



PRIPREMA ZA LABOSE → gradivo za liceve?

3. PREKIDNI RAD

3.1. SIGNALI (* PREKIDI)

- 1. Laboratorijska vježba
- na računu OS

- procesi dobivaju signalne od:
 - jezgre OS (kada proces pristupi zadnjemoj lokaciji)
 - samog sebe (signalizacija)
 - drugog procesa
 - konzolice (CTRL-C)

- proces može ali i ne mora reagirati na signal (programisti se definira ponavljanje nakon primanja signala)

⇒ 1. LABOS

- signal ne sadrži nikakvu informaciju

* VJEŽBA 1 → novi učio counter, samo zastavice

- prekide povezivanje signalima
- #include <Signal.h>

* FUNKCIJE

- o sighold (<Signal>) ... zadržava određeni signal
- o sigrelse (<Signal>) ... otpušta signal na otkaštu

- primer:

sighold (sigint);

ctrl - c

sigrelse (sigint);

tu se tek omogućava

- definiraju ponavljanja procesa nakon dohvata metoda signala
- MASKIRANJE SIGNALA ⇒ sigset (<Signal>, <politefna-func>)

- primer: void obrada (int s) // s - kod signala

int main (),

{

 sigset (sigint, obrada); CTRL-C

}

- nakon fje za dohod signal, proces nastavlja izvodenje na mjestu gdje je signal dobiove

- "f" je za dohod signal, istovremeno signal je zadržan do kraja funkcije (tako da ako na početku f je stavljen signal za tog signal)

o sleep (<br sek>)

... čeka zadani broj sekundi ili do primjedbe blokiranog signala; vraca broj neprospravanih sekundi

④ o usleep (<br-fns>)

POKRETANJE VIŠE PROCESA

- dječe dobiva kopiju sekvencije instrukcija : podataka od roditelja
- proces roditelj stvara proces dječeta
- dodaci nisu zajednički
- procesi mogu komunicirati samo uz pomoć mehanizama OS-a

STVARANJE NOVOG PROCESA

int FORK (void)

... Proces dječe dobiva Ø, a roditelj PID dječeta
 ako uspije proces stvaranja novog dječeta,
 ako ne uspije vraca -1

- primjer : int i;

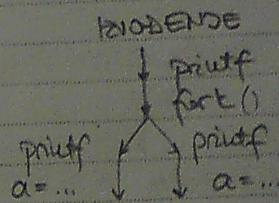
```
....  

printf ("PRIJE");  

fork();  

printf ("POSUDE");  

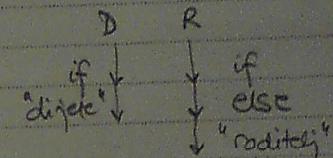
a = 2+3;
```



```
i=fork();  

if(i==0) printf ('dijete')  

else printf ("roditelj");
```



ispis :

PRIJE
POSUDE
POSUDE
DODETE
RODITELJ

ZELANJE PROCESA DJEČETA

- čeka se na kraju izvođenja programa
- int wait (int *STAT USP)

wait (NULL);

- poziva se jednako broj puta koliko ima procesa djece

```
- primjer : #define bpr 3 < void proces (int i);  

int main ()  

{
```

```
    int i;  

    for (i=0; i<bpr; i++)  

        switch (fork()) {  

            case -1 : printf ('...'), exit(1);  

            case Ø : proces (i), exit(Ø); // DODETE  

            default : // RODITELJ  

        }
```

```
    for (i=0; i<bpr; i++)  

        wait (NULL);  

    return 0;
```

PRIPREMA ZA 2. LABORATORIJSKU VJEŽBU

- DRETVA = niz instrukcija u izvođenju
- skup zauzettih sredstava je isti za sve dijelove istog procesa
- postoji zajednička memorija (globalne varijable)
- komunikacija među dretvama je zavato brža od komunikacije među procesorima
- dretve istog procesa komuniciraju bez upitovanja OS-a
- JEDINSTVENO ZA SVAKU DRETVE \Rightarrow ID dretve, stanje registara, stog, privatni prostor
- STVARANJE DRETVE: `pthread_create(...)` return \emptyset OK
return $\neq \emptyset$ GREŠKA
- NIZ ID-OVA DRETVE: `pthread_t ID [br_drs]`
(+p podatka)
- ZAVRŠETAK DRETVE: `pthread_exit(NULL)` u završetak funkcije
- DOČEKIVANJE DRETVE: `pthread_join(ID_dretve, NULL)`
- PREVODENJE PROGRAMA: `gcc -lpthread name.c`

DRETVE

PRIMJER

```

pthread ID [2];
int arg;
pthread_create (&ID[0], NULL, fja_dretve, &arg);
int polje [ ];
pthread_create (&ID[1], NULL, fja_dretve, &polje [0]);
    
```

→ Mjesto gdje se spravljaju ID-ovi
→ mogućnost
 ↓
parametar funkcije fja_drete
 ↓ slaviže više argumentata

```

#define br_dr 3
void *dretva (void *arg)
{
    int i, rbr;
    rbr = *(int *) (arg)
    for (i=1; i<4; i++)
        printf ("Dretva br %d , iteracija %d\n", rbr, i);
}
    
```

```

int main()
{
    int i, arg [br_dr];
    pthread_t [br_dr];
    for (i=0; i<br_dr; i++)
    {
        arg [i] = ++i;
    }
}
    
```

```

if (pthread_create (&ID[i], NULL, dretva, &arg))
    printf ('Greška !');
    exit (1);
}
    
```

različitim dretvama
ne podeljuju
prosječivati
adrese iste
varijable

$\&arg [i]$

```

for (i=0; i<br-dr; i++)
    pthread_join (id[i], NULL);
return 0;
}

```

- proslijedjuje više od 1 argumenta = POSE
 - arg [br-dr][...]
 - globalna varijabla (zajednička)

DEKKEROV ALGORITAM

- opcijski zadatok
- sinkronizacija dreti u procesu (preporučeno dretve)
- pravo, zastavice = globalne varijable
- ipcs → provjera da li način je ostalo zauzete memorije
- ipcrm → brisanje / oslobađanje memorije

VJEŽBA ZA BLIC

1.

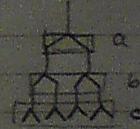
```

fork();
printf("a");
fork();
printf("b");
fork();
printf("c");

```

→ Koliko puta će se ispisati a/b/c ?

- 2/4/6
- 2/4/2
- 2/2/2



2.

```

fork();
fork();
pthread_create(..., ..., f1, ...);
pthread_create(..., ..., f2, ...);

```

→ Koliko je ukupno dreti na kraju odjeća? (1 na početku)

- 4
- 8
- 12

$$1P + 1D \xrightarrow{\text{fork} \times 2} 4P + 4D \xrightarrow{\text{Static P } \approx 2D} 4P + 12D$$

③

1. $(\alpha + \beta)^2 = \alpha^2 + 2\alpha\beta + \beta^2$

2. $(\alpha - \beta)^2 = \alpha^2 - 2\alpha\beta + \beta^2$

3. $\alpha^2 - \beta^2 = (\alpha + \beta)(\alpha - \beta)$

4. $\alpha^3 - \beta^3 = (\alpha - \beta)(\alpha^2 + \alpha\beta + \beta^2)$

5. $\alpha^4 - \beta^4 = (\alpha - \beta)(\alpha^3 + \alpha^2\beta + \alpha\beta^2 + \beta^3)$

6. $\alpha^n - \beta^n = (\alpha - \beta)(\alpha^{n-1} + \alpha^{n-2}\beta + \dots + \beta^{n-1})$

PRIPREMA ZA 3. LABORATORIJSKU VJEŽBU

SEMAFORI

- 2 proizvodča -> 1 potražač

- GLAVNI PROSES :
- dodeli semafore (20secu + 1Bsecu)
 - inicijalizacija semafora (5, 0, PUN = 1)
 - zaustavi zajedničku memoriju
 - inicijalizirati zajedničke varijable
 - postavi funkciju obrade prekida SIGINT - odobrava memorije (upr. obaveštenje korištenja programu)
 - snimni proces (2 proizvodča + 1 potražač)
 - čekaj proces
 - raspodjeli sredstva (semafoni, memorija)

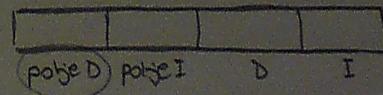
- SEMAFORI :
- PIŠI - Bsecu za zadnje pisanje; prihvatač; PISI = 1;
 - PUN - 0secu; broj punih pretinaca; PUN = 5;
 - PRAZAN - 0secu; broj praznih pretinaca; PRAZAN = 0

- POMOĆNE FUNKCIJE :
- uputa
 - semop ($\langle ID \rangle$, -1) ... čekaj
 - semop ($\langle ID \rangle$, +1) ... postavi
- početna autonomizacija potražača => učitavaanje teksta zaštiti Bsecu;

- ZAJEDNIČKA MEMORIJA :
- 2 integer varijable za pointere
 - polje N[5] za međuspremnik (char)
 - PIMUS : min 128 bojntova; 100 / 128 integera

PRIMER : 1 segment za 2 miza i 2 varijable

```
double *poljeD, *D;  
integer *poljeI, *I;
```



```
int ID = slnuget(...); // prvični ?
```

```
poljeD = (double *) slnmat (ID, ...);
```

```
poljeI = (int *) (poljeD + <veličina-polja>);
```

```
D = (double *) (poljeI + <veličina-polja>);
```

```
I = (int *) (D + <veličina-varijable>);
```

...

```
slnuct ((char *) poljeD);
```

"pointer na početak"

OSLOBĀDANJE SEMAFORA : o semafore () ;

- napisati funkcije za oslobadanje memorije i sredstava opredelito politom konzistentnog signala SINIT
- za reproduciranu izvršetak programa, koristi naredbe IJUSTICE : IPCS, IPCRM
- priporodio su zavrsi : makro drugog lockstasha

ipcs

ipcrm -m <broj>

ipcrm -s <broj>

MONITORI

VARIABLE NEUSOBNOG ISKLJUČIVANJA - djeluje kao bsem

- PRIMER : (MUTEX)
- pthread_mutex_t klijet [n]; ... identifikator
 - pthread_mutex_init (&klijet[1], NULL), ... inicijalizacija
 - ući u monitor pthread_mutex_lock (&klijet[1]); ... čekaj bsem (+)
 - izići iz monitora pthread_mutex_unlock (&klijet[1]); ... postavi bsem (-)

- klijetovi moraju biti globalne varijable - da ih vide sve druge

VIDETNE VARIABLE = oznaka (D) određenog ujetka

- globalne

- ukoliko ujet nije zadovoljen, okreće učinak u red ujetka te istoimenučno oblažćava klijet

cond_wait (<ujet>, <klijet>);

- druga držva može pozvati spajanje ujetka :

pthread_cond_signal (<ujet>) ... oslobođa 1 držvu

pthread_cond_broadcast (<ujet>) ... oslobođa sve druge

- ako se pozove cond_signal (cond_broadcast), a u redu ujetka neuna nitioga, to će značiti sljedeći polaz za idući drživa koji pozove cond_wait

OSTVARENJE MONITORA U LAB. OKRUŽENJU

- ekvivalentno ponavljanje funkciona

- o mutex_lock → uči u monitor
- o mutex_unlock → izadi iz monitora
- o cond_wait → vršiti u red uvjeta + uči u monitor !
- o cond_signal → oslobođi iz reda uvjeta + uči u monitor

↳ fja cond_signal

počinjti kontekst u opšit Activne-D,
 što je (red uvjeta (uvjet) nepromazan)
 ↓ pri opšitu iz reda → Pripravne-D;
 poneti kontekst Aktivne-D,
 obnoviti aktide;
 čekati se iz prethodnog zadaka rada;