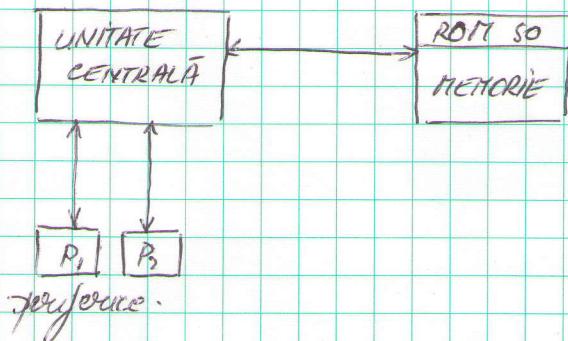


NIVELE DE ORGANIZARE A CALCULATORULUI

- ① FIZIC
- ② MICROPROGRAMAT
- ③ LIMBAJ DE ASAMBLARE
- ④ SISTEM DE OPERARE
- ⑤ APlicații

Nivelul fizic este organizat după schema lui von Neumann.



Regeștiere: SP, PC ..

Salturi JMP, CALL, RET

```

CALL adr;
adr : c1
c2
...
RET
  
```

Orice procesor poate să trateze în nivale de intrerupere 0, 1, ..., n-1

Fiecare intrerupere are asociată o rutină handler.

INTRERUPERI

Orică procesor tratează în nivele de Interrupere 0, ..., n-1
 Interruperea de nivel 0 are cea mai mare prioritate
 În timpul unei interruperi de nivel k, nu se ia în
 considerare interrupările de nivel k+...

MOV ... *intreruperea scârba corespondentă locației de*
lifii *memorie*
 MOV ...

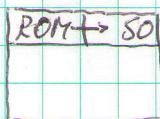
D_i

MOV ... *Interrupările nu sunt luate în considerație.*
 MOV ...

E_i

Pending.

SISTEMUL DE OPERARE



FUNCȚIONALITĂȚILE SISTEMULUI DE OPERARE

1. INTERFAȚA CU UTILIZATORUL

A. LINIE DE COMANDĂ (SHELL)

B. GRAFICĂ TIP WINDOWS

2. GESTIUNEA PERIFERICELOR

3. GESTIUNEA FIȘIERELOR

4. GESTIUNEA PROCESELOR (MULTITASKING)

5. GESTIUNEA UTILIZATORULUI și SECURITATE (MULTIUSER)

6. GESTIUNEA și PROTECȚIA MEMORIEI

7. FACILITĂȚI DE LUCRU ÎN REȚEA

8. APELURI 8787EM

GESTIUNEA PROCESELOR

CARACTERISTICI

1. CONTEXT

Circuite care joacă rolul de periferice: ceasuri

Ceasurile pot fi programate: -periodic 

-one shot 

2. IDENTIFICATOR UNIC

a) IDENTIFICATOR PROCES PÂRINTE (UO - Unix only) PID

3. STAREA PROCESULUII - stare Ready

-stare Running (User mode, Kernel mode)

-stare Sleeping

-Stepped UO

-Zombie UO

4. PROPRIETAR

5. DIRECTORUL CURENT

6. VARIABILE DE MEDIU și VALOREA LOR.

APELURI SISTEM UNIX

- se împart în două categorii :

- care returnează int (în caz de eroare, returnează -1)
- care returnează pointer (în caz de eroare returnează null)

int errno : variabilă globală folosită pt. a depista erorile int.

errno = 0 \Rightarrow succes
 $\neq 0 \Rightarrow$ eroare

void perror (char *c) : afisează mesajul asociat variabilei eroare

INCEPEREA PROCESELOR

- după d.v. al unui programator, procesul începe cu funcția main () .

- main : funcție apelată de modulul start

int main (int argc, char **argv, char **env)

env : environment

- în UNIX main poate avea 3 argumente.

env : lista de seturi de caractere de formă var = val.

ultimul pointer este null.

TERMINAREA PROCESELOR

- orice proces emite un cod de return (intre 0 și 255)

- emisarea unui cod de return se face prin apelul funcției:

void exit (int k);

exit (0); s-a terminat normal

exit (1);

dacă nu se face un exit(0) explicit, funcția care apelează pe main face automat un exit(0)

main apelează pe f, f pe g, g pe h, h pe main
 return h din main = exit(h) decat daca main este pe primul
 nivel de iteratie.

(1 - ~~descriere, scriere si run~~) ~~introducere~~

CREAREA PROCESELOR

int fork() : singurul mod in care programul poate crea procese copii.

a = b;

fork();

c = d;

: creează un nou proces f. asemănător cu procesul deja creat.

procesul execută imediat c = d

executia copilului incepe după fork

codul de return al lui fork: 1 - eroare (nu s-a creat proces copii)

> 0 procesată, pid copii

= 0 proces copii.

int pid;

pid = fork();

if (pid < 0) {

}

/* err */

if (pid > 0) {

}

/* tata */

else {

}

(* pid = 0)

}

(* copii */)

TERMINARE

TERMINARE

Dacă s-a terminat procesul copil, iar procesul tata nu a făcut față, procesul intră în stare zonărie.

st : parametru de ieșire, unde funcția wait scrie felul în care s-a terminat procesul copil.

int * st }
 wait(st) } gresit ⇒ tb. initializat st.

int * st;
 st = malloc(sizeof(int)); } ok, dar daca apelăm malloc, se
 wait(st); } face free st
 ⇒ malloc - funcție costisitoare.

int st;
wait (&st); } ok.

wait (NULL); // ok, cind nu ne intereseaza codul de return

int st;
wait (&st);
WIFEXITED (st); // daca programul s-a terminat cu exit
WEXITSTATUS (st); // - codul de return
if (WIFEXITED (st))
 re = WEXITSTATUS (st);
WIFSIGNALED (st) // macro care testeaza daca programul
// s-a terminat in urma unui semnal
WTERMSIG (st) // cu ce semnal s-a terminat programul

wait : dezavantaje :

- apel blocat, sistemul ramane in sleeping
- daca se creeaza mai multe procese copii, iar noua ne trebuu anume.

int waitpid (int pid, int *st, int opt)
 ↓
 pid -ul asteptat

pid > 0 ⇒ astept dupa un pid anume

pid = -1 ⇒ astept dupa orice proces copil ; exact ca la wait

st : - returneaza pid
- se interpreteaza aproape la fel ca la wait

opt 0 = nicio optiune

opt WNOHANG - apel neblocant
- nu sta sa astepte, se scade automat catre cu
cod de return -1, iar urmatoare va barea EAGAIN

opt WUNTRACED → cod de return > 0 ⇒ pt. procesele stopate

Marea stopped : indiferent de comanda se opreste /imprejurata comanda

WIFSTOPPED (st) - daca a fost stopat

WSTOPSIG (st) - in urma carui semnal a fost stopat.

FAMILIA DE APELURI exec

- are ca prim argument un char * numele fișierului.

- ① int exec (char *exe, char *arg0, char *arg1, ... char *argn, NULL)

ex: ls -l

exec (" /bin /ls ", "ls ", "-l ", NULL);

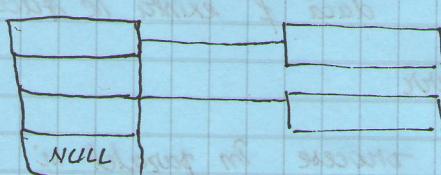
- ② int execvp (char *exe, char *arg0, char *arg1, ..., char *argm, NULL)

Obs: nu mai trebuie să dăruim exe (executabilul) cu cale completă pt. că time cout de variabilele de mediu PATH - lista de directoare cu cale completă separată prin : în UNIX (; pe windows)

PATH = . : \$PATH - în PATH se adaugă directorul curent
export PATH

- ③ int execle (char *exe, char *arg0, ... char *argn, NULL, char ** env)

env - lista de adrese către gruuri de caractere, de cele patru trebuie să arbă sintaxa variab = valoare
ultimul pointer = NULL



- ④ int execv (char *exe, char **argv)

↓ → ultimul pointer NULL

cale completă (cale care nu se termină cu "\0").

- ⑤ int execvp (char *exe, char ** argv)

- ⑥ int execve (char *exe, char ** argv, char ** env)

COMUNICARE INTRE PROCESE

write(a);

read(a);

write(b);

read(b);

1. FISIERE DE TIP SPECIAL

2. IPC SYSTEM V

3. SEMNALE

1. FISIERE DE TIP SPECIAL

- se folosesc două tipuri de fiziere: pipe, socket

- modul de operare exclude automat trei fiziere pe terminal

• stdin → tastatură

• stdout → ecran

• stderr

com < f - stdin nu mai e pe terminal, e pe fisierul f (citește din fisierul f)

com > f - redirectionare stdout (scrise în fisierul f)

com >> f - redirectionare stdout

diferența la > f dacă f există

la >> f dacă f există ne face append

com 2>f - redirectionarea stderr

c1 | c2 | ... | cm - se lansează cele m procese în paralel și

stdout c1 = stdin c2 ...

pe ecran ? lui cm ↓

1 pipe

- comunicări filtru

- cum lucat ? n-1 fiziere de tip special pipe

- pt. comunicare unidirectională între procese



Socket

- pt. comunicare bidirectională

- pt. comunicarea între procese aflate pe margini diferențiate

2. IPC SYSTEM V

- cozi de mesaje (msg), vectori de semafoare (sem), zone de memorie (shm)

- Caracteristici:

- identificatorul extern (*) internum

(*) key_t ftok (char *f, int nr)

figier, care nu să existe

- nu este apel sistem

- rezultaice numele unui identificator extern

msg

Pt fiecare IPC trebuie să avem un apel sistem care să recuperereze identif internum folind de la cel extern

int msgget (key_t cheie, int opt)

-1 în caz de insucces

> 0 în caz de succes : identificatorul internum al cozii de msg.

cheie ftok, IPC_PRIVATE

opt IPC_CREAT | IPC_EXCL | drepturi de acces

int msgsnd (int id, void *p, int lg, int opt)

adresa din memorie unde am construit msg

lungime msg

typedef struct myMsg {

unsigned long tip;

}

opt = IPC_NOWAIT

int msgrev (int id, void *p, int lg, long tip, int opt)

- citeste mesajul pînă la adresa p - mesaj se distruge
(cel mai vîndus mesaj de tipul tip)

tip > 0 cel mai mult mesaj de tipul respectiv

= 0 -- dim coada indiferent de tip

< 0 -- de tipul ? tip

opt : IPC_NOWAIT, MSG_NOERROR

↳ transmită mesaj la lg dacă e prea mare

int msgctl (int id, int oper, struct msqid_cbs *p)

oper - tipul operatiei dorite

IPC_RMID sterge IPC complet din sistem.

preliminare

oaf gaoi hagam

TAIWAN - SAI = 300


```
struct sembuf {
    unsigned short sem-mnum;
    short sem-op;
```

// descrie operatiu asupra unui
semajon atomic.
// index-ul unui reu atomic in vect.
de semajon.

// descrie tipul operatiei: HIGH/LOW
(de mai sus).

sem-op = pozitiv \Rightarrow re ader. cu nr.
de valori = sem-op.
• negativ \Rightarrow decrementare cu un
nr. de valori

(se blocheaza daca ajunge la 0)
(si asteapta)

$\bullet = 0 \Rightarrow$ asteapta ca sem.

să ajungă pe 0

short sem-flg;

// optiuni ale operatiei'

IPC_NOWAIT

SEM_UNDO \Rightarrow la terminarea

procesului, efectul nu
fie reversat.

Apelul operatiei este sau nu

int semop (int id, struct sembuf *tab, int nr)

cel membrat
prin semget

nr. de elem
din tab.

- se poate opera pe mai multe sem. din vect deodata.
- apelul poate rămașe blocat, daca cel putin una din oper. duce in stare blocață.
- returnează -1, daca id-ul e invalid
- cod de return: 0 : toate op. au avut succes.

- apel la nivelul interrupților.

(stop tru, stop tru, stop tru) tempor tri : librat

stop. stop. stop.
stop. stop. stop.
stop. stop. stop.

deosebit de
lipsa stoping
(stop. stop.)

lipsa stoping la initial la -1 - OK după ce librat

. stoping - + -

Apelul

int semctl (int id, int oper, ...)

Operatiile:

IPC-STAT → încarcă obiecturi în structură

IPC-SET → instalează obiecte în structură.

IPC-RMID

GETVAL → arg 3 tb. nu fie iut și nr. numărator.

în caz de succes returnează val. numerică a scu.

SETVAL → modifică val. numărator; arg 3 → indicele sau.

arg 4 → noua val a scu

GETALL → arg 3 : vector de int ; ret. val tuturor scu

SETALL → modif val. tuturor scu. ; arg 3 : vect. de int cu
noile valori.

• utilă, pt. initializarea semajonului.

→ nu se folosește pt. arbitrarea proceselor concurente (vezi semop)

ex: tip de date FILE dim stdio.h e definit printr-un struct,

dar se folosește ca atare.

FILE : struct {

(fbo * fbo) fbo; /* fisiere

char * name; /* nume

char * type; /* tip

(q * 2b - biende totale, 1000 fisiere, bi - fisiere) biende fisiere

TAT3_391

TSZ_391

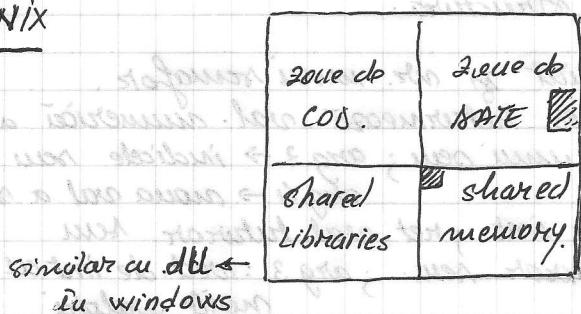
ATMS_391

scrierile este să comunică între ele informații și să interacționeze

3. ZONE DE MEMORIE PARTAJATA

- în zona unui proces nu poate scrie niciun alt proces.

HP-UNIX



pe lângă z.-de date datei procesului de S.O. ne mai poate folosi ap`o zonă în cadrul procesului care nu mai are certitudinea că un alt proces nu o mai folosește.

Folosinu apelul :

- `int shmget (key_t cheie, int nr, int opt)`
→ recuperază id-ul.

- `void *shmat (int id, void *adr, int opt)`
→ operatie prin care se atrelgează.
→ exec : met NULL.

→ optiuni : SHM_RDONLY

se punte NULL, pt. a lăsa S.O. să aleagă adresa.

- `int shmdt (void *adr)`
→ operatia de detasare.
→ adr': adresa recuperată prin shmat

- `int shmctl (int id, int oper, struct shmid_ds *p)`

IPC_STAT

IPC_SET

IPC_RMID.

Concluzie

⇒ instrumente puternice pt. comunicarea între proceze.

SEMNALE

- procesele utilizatorilor normali nu pot trimit semnale decât către procesele lansate tot de ei.

signal.h

NSIG

#define SIGINT 4 // fiecare semnal e identif. printr-un nr.

- pot exista și semnale neportabile către sisteme.
- la primirea unei semnale procesul se comportă ca la primirea unei Interruperi.

handler

- (1) implicite // fiecare semnal are handlerul lui
 - (2) scrisă de programator // există un apel de sistem care ne permite să instalăm handlerul nostru
- majoritatea handlerului nostru se term în cu exit();
 - (2) 1-ar termina prin return

shell

kill pid // trimite către sistem SIGTERM

kill -nr pid

kill -INT pid

numele generic al semnalului din signal.h
fără prefixul SIG, care e obligatoriu

Apelul sistem

int kill (int pid, int sig)

- se folosesc numele generic ale semnalelor.
- exec : -nu există semnalul -
- nu am obținut către procesul.

* Semnale din partea sistemului de operare

■ de la tastatura de control

* SIGINT

la apăsarea tastei Intr (de obicei are val. ^C
Ctrl-C)

* comportament implicit de term procesului:

SIGQUIT

la apăsarea ^\

* SIGSUSP

la apăsarea tastei SUSP, cel mai ușor: ^@

- pună procesul în stare stopată.

* SIGCONT

- are efect de la tast.

- dacă procesul era stopat, îl continuă / trezeste.

SIGKILL

- tot nemodificabil

- sfondul este exit()

(kill - kill sau kill -9)

■ în urma greselilor de programare

SIGSEGV

- încercare de scriere unde nu avem dreptul.

- comportament implicit: corte și exit.

SIGILL

- când programul computer-ului ia o valoare ilegală.

SIGBUS

- pe lângă nu se prevede

- eroare de magistrată.

Paranteza:

Pointeri către funcții

P. normali → constante
P. normali → variabile

P. către funcții

ex: int f (...) {
 ...
}

f (...);

f; // adresa din zona de cod unde se află fct. f.

int (*p)(); // p reprez. un pointer către f.
în acest caz neinitializat.

(*p)(...) // se primește SIGILL dacă p nu e initializat
în cursa programării.

int g(...);

...
}

if (a < b)
 p = f;
else
 p = g;

ex: int qsort(void *tab, int size_element, int nr_elem,
 int (*comp)(void *a, void *b))

// arg 4: pointer către fct, care primește ca arg 2 pointeri

Vectori de pointeri către funcții

ex: int (*tab)()[100] = {f0,..fi,...} // ? sintaxă
 (*tab[i])(...)

↑
oper.