISOLUZIONE DI UN \forall SC:

- MUTUA ESCLUSIONE: $\exists! P/T$ PUÒ ACCEDERE ALLO SC UNA VOLTA

\wedge IN PRENOTAZIONE

- PROCESSO: P/T PUÒ ENTRARE IN SC SE NESSUN ALTRO

\rightarrow EVITARE DEADLOCK
 P/T SI TROVA IN SC

- ATTESA DEFINITA: ACCADE ENTRARE STARVATION DI P/T

- SOLZ. SIMMETRICA: SEZ. DI ACCESSO ALLO SC \Leftrightarrow PRIORITÀ o VECCHIAZIA RELATIVA DI P/T

S02 - SOLUZIONI SW:

• BASATE SULL'UTILIZZO DI DUE VAR. GLOGLI

NO ME

NON ATOMAR
OPERE ISOL.

• SOL1: VET. flag[2] = {FALSE, FALSE} \rightarrow NO : $\left\{ \begin{array}{l} \text{NON CONFERMA MUTUA ESCLUSIIONE} \\ \text{VAR. DI LOCK HA STATO 0+2=1512. A SE STAM} \\ \text{ATTESA ATTIVA} \rightarrow SERVIZIO CPU \end{array} \right.$

\wedge TIPO "PRENOTAZIONE"

SPIN LOCK \wedge STARV

• SOL2: SOL1 / scenario TEST + SET rec PURG \rightarrow NO : P_i e P_j POSSIBILITÀ DEADLOCK

• SOL3: USO VAR. GLOGLIE int turn = i / j \rightarrow NO \rightarrow NO PROB.

ATTESA NON DEFINITA
 \rightarrow POSSIBILITÀ DI STARVATION PER P_i (P_j)
se P_3 (P_1) NON INGRESSA ALLO SC

• SOL4: SOL1 + SOL2 + SOL3

\wedge PEPPERSON: MI PREZERO, MA DO IL TURNO

\rightarrow S1: RISOLVE TURN (0 O 1) IN PARALELLI

```
while (TRUE) {
    flag[i] = TRUE;
    turn = j;
    while (flag[j] & turn==j);
    SC
    flag[i] = FALSE;
    sezione non critica
}
```

```
while (TRUE) {
    flag[j] = TRUE;
    turn = i;
    while (flag[i] & turn==i);
    SC
    flag[j] = FALSE;
    sezione non critica
}
```

• PROGRAM SOL. SW:

- ASSEGNAZIONE / CONTROLLO DI UNA

VAR. DA POSS DI P/T È "INIZIALE" AL SCRIVI P/T

- OP. DI CONTROLLO / MODIFICA NON SONO ATOMICHE \rightarrow POSSIBILITÀ DI READING FROM VAR. POSSIBILE E NON REALE

- COMPRESSITÀ GENETIQUE

\rightarrow ME \rightarrow TURN
PROCESSO $i \rightarrow$ flag[j] = FALSE
STANZA \rightarrow DATA SC \rightarrow flag[i] = FALSE
SIMMETRIA \vee

SD3 - SOLUZIONI HW:

SOLUZIONI:

• NO DIRITTO PREEMZIONE: i P/T in exec non possono essere nel interrupt oppure CPU

• INAFFIDABILITÀ, CONTENUTO VOLONTARIO DEL RELEASE PER P/T

• SI DIRITTO PREEMZIONE:

○ POSSIBILITÀ DI INTERRUZIONE DAL SO TRAMITE INTERRUPT

SOLUZIONE IN SIST. 1-processori: INTERRUPT OFF \rightarrow SC \rightarrow INTERRUPT ON (NEL CODICE)

L, S: • PROCEDURA INSICURA: Puttano diritto su INTERRUPT && Puttano finta. SUPERATO?

• IN SIST. N-processori: necessari diversi INTERRUPT & PROCESSORI

\rightarrow f. LUNGO PROCESSAMENTO, LEGGERE SIST. NON REAL-TIME

• ACTERNAMENT: ESTENDERE LE SOL. SN:

• USO LOCK DI PROTEZIONE & SC

• ISTRUZIONE ATOMICHE PER MANIPOLAZIONE LOCK \rightarrow EXEC IN 1 UNICO MEMORY CYCLE

• MECCANISMI DI LOCK/UNLOCK:

• TEST AND SET:

\rightarrow GET e RETURN di VAR. VAR. WORKER

\rightarrow AGIRE IN MANIERA ATOMICA \rightarrow 1 SOLO CICLO

S: • TEST AND SET() NON DEVE ESSERE INTERRUZIONE

• UTILIZZO CPU SU WHILE()

• SWAP:

\rightarrow USO DI 2 VAR: { \rightsquigarrow VAR. WORKER
key \rightarrow PROTEZIONE SC
lock \rightarrow SC LOCKED/UNLOCKED
 \rightsquigarrow VAR. WORKER
BANDO È SETTATO

• SE lock == FALSE: SC c'è \rightarrow ACCEDO
swap() \rightarrow key = TRUE, lock = FALSE

S: (ANALOG. TEST AND SET)

• ENTRARE LE SEC. NON ASSICURANO "NON STARVATION":

L, SOLUZ.: BURNS (1979)

• CONCLUSIONI SOL. HW:

V: UTILIZZABILE IN MULTI-PROCESSORI ESTENSIONE N P;
SERVIZIO UTILIZZATO PER V, SIMMETRIA

S: DIFFICOLTÀ IMPL. IN HW, STARVATION (es. 2nd. del P; IN STIMA ASSISTITA DA P; STESSI, NON SO)

- INIEZIONE DI PAROLE

```
char TestAndSet (char *lock) {
    char lock = FALSE;
    char val;
    val = *lock;
    *lock = TRUE; // Set new lock
    return val; // Return old lock
}
```

```
while (TRUE) {
    while (TestAndSet (&lock)); // lock
    SC
    lock = FALSE; // unlock
    sezione non critica
}
```

\rightsquigarrow INIZIALIZZAZIONE
char lock = FALSE;

```
while (TRUE) {
    key = TRUE;
    while (key==TRUE)
        swap (&lock, &key); // Lock
    SC
    lock = FALSE; // Unlock
    sezione non critica
}
```

.BURNS:

```
while (TRUE) {
    waiting[i] = TRUE;
    while (waiting[i] && TestAndSet (&lock));
    SC
    j = (i+1) % N;
    while ((j+1) % N; and (waiting[j]==FALSE))
        j = (j+1) % N;
    if (j==i)
        lock = FALSE;
    else
        waiting[j] = FALSE;
    sezione non critica
}
```

Un vettore di attesa, i.e., prenotazione, con un elemento per ogni P/T, inizializzato a FALSE

Pi

Lock globale, unico, inizializzato a FALSE

I P/T in coda entrano in SC perché ricevono il testimone dal precedente

• S04 - SOLUZIONI AD NOC, SEMAFORI:

- SEMAFORO: PRIMITIVE FORNITE DAI SD (DIJKSTRA, 1965) \rightarrow NO SPEDO CPU
 - \rightarrow S È: VAR. INTESA CONDIVISA PROTETTA DAL SO
 - OP. SU S EXEC IN MODO ATOMICO

• PRIMITIVE:

- **init(S, k)**: INIZIALIZZA S, CON VALORE K
- **wait(S)**: PERMETTE DI OBTENERE L'ACCESSO RISERVA SC PROTETTA DA S

NON È CONC
LA wait()
DEI PROCESSI FILII

- SE $S < 0$: BLOCCA P; CONTINUAMENTE \rightarrow RISORSA NON DISP.

\hookrightarrow IN QUESTO CASO, SI INDIRIZZA N° P; IN ATTESA

- **signal(S)**: INCREMENTA S ($S++$)

- SE S ERA < 0 : SERVONO L'ACCESSO PER P/T ONE VOLTA ACCESO

- **destroy(S)**: ESEGUE LA FREE DI S

• S REGOLE:

- $\exists S < 0 \rightarrow |S|: n^{\circ}$ P IN ATTESA

- UTILIZZO UNA SIMILARE A FIFO/LIFO (DIPENDE DAL SISTEMA)

PRIORITÀ P IN ATTESA

• S TRAMITE PIPE:

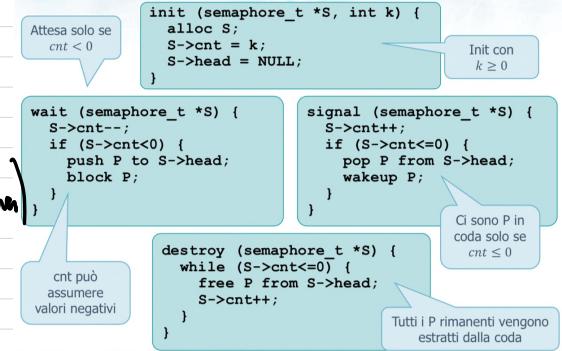
UTILIZZO DI `write()` E `read()` TRAMITE IL CONCETTO DI TOKEN, SU UNA PIPE

LA `wait()` READS (P) IN ATTESA
NON IMPLOMECHANISM OF BUSY WAITING

```
wait (S) {
    while (S<=0);
    S--;
}
```

\rightarrow NO BUSY WAITING

IMPLEMENTAZIONE REALE SEMAFORO:



#include <unistd.h>

```
void semaphore_init (int *S) {
    if (pipe (S) == -1) {
        printf ("Error");
        exit (-1);
    }
    return;
}
```

#include <unistd.h>

```
void semaphore_signal (int *S) {
    char ctr = 'X';
    if (write(S[1], &ctr, sizeof(char)) != 1) {
        printf ("Error");
        exit (-1);
    }
    return;
}
```

#include <unistd.h>

```
void semaphore_wait (int *S) {
    char ctr;
    if (read (S[0], &ctr, sizeof(char)) != 1) {
        printf ("Error");
        exit (-1);
    }
    return;
}
```

• S TRAMITE POSIX:

DEFINIMI IN `<semaphore.h>`, `sem_t` S;

```
int sem_init (
    sem_t *sem,
    int pshared,
    unsigned int value
);
```

BLOCCANTE: `int sem_wait (sem_t *sem)`
BLOCCA IL CHIAMANTE FINO A CHE NON È 0 O SOLO 0;
`int sem_trywait (sem_t *sem)`

\rightarrow ANALOG. `signal(S)`
 \rightarrow S++
`int sem_post (sem_t *sem)`
SE $S=0$: RETURN -> ELSE $S--$

```
int sem_getvalue (
    sem_t *sem,
    int *valP
);
```

RETURNA ->
SE S È VERSO DA
UN ALTRO P

```
int sem_destroy (
    sem_t *sem
);
```

$/pshared: = 0 \rightarrow$ S LOCAL

$\neq 0 \rightarrow$ S CONDIVISO

• S CON MUTEX (BINARIO):

`pthread_mutex_t S;` $(\cancel{()})$ SIMILAR, POSIX

\hookrightarrow SHARE S1 ÷ S2

SOS - PROBLEMI TIPICI :

• PRODUTTORI e CONSUMATORI :

1 Produttore
1 Consumatore

Invece di n utilizza
Elementi pieni
Elementi vuoti

init (full, 0);
init (empty, SIZE);

```
Producer () {
    int val;
    while (TRUE) {
        produce (&val);
        wait (empty);
        enqueue (val);
        signal (full);
    }
}
```

```
Consumer () {
    int val;
    while (TRUE) {
        wait (full);
        dequeue (&val);
        signal (empty);
        consume (val);
    }
}
```

P Produttori
C Consumatori

Occorre forzare ME
tra P e tra C

init (full, 0);
init (empty, SIZE);
init (Mfp, 1);
init (Mfc, 1);

```
Producer () {
    int val;
    while (TRUE) {
        produce (&val);
        wait (empty);
        enqueue (val);
        signal (Mfp);
        signal (full);
    }
}
```

```
Consumer () {
    int val;
    while (TRUE) {
        wait (full);
        wait (Mfc);
        dequeue (&val);
        signal (Mfc);
        signal (empty);
        consume (val);
    }
}
```

READERS & WRITERS

• PRECEDENZA AI LETTORI :

Reader

1 R una volta
Piu' scrittura obbliga
SEZIONE (meR)

```
wait (meR);
nr++;
if (nr==1) 1 R scrittura wait (w);
signal (meR);
...
lettura
...
wait (meR);
nr--;
if (nr==0)
    signal (w);
signal (meR);
```

nR = 0;
init (meR, 1);
init (w, 1);

Writer

```
wait (w);
...
scrittura
...
signal (w);
```

• PRECEDENZA AI SCRITTORI :

Reader

nR = nW = 0;
init (w, 1); init (r, 1);
init (meR, 1); init (meW, 1);

```
wait (r);
wait (meR);
nr++;
if (nr == 1)
    wait (w);
signal (meR);
signal (r);
...
lettura
...
wait (meR);
nr--;
if (nr == 0)
    signal (w);
signal (meR);
```

Writer

```
wait (meW);
nw++;
if (nw == 1)
    wait (r);
signal (meW);
wait (w);
...
scrittura
...
signal (w);
wait (meW);
nw--;
if (nw == 0)
    signal (r);
signal (meW);
```

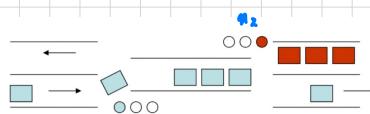
• TUNNEL A SENSO ACERCIATO :

n1 = n2 = 0;
init (s1, 1); init (s2, 1);
init (busy, 1);

```
left2right
wait (s1);
n1++;
if (n1 == 1)
    wait (busy);
signal (s1);
...
Run (left to right)
...
wait (s1);
n1--;
if (n1 == 0)
    signal (busy);
signal (s1);
```

```
right2left
wait (s2);
n2++;
if (n2 == 1)
    wait (busy);
signal (s2);
...
Run (right to left)
...
wait (s2);
n2--;
if (n2 == 0)
    signal (busy);
signal (s2);
```

$\rightsquigarrow S_1$ priorità $\rightsquigarrow S_2$ $\rightsquigarrow M_1$ $\rightsquigarrow M_2$



5 SEM \rightarrow 1 F semaforo

• PROBLEMI S FILOSOFI :

❖ Struttura dati

- Uno stato per ogni filosofo (THINKING, HUNGRY, EATING)
- Un semaforo per ogni filosofo (per l'accesso al cibo)
- Un semaforo ulteriore unico per l'accesso alla variabile di stato del filosofo stesso

```
while (TRUE) {
    think ();
    takeForks (i);
    eat ();
    putForks (i);
}
```

int state[N] = THINKING, HUNGRY, EATING;
init (mutex, 1);
init (sem[0], 0); ...; init (sem[4], 0);

```
takeForks (int i) {
    wait (mutex);
    state[i] = HUNGRY;
    test (i);
    signal (mutex);
    wait (sem[i]);
}
```

```
putForks (int i) {
    wait (mutex);
    state[i] = THINKING;
    test (LEFT);
    test (RIGHT);
    signal (mutex);
}
```

```
test (int i) {
    if (state[i]==HUNGRY && state[LEFT]!=EATING &&
        state[RIGHT]!=EATING) {
        state[i] = EATING;
        signal (sem[i]);
    }
}
```

• VOY - LO SCHEDULING :

- S01: \hookrightarrow QUESTIONE DEI PROCESSI IN ATTESA DI ESECUZIONE

• LO SCHEDULER DECIDE DI quale ALGORITMO UTILIZZARE:
PER LO SVOLGIMENTO DI UN M/M-M/RR/R/P
TRAMITE DIVERSE FUNZIONI DI COSTO
 \hookrightarrow TRASMETTERE UTILIZZO CPU

SENZA PRELACIONE: [CPU NON PUÒ ESSERE SOTTOPOSTA A UN TASK, T REVIE RILASSATA]
CON PRELACIONE: [CPU PUÒ ESSERE SOTTOPOSTA A TASK, PERMETTENDO CPU BURST: t_{max} DI ESSERE VOLONTARIAMENTE

- ALGORITMI

• FCFS (FIRST COME FIRST SERVED): $\hookrightarrow t_{attesa} = t_{start,T} - t_{arrivo}$

CPU ASSOCIAZIONE AI TASK IN BASE ALL'ORDINE D'ARRIVO \rightarrow CODA FIFO

• NO PRELACIONE

V: FACILITÀ DI CONVERGENZA E IMPLEMENTAZIONE

S: t_{attesa} (non ottimale), NO PRELACIONE, ATTESA UNICA SOGLI TASK NEGLI IN CODA

- SJF (SHORTEST JOB FIRST):

T SCHEDULATI IN ORDINE DI BURST-TIME (FCFS IN CASO DI PARITÀ)

V: ACC. OTTIMO (USANDO IL CONTRASTO DI t_{attesa})

S: POSSIBILE STARVATION, DIFFICILE ADEGUAMENTO (NON SEMPRE SI CONOSCE IL t DI BURST SUCCESSIVO)

\rightarrow PER STIMA t_{n+1} -ESIMO BURST, USO MEDIA-ESPONENZIALE:

$$\tilde{t}_{n+1} = \alpha \cdot t_n + (1-\alpha) \cdot \tilde{t}_n \quad / \quad t_n: t_{n-th} \text{ BURST}, \tilde{t}_n: t_{stima} n, \alpha: PESO ESponentiale$$

$\rightarrow \uparrow \alpha \rightarrow \downarrow$ PESO DEI TERMINI

$$\tau_{n+1} = \alpha \cdot t_n + (1-\alpha) \cdot \alpha \cdot t_{n-1} + \dots + (1-\alpha)^j \cdot \alpha \cdot t_{n-j} + \dots + (1-\alpha)^{n+1} \cdot \tau_0$$

- PS (PRIORITY SCHEDULING):

$\forall T \rightarrow$ ASSOCIA UNA PREFERENZA p , CPU ASSOCIA A T CON p_{max}

$\rightarrow PS = SJF \quad | \quad$ CPU BURST = p ASSOCIA \rightarrow NON CONTA t_{burst} DI T

$$\begin{aligned} \tilde{t}_3 &= \alpha \cdot t_2 + (1-\alpha) \cdot \tilde{t}_2 = \\ &= \alpha \cdot t_2 + (1-\alpha) \left(\alpha \cdot t_1 + (1-\alpha) \cdot \tilde{t}_1 \right) = \\ &= \alpha \cdot t_2 + (1-\alpha) \cdot \alpha \cdot t_1 + (1-\alpha)^2 \cdot \tilde{t}_1 = \frac{T_0}{\tilde{t}_1} \end{aligned}$$

S: POSSIBILITÀ STARVATION (ZONA p PREFERITO NON ESSERE MA ESTERNA)

\hookrightarrow SOL: AGING DEI TASK $\rightarrow \uparrow t \rightarrow \uparrow p_T$

- RR (ROUND ROBIN):

FCFS CON PRELACIONE: UTILIZZO CPU DIVISO IN PORZIONI TEMPORALI DI DURATA T_0

$\rightarrow \forall T$ VENGONO ESEGUITE FINO A UNA DURATA T_0 , Poi REINGRESSO IN CODA FIFO SENZA CONTINUARLO

S: t_{attesa} NELLA RELAZIONE LINEARE, DIPENDENZA FORTE DELLE PREFERENZE DATA DURATA DI T_0

- SRTF (SHORTEST-REMAINING-TIME FIRST):

SJF CON PRELACIONE: SE ARRIVA T_i CON BURST PIÙ BREVE, VENGONO PASSATI, T_j VENGONO POLTO

V: T COMUNI VENGONO PASSATI

S: ALTRIMENTI STIME ACCUMULATE DI t_{esec} , STARVATION POSSIBILE

Algoritmi senza prelazione

FCFS (First Come First Served)
Scheduling in ordine di arrivo

SJF (Shortest Job First)
Scheduling per brevità

PS (Priority Scheduling)
Scheduling per priorità

MQS (Multilevel Queue Scheduling) Scheduling a code multi-livello

Algoritmi con prelazione

RR (Round Robin)
Scheduling circolare

SRTF (Shortest Remaining Time First)
Scheduling per tempo rimanente minimo

MQS (MULTIUSER QUEUE SCHEDULING): \rightarrow APPLICABILE ONDE I TASK POSSANO CONDIVIDERE CICLISMO IN VOLTI

REPOR EVENTI SUOGLI IN DIVERSI CODE, OWNED CON IL PROSSIMO ACCADEMICO

\rightarrow POSSIBILITÀ DI TRASFERIMENTO GI T TRA LE VIE CODE

- V10 - STALLO :

• SO1 - DEADLOCK:

\cdot P/T ENTRA IN UN STATO DI ATTESA CHE NON TERMINA, DATO CHE IN VOLTALE A P/T ATTENDE UN EVENTO CAUSATO SOLO DA P/T STESO INSISTE

• MODELLAZIONE DI DEADLOCK:

$$G = (V, E) \text{ GRADIENTE, DONDE}$$

- $V: P \in \{P_1, \dots, P_n\}$ PROCESSI $\vee R \in \{R_1, \dots, R_m\}$ RISORSE DI SISTEMA / R; m W; ISTANZE
- $E: P_i \rightarrow R_j$: RICHIESTA RISORSA $R_i \rightarrow P_j$: ASSEGNAZIONE RISORSA
- GRAFO DI ATTESA: CONSIDERATO G / AR ; INDICATO DA LINEA TRATTINATA
- GRAFO DI RIVANGOLAZIONE: $P_i \rightarrow \dots \rightarrow R_j \rightarrow \dots \rightarrow P_i$ FUTURA RICHIESTA A P_i PER R_j

• TECNICHE GESTIONE A POSTERIORI:

• RICHIARIZIONE:

VERIFICA LA PRESENZA DI CICLI: \rightarrow SE \exists CICLI \rightarrow NO DEADLOCK

• RIPRISTINO:

Strategia	Descrizione
Terminare tutti i processi in stallo	<ul style="list-style-type: none"> Complessità: semplice causare inconsistenze sulle basi dati Costo: molto più alto di quanto potrebbe essere strettamente necessario
Terminare un processo alla volta tra quelli in stallo	<ul style="list-style-type: none"> Complessità: alta in quanto occorre selezionare l'ordine delle vittime con criteri oggettivi (priorità, tempo di esecuzione effettuato e da effettuare, numero risorse possedute, etc.) Costo: elevato, i.e., dopo ogni terminazione occorre rivedificare la condizione di stallo
Prelimolare le risorse a un processo alla volta	<ul style="list-style-type: none"> Complessità: occorre effettuare il rollback, i.e., fare ritornare il processo vittima a uno stato sicuro Costo: la selezione di una vittima deve minimizzare il costo della prelazione

\rightarrow RICHIARIZIONE + RIPRISTINO SONO OPERAZIONI COMPLESSE LUCUOLANTE + DURATIVE TEMPORALMENTE

\rightarrow SE P RICHIESTE MOLTE RISORSE \rightarrow POSSIBILE STARVATION

• SO2 - PREVENIRE:

SI CERCA DI CONTINUARE LA MADRIGA DI RICHIESTA DELLE RISORSE, VERIFICANDO CHE ABBIANO UNA DENGUE

4 CONDIZIONI NON SI VERIFICHI

• MUTUA ESCLUSIONE:

• POSSESSO + ATTEGA:

• IMPOSSIBILITÀ DI PRELAZIONE:

• ATTESA CIRCOLARE:

Un stallo si verifica a causa di un "possesso e attesa" quando un P possiede una o più risorse e ne chiede di ulteriori

Quindi una condizione di possesso e attesa potrebbe essere evitata imponendo che un P chieda risorse solo se non ne possiede altre

Request All First (RAF)

I P devono acquisire tutte le risorse possedute dal processo stesso

Scarsa utilizzo delle risorse

Risorse eventualmente assegnate molto prima di essere utilizzate

Prima di ogni nuova richiesta ogni P deve rilasciare le risorse già possedute

• Possibilità di starvation

P che richiedono molte risorse molto utilizzate possono dover "ricominciare" molto spesso

Un stallo si verifica a causa dell'"impossibilità di prelazione" quando una risorsa non può essere sottratta a un P

In generale è complesso sottrarre risorse a altri processi in esecuzione

Però si potrebbe ottenere un effetto simile

Se un processo ha già acquisito risorse, ne sottrae un'altra che non può essere allocata immediatamente, è costretto a rilasciare tutte le risorse mantenute (prelazione)

Le risorse rilasciate appartengono alla lista delle risorse che il processo attende di acquisire

Il processo sarà svegliato solo quando potrà recuperare le risorse che aveva già acquisito e quelle nuove

che richiede

Se la nuova richiesta non è disponibile si verifica

Se il processo non possiede la possibilità

Se il processo ha la possibilità di averla

• S03 - GUITARE:

• TECNICHE CHE FORZANO P_i A FORNIRE INFO SU LE RICHIESTE CHE EFFETTUAVANO

• STATO SICURO:

IL SIST. È IN STATO σ_1 : AZIONE NOME DI R_i AI P_i , IMPARE STATO, TRAMITE SEQ. SICURA

• SEQ. SICURA: $P_1, \dots, P_n / R_i$ DI P_i SONO DISPONIBILI TRAMITE LE RISORSE DISPONIBILI E AVRAI P_j

\rightarrow bisogna cercare di mantenere IL SISTEMA IN UNO STATO SICURO

\hookrightarrow NUOVA R_i ATTACATA (\hookrightarrow ASSUNTO) SÌ. IN UNO STATO NUOVO, SONO ATTIVI

• ALGORITMO PER ISTANZE UNITARIE:

IMPLEMENTA UN GRAPPO DI RIFERIMENTO: CONSIDERA TUTTI LE POSSIBILI R_i ,
ANCHE NUOVE FUTURE (...)

\rightarrow PER GARANTIRE UNO STATO SICURO:

SE $\exists \hookrightarrow$ INTROME CICLO \rightarrow ASSEGNAZIONE
RISONDANTE



• ALLOCATORE PER ISTANZE MULTIPLE:

VANTAGGIO DI STATO DEL SIST. PER VANTARE SE R_i SONO SUFFICIENTI PER TUTTI I P_j

\rightarrow NECESSITÀ DI DICHIARE I PRIMI IL N° MAX USO DI RISORSE

\rightarrow POSSIBILITÀ DI SCOPRIRE R_i , GARANTIRE CHE R_i RISOLVIA IN FINITO

• ALGORITMO DEL BANCHIERE (DIJKSTRA, 1965):

$\rightsquigarrow O(n \cdot n^2)$

IS

S; - POCO DIFENSIVO 1. VERIFICO CHE UNO STATO SIA SICURO

- $\downarrow R_i, P_i$ 2. \hookrightarrow CHE UNA NUOVA RICHIESTA POSSA ESSERE SODDISFACTA RISPARMIANDO IN UNO STATO SICURO

\rightarrow

• $P_r \in P_1, \dots, P_m \rightsquigarrow$ PROCESSI

• $R_c \in R_1, \dots, R_m \rightsquigarrow$ RISORSE

\rightarrow

$P_1, P_2, \dots, P_n, \dots, P_m$

R_1

R_2

\vdots

R_c

\vdots

R_m

1) CALCOLO NECESSITÀ = MASSIMO - ASSEGNAME

2) SEGUO UNA SEQ. SICURA: SE MUO DISPONIBILE (P_j), QUAI NECESSITÀ (P_i)

POSSO SONO SPARIRE? (cioè VERIFICA CHE DISPONIBILE (P_j) - NECESSITÀ (P_i) > 0)

\rightarrow SUGGERO P_i \rightarrow DISPONIBILE += ASSEGNAME (P_i), FINE (P_i) = T

• RICHIESTA P_k SODDISF.? VERIFICO $(R_k < NECESSITÀ(P_k)) \wedge (R_k < DISPONIBILE(P_k)) \rightsquigarrow ASSEGNAME(P_k) += R_k$

\rightsquigarrow PK

Nome	Dim.	Contenuto e significato
Fine	[n]	Fine[r]=false indica che P_r non ha terminato
Assegnate	[n][m]	Assegnate[r][c]=k P_r possiede k istanze di R_c
Massimo	[n][m]	Massimo[r][c]=k P_r può richiedere al massimo k istanze di R_c
Necessità	[n][m]	Necessità[r][c]=k P_r ha bisogno di altre k istanze di R_c $\forall i \forall j \text{ Necessità}[i][j] = \text{Massimo}[i][j] - \text{Assegnate}[i][j]$
Disponibili	[m]	Disponibili[c]=k disponibilità pari a k per R_c