Основи рачунарских система 3

Оперативни системи

Конкуретност и синхронизација

Александар Картељ

aleksandar.kartelj@gmail.com

Природно-математички факултет Бања Лука

Дељени ресурси

- Конкурентно извршавање процеса
  - Паралелно
  - Псеудопаралелно
- Процеси који се конкурентно извршавају
  могу приступати истим ресурсима, нпр. меморији, штампачу итд.
  - Такви ресурси се зову дељени ресурси
- У раду са дељеним ресурсима могу се јавити грешке
  - Пример. Нека је X променљива доступна већем броју процеса
    и нека сваки од тих процеса покушава да уради операцију X=X+1
  - Шта може лоше да се деси?

Дељени ресурси (2)

- Операција X=X+1 није атомична!
  - Атомичност подразумева целовитост у извршавању, тј. ако је операција атомична, онда се она не може половично извршити
- X=X+1 на машинском нивоу се разбија на више операција, нпр.
  1. MUA X - узима променљиву X из меморије и убацује у акумулатор AC
  2. SAB X 1 – додаје вредност 1 на садржај AC
  3. AUM X – враћа садржај AC у меморију на локацију X
- Шта се дешава ако два процеса извршавају ово истовремено, пронађите сценарио у којем долази до проблема?

Проблем произвођача и потрошача

- Два процеса:
  1. Први процес производи ресурс, односно повећава број производа, нпр. пекара у којој пекари производе хлеб и постављају га на рафове
  2. Други процес су потрошачи, у овом случају муштерије које долазе у пекару и купују хлеб
- Написати имплементације произвођача и потрошача?
- Да ли може да дође до проблема са дељеним ресурима?

Проблем произвођача и потрошача (2)

```
main() //algoritam pp1
        brojac=0;
        kapacitet=100;
        proizvodjac();
        potrosac();
```

```
proizvodjac()
    while(true)
        brojac++;
```

```
potrosac()
    while(true)
        brojac--;
```

- Да ли је ово добро решење?

Проблем произвођача и потрошача (3)

```
main() //algoritam pp2
        brojac=0;
        kapacitet=100;
        proizvodjac();
        potrosac();
```

```
proizvodjac()
    while(true)
        while(brojac==kapacitet)
            ; //aktivno čekanje
        brojac++;
```

```
potrosac()
    while(true)
        while(brojac==0)
            ;
        brojac--;
```

- Да ли је ово боље, ако јесте, зашто?
- Које операције у произвођачу и потрошачу су проблематичне?
- Осмислите сценарио у којем долази до неконзистентности

Критична секција

- Критична секција представља део кода у којем може да дође до непожељних резултата
- У раду са дељеним ресурсима, критичне секције су обично места на којима се дешава манипулација дељеним ресурсом
- Да ли је проблем ако више процеса само користи,
  а притом не мења дељени ресурс?
- Шта је критична секција у функцијама произвођач и потрошач?
- Како бисте модификовали претходни код тако да до проблема више не долази?

Заштита критичних секција

- Добро решење треба да има следеће особине:
  1. Узајамна искључивост – два процеса не могу истовремено да буду у критичној секцији
  2. Услов прогреса – процес који није у критичној секцији и не жели да уђе у њу, не треба да омета друге процесе да у њу уђу
  3. Услов коначног чекања – треба да постоји коначна горња граница чекања за улазак у критичну секцију
  4. Праведност – ово није обавезно, али је пожељно својство
- Како да постигнемо узајамну искључивост:
  Како у свакодневном животу штитите ствари, нпр. стан, ауто…

Решење које омогућава узајамну искључивост (A1)?

```
main() //algoritam A1
        slobodno=false;
        proces1();
        proces2();
```

```
proces1()
    while(!slobodno)
        ;
    slobodno = false;
    //Kritična sekcija
    slobodno = true;
```

```
proces2()
    while(!slobodno)
        ;
    slobodno = false;
    //Kritična sekcija
    slobodno = true;
```

- Користимо додатну променљиву као индикатор да ли је неко већ унутра (ово је као некакав катанац, који може бити откључан или закључан)
- Процеси чекају док год катанац не постане откључан
- Чекање је притом активно, процеси „врте празну петљу“!
- Да ли може да се деси проблем у оваквом решењу?

Побољшани алгоритам са гаранцијом узајамне искључивости (A2)

- Код претходног решења је могло да се деси да два процеса уђу:
  - Проблем је код постављања променљиве **slobodno,** јер је и тај део кода К.С
  - Може се десити прекид у незгодном моменту, објасните?
- Ново решење користи две променљиве које најављују да је процес заинтересован за улаз у критичну секцију

```
main() //algoritam A2
        zeli1=false;
        zeli2=false;
        proces1();
        proces2();
```

```
proces1()
    zeli1 = true
    while(zeli2)
        ;
    //Kritična sekcija
    zeli1= false;
```

```
proces2()
    zeli2=true
    while(zeli1)
        ;
    //Kritična sekcija
    zeli2= false;
```

Декеров алгоритам

- Претходни алгоритам А2 је такође имао проблем:
  - Могло се десити заглављивање, чиме би био нарушен услов прогреса
  - Објаснити како би могло да се деси заглављивање у А2?
- Декеров алгоритам решава сва три проблема критичних секција:
  - (1) узајамну искључивост, (2) услов прогреса и (3) услов коначног чекања
- Потребне су три променљиве:
  - zeli1 – први процес најављује да жели да уђе у К.С.
  - zeli2 - други процес најављује да жели да уђе у К.С.
  - na_redu – одређује ко је на реду

Декеров алгоритам (2)

```
proces1()
        zeli1 = true;
        //ako drugi želi
        while(zeli2){
                //i pritom je na redu
                if(na_redu==2){
                    zeli1 = false;  //ne smetamo
                    while(na_redu==2)
                        ;//čekamo ga
                    zeli1 = true; //ponovo tražimo
                }
        }
        //Kritična sekcija
        na_redu=2;
        zeli1= false;
```

```
proces2()
        zeli2 = true;
        //ako drugi želi
        while(zeli1){
                 //i pritom je na redu
                if(na_redu==1){
                            zeli2=false;
                            while(na_redu==1)
                                        ;
                            zeli2=true;
                }
        }
        //Kritična sekcija
        na_redu=1;
        zeli2=false;
```

Питерсонов алгоритам

- Декеров алгоритам је намењен раду са два процеса
  - Не гарантује притом наизменично извршавање односно **праведност**
  - Питерсонов алгоритам је такође за два процеса, али гарантује и праведност
  - Зове се још и „Џентлменски алгоритам“

```
proces1()
        zeli1 = true;
        na_redu=2;
        while(zeli2 and na_redu==2)
          ;//čekamo ga
        //Kritična sekcija
        zeli1= false;
```

```
proces2()
        zeli2 = true;
        na_redu=1;
        while(zeli1 and na_redu==1)
          ;//čekamo ga
        //Kritična sekcija
        zeli2= false;
```

Лампортов (пекарски) алгоритам

- Ово је уопштење Питерсоновог алгоритма за рад са више од 2 процеса
- Идеја: нпр. муштерије долазе у пекару да купе хлеб
  - Свака муштерија добија број већи од бројева претходних муштерија
  - Муштерије се потом услужују редом према добијеним бројевима
  - Пошто је и додела бројева К.С. исти број се може доделити више пута
    - Тада се процес са мањим индексом опслужује први (може се користити PID)

Лампортов (пекарски) алгоритам (2)

```
proces_i()
      //niz koji štiti uzimanje brojeva – i-ti proces najavljuje uzimanje
       uzima[i]=true;
       broj[i]=max(broj[0],broj[1],…,broj[n-1])+1;
       uzima[i]=false;
       for(k=0; k<n; k++){
              while(uzima[k])
                     ; //čekamo dok neko drugi uzima broj
              while(broj[k]!=0 and (broj[k],k)<(broj[i],i))
                     ; //čekamo dok neki proces ima prednost
       }
       //Kritična sekcija
       broj[i]=0; //označavamo da ne čekamo više na procesor
```