

un semafor interacție se poate

VECTORII DE SEMAFORE

judicat buna

bun în ceea ce se referă la -

o secvență de

(W1) \rightarrow primul proces

write(a);

write(b);

al doilea proces

read(a);

read(b);

* zone critice \leftarrow intreruperile unei proces și continuarea celuilalt
poate produce informații eronate.

Soluție : Semafore, Dijkstra

= un obiect al sist. de op

cu o valoare mai mare sau egală cu 0.

HIGH(s); \rightarrow incrementează

valoarea semaforului

} cu o unitate.

LOW(s); \rightarrow decrementează

în cazul $s=0$ LOW se blochează până când un proces HIGH incrementează s și apoi decrementează.

\rightarrow rezolvă prob. de concurență (de mutualitate)

s = 1

LOW(s);

write(a);

write(b);

HIGH(s)

LOW(s);

read(a);

read(b);

HIGH(s);

\rightarrow cele două procese nu pot acționa în același timp în zonă critică.

Acesta este semafor de tip mutrex. (ia valori de la 1 și 0).

UNIX

\rightarrow implementarea unui semafor nu-uu sens mai larg decât Dijkstra.

\rightarrow prim semafor se intinde ca un vector de semafore (nu un sem. atomic)

Apelul : int semget (key_t cheie, int nr, int opt)

\rightarrow se recuperă
un IPC (general,
private sau
nou creat)

crt. de elem. dir
sec de sem.
are sens cănd creăm
un sem. nou

codul de return > 0 -> -ul intern al obiectului

-1 -> -insuccess.

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

// descrie operatiu asupra unui
semajon atomic.

// index-ul numerelor atomici in vector
de semajon.

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

sem-op = pozitiv \Rightarrow re incrc. cu nr.

de redare = sem-op.

• negativ \Rightarrow decrementare cu nr.
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 operatiilor sem op

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

cel membrat
prin semget

nr. de elem
din tab.

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

- apel la nivelul interrupcilor.

(stop task, stop thread, stop process, tasklet, interrupt)

stop task, stop thread, stop process

tasklet, interrupt

stop process

tasklet - + -

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 iată și nr. numărator.

în caz de succes returnează val. numerică a scui.

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

arg 4 → noua val a scui

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

SETALL → modif val. tuturor scui ; 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

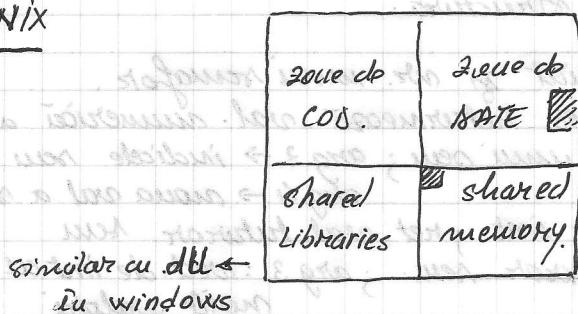
ATMS_391

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 date 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 desasare.
→ 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 procente.

SEMNALE

- procesele utilizatorilor normali nu pot trimit semnale decat catre procesele lansate tot de ei.

signal.h

NSIG

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

- pot exista si semnale reportabile altre sisteme.
- la primirea unei semnale procesul se comporta ca la primirea unei interruperi.

handler

- (1) implicite // fiecare semnal are handlerul lui
 - (2) scris de programator // exista un apel de sistem care ne permite sa instalam handlerul nostru
- majoritatea handlerului nostru se term cu exit();
 - (2) 1-ar termina prim return

shell

kill pid // trimite catre sistem SIGTERM

kill -nr pid

kill -INT pid

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

apelul sistem

int kill (int pid, int sig)

- se folosesc numele generice ale semnalelor.
- exec : -nu exista semnalul -
- nu am obiectua altora procesului.

* 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

- sfondlerul 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.