Programska podrška mjernih i procesnih sustava

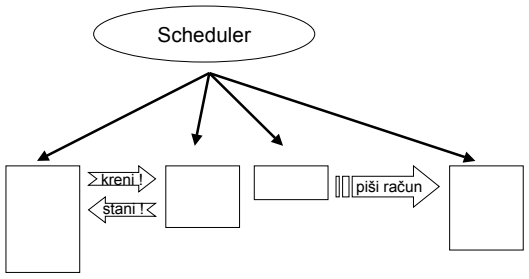
Branko Jeren i Predrag Pale

Fakultet elektrotehnike i računarstva
Zavod za elektroničke sustave i obradbu signala

IPC
Interprocess Communication

komunikacije među procesima:
osnove, problemi i rješenja

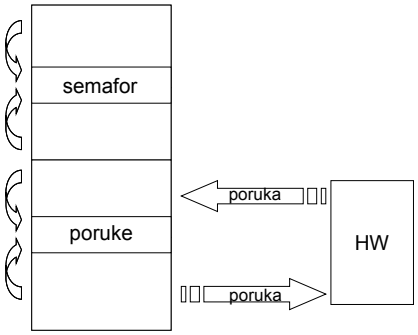
Međuzavisnost i suradnja procesa



Komunikacija među procesima (IPC)

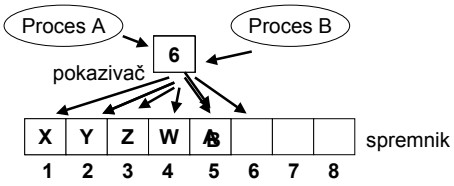
- paralelni procesi su često suradni i/ili zavisni
- potrebe
  - sinkronizacija
  - prijenos poruka
- uvjeti
  - jednoprocesorski sustav
  - višeprocesorski sustav
  - distribuirani procesori
- rješenja
  - dijeljeni medij/memorijski prostor
  - poruke komunikacijskim kanalima

Moguća rješenja



IPC problemi
(dijeljeni medij)

- istovremeni pristup dva ili više procesa jednom dijeljenom mediju
- nastaje “race condition”



LS&S

Slijed stanja u dijeljenoj memoriji (željeni)

```
       PROCES A           PROCES B     pokazivac spremnik
kamo = pokazivac;                            5     [5] = ?
spremnik[kamo] = naziv;                      5     [5] = A
pokazivac += 1;                              6     [6] = ?
.
.
                  kamo = pokazivac;          6     [6] = ?
                  spremnik[kamo] = naziv;    6     [6] = B
                  pokazivac += 1;            7     [7] = ?
```


LS&S

Slijed stanja u dijeljenoj memoriji (race condition)

```
       PROCES A           PROCES B     pokazivac spremnik
kamo = pokazivac;                            5     [5] = ?

                  kamo = pokazivac;          5     [5] = ?
                  spremnik[kamo] = naziv;    5     [5] = B
                  pokazivac += 1;            6     [6] = ?
                  .
spremnik[kamo] = naziv;                      5     [5] = A
pokazivac += 1;                              6
```


LS&S

IPC rješenja ?

- zabrana prekida
  - mora moći raditi korisnik
  - opasno ! jer mogao bi blokirati IRQ zauvijek
- “lock” varijabla
  - ponovo isti problem
- strict alternation
  - beskorisno trošenje CPU
  - problem s procesima različite brzine

= neupotrebljiva

IPC rješenja

- Petersonovo rješenje
  - unapređenje alteracije
  - teoretski upotrebljivo
  - u osnovnoj verziji nije učinkovito
- TSL = Test &Set Lock
  - HW rješenje
  - "atomička" operacija čitanja i odmah pisanja

= tzv. Mutual Exclusion with Busy Waiting

neučinkovito trošenje vremena u petlji čekanja

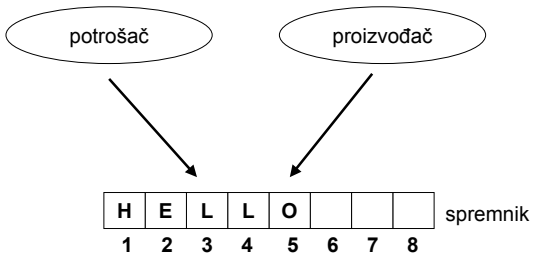
IPC rješenja

- prihvatljiva (i korištena) rješenja
  - sleep & wakeup
  - semafori
  - event counter
  - monitor
  - message passing
- u osnovi semantički jednaka
- svako se može realizirati uz pomoć drugih

Sleep & wakeup

- SLEEP i WAKEUP
  - system calls
- kada proces ne može nastaviti
  - ne čeka u jalovoj petlji (idle loop)
  - već "zaspi", i pusti druge izvoditi
- kad se ispune uvjeti za nastavak
  - ne pokreće ga se odmah
  - već ga se iz "Block" stanja prebaci u "Ready"

Sleep & wakeup:
problem proizvođača i potrošača



Sleep & wakeup:
problem proizvođača i potrošača

```
#define N 8

int count = 0;

void proizvodjac (void) {
int znak;

   while (TRUE) {
        nabavi (&znak);
        if (count == N) sleep();
        stavi (znak);
        count = count + 1;
        if (count == 1) wakeup (potrosac);
   }
}

void potrosac (void) {
int znak;

   while (TRUE) {
        if (count == 0) sleep();
        uzmi (&znak);
        count = count - 1;
        if (count == N - 1) wakeup (potrosac);
        potrosi (znak);
   }
}
```


Sleep & wakeup

- kada proces ne može nastaviti
  - ne čeka u jalovoj petlji (idle loop)
  - već “zaspi”, i pusti druge izvoditi
- kad se ispune uvjeti za nastavak
  - ne pokreće ga se odmah
  - već ga se iz “Block” stanja prebaci u “Ready”
- problem buđenja neusnulog procesa
  - rješenje: semafori
    - sve “wakeup” naredbe se registriraju u semaforu
    - prije “spavanja” proces provjerava semafor

Semafori

- DOWN i UP umjesto SLEEP i WAKEUP
- semafor je brojač čekajućih buđenja
- DOWN
  - napravi SLEEP ako je smeafor 0
  - u protivnom ga samo smanji za jedan
  - sve obavlja kao jednu **atomičku** akciju
- UP
  - povečava semafor za jedan
  - poziva sistem da probudi nekog ako spava na tom semaforu
  - i UP sve obavlja kao jednu **atomičku** akciju

Semafor: problem proizvođača i potrošača

```
#define N 8

typedef int semafor;

semafor rezervacija = 1;
semafor puni = 0;
semafor prazni = N;

void proizvodjac (void) {
int znak;

   while (TRUE) {
        nabavi (&znak);
        down (&prazni);          /* if (count == N) sleep(); */
        down (&rezervacija);     /*                          */
        stavi (znak);
        up (&rezervacija);       /* count = count + 1; */
        up (&puni);              /* if (count == 1) wakeup (potrosac); */
   }
}


void potrosac (void) {
int znak;

   while (TRUE) {
        down (&puni);            /* if (count == 0) sleep(); */
        down (&rezervacija);     /*                          */
        uzmi (&znak);
        up (&rezervacija);       /* count = count - 1; */
        up (&prazni);            /* if (count == N - 1) wakeup (potrosac); */
        potrosi (znak);
   }
}
```


Semafori: vrste

- binarni semafor
  - osigurava uzajamnu isključivost
    - (boravljenja u kritičnoj regiji)
    - “mutex” = mutual exclusion (“rezervacija”)
  - ima vrijednost 1 ili 0
- sinkronizacijski semafor
  - jamče slijed događaja ili brane slijed događaja
    - “puni” i “prazni”
  - ima više mogućih vrijednosti

Monitor

- modul (paket) koju čini nakupina:
  - procedura
  - varijabli
  - podatkovnih struktura
- podaci su dostupni samo procedurama **unutar** monitora
- u jednom trenutku **samo jedan proces** može izvoditi isti monitor
- ostvaruje se u compileru
- rijetka primjena

Message passing (poruke)

- SEND i RECEIVE
  - system calls
  - u načelu zahtjeva HW za prijenos poruka
- jedino rješenje za distribuirane procesore
- radi i na jednoprocesorskim sustavima
  - pomalo neefikasno (u usporedbi sa semaforom)
  - efikasnije su realizacije kroz memoriju
- problemi koje nemaju semafori
  - izgubljene poruke
  - rješenje ? potvrda prijema (“acknowledge”)

Poruke: problem proizvođača i potrošača

```
#define N       8
#define PORLEN 4

typedef int poruka[PORLEN];

void proizvodjac (void) {
int znak;
poruka p;

   while (TRUE) {
        nabavi (&znak);
        receive (potrosac, &p);
        sastavi_poruku (&p, znak);
        send (potrosac, &p);
   }
}


void potrosac (void) {
int znak, i;
poruka p;

   for (i=0; i<N; i++) send (proizvodjac, &p);
   while (TRUE) {
        receive (proizvodjac, &p);
        procitaj_poruku (&p, &znak);
        send (proizvodjac, &p);
        potrosi (znak);
   }
}
```

IPC rješenja

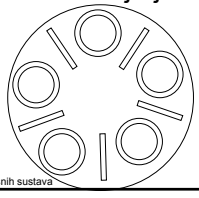
- prihvatljiva (i korištena) rješenja
  - sleep & wakeup
    - bazično i skučeno
  - semafori
    - najčešće korišteno
  - event counter
    - varijanta semafora
  - monitor
    - rijetko se koristi, treba posebni compiler
  - message passing
    - vrlo povoljno za distribuirane procesore
- u osnovi semantički jednaka
- svako se može realizirati uz pomoć drugih

Klasični IPC problemi

- najčešće se koriste za test
- svaki novi IPC mehanizam ih pokušava riješiti
- najpoznatiji
  - "Dining Philosophers" *(Dijkstra, 1965)*
  - "Readers and Writers" *(Curtois et al., 1971)*
  - "Sleeping Barber"

The Dining Philosophers Problem

- filozofi sjede za okruglim stolom
- po jedan štapić
- za rižu su potrebna dva štapića
- malo jedu, a malo razmišljaju
- omogućiti da što učinkovitije jedu



Jednostavno “rješenje”

```
#define N   5

void filozof (int i) {
  while (TRUE) {
      misli ();
      uzmi_stapic(i);
      uzmi_stapic((i+1) % N);
      jedi ();
      vrati_stapic(i);
      vrati_stapic((i+1) % N);
  }
}
```

- **pogrešno !!!**
- nastaje **“deadlock”**
- odustajanje kod zauzeća
  - nastaje “**starvation**”
- slučajno čekanje
  - nepredvidljivo
  - neučinkovito

Pravo rješenje

```
#define N 5
#define LIJEVI      (i-1)%N
#define DESNI       (i+1)%N
#define MISLI       0
#define GLADAN      1
#define JEDE        2

typedef int semafor;
int stanje[N];
semafor rezervacija = 1;
semafor s[N];

void filozof (int i) {
    while (TRUE) {
          misli ();
          uzmi_stapice(i);
          jedi ();
          vrati_stapice(i);
    }
}

void uzmi_stapice (int i) {
    down (&rezervacija);
    stanje [i] = GLADAN;
    test (i);
    up (&rezervacija);
    down (&s[i]);
}


void vrati_stapice (int i) {
    down (&rezervacija);
    stanje [i] = MISLI;
    test (LIJEVI);
    test (DESNI);
    up (&rezervacija);
}

void test (int i) {
    if (stanje[i] == GLADAN && stanje[LIJEVI] != JEDE && stanje[DESNI] != JEDE) {
          stanje[i] = JEDE;
          up (&s[i]);
    }
}
```


Programska podrška mjernih i procesnih sustava

www.zesoi.fer.hr

(http://www.ZESOI.FER.hr/hrzesoi/dodip/ppmips.htm)

ppmps@zesoi.fer.hr