Համակարգային ծրագրավորում

Պրոցեսներ և հոսքեր

Պրոցեսներ

Պրոցեսը պրոցեսորի կողմից կատարվող ծրագիրն է, որն ունի

- իր սեփական հասցեային տարածությյունը,
- Իր պահունակը,
- Իր հրամանների հաշվիը,
- իր ռեգիստրները՝ իրենց արժեքներով,
- իր փոփոխականները՝ իրենց արժեքներով:

Պրոցեսների կառավարում

- պրոցեսների ստեղծում/հեռացում
- պրոցեսների պլանավորում
- պրոցեսների սինխրոնիզացիա
- պրոցեսների կոմունիկացիա
- փակուղային իրավիճակների հանգուցալուծում:

Պրոցեսի կյանքի ցիկլ

Պրոցեսը ստեղծվում է եթե.

- բեռնվում է համակարգը,
- աշխատող պրոցեսը կատարում է համակարգային կանչ (fork, CreateProcess) նոր պրոցես ստեղծելու նպատակով
- օգտատերը կատարում է համակարգային կանչ (fork, CreateProcess) նոր պրոցես ստեղծելու նպատակով
- իրականացվում է փաթեթային առաջադրանք:

Պրոցեսի կյանքի ցիկլ

Պրոցեսն ավարտվում է եթե.

- կատարվում է պրոցեսն ավարտող համակարգային կանչ (exit,ExitProcess)
- տեղի է ունեցել սխալ՝ հնարավոր է ուղղվող (պրոցեսն ընդհատվում է)
- տեղի է ունեցել անուղղելի սխալ
- կատարվում է պրոցեսին ոչնչացնող համակարգային կանչ (kill, TerminateProcess)

Պրոցեսի վիճակ

- նոր ստեղծված
- պրոցեսորի կողմից կատարվող
- որևէ իրադարձության ավարտի սպասող (օրինակ մուտքի/ելքի ավարտի)
- պատրաստ՝ սպասում է պրոցեսորի ազատմանը
- ավարտված:

Պրոցեսի վիճակներ և կյանքի ցիկլ

Գործողությունների տիպերը

- կեկ անգամ կատարվող (պրոցեսի ստեղծում- պրոցեսի ավարտում)

- բազմաթիվ անգամ կատարվող (պրոցեսի կանգառ– պրոցեսի թողարկում, պրոցեսի արգելափակում – պրոցեսի ապաարգելափակում)

Մեկ անգամ կատարվող գործողություններ. պրոցեսի ստեղծում

- Պրոցեսը ստանում է **հասցեային տարածություն,** որի մեջ բեռնվում է պրոցեսի ծրագրային կոդը:
- Պրոցեսին տրամադրվում է **պահունակ** և **սիստեմային ռեսուրսներ**:
- Չնավորվում է պրոցեսի **հրամանների հաշվիչի** (IP ռեգիստրի)արժեքը:
- Ստեղծված պրոցեսը բերվում է **պատրաստ է** վիճակի:

Մեկ անգամ կատարվող գործողություններ. պրոցեսի ավարտ

- Պրոցեսը բերվում է **ավարտված է** վիճակի (զոմբի պրոցեսներ Unix-ում);
- Ազատվում են պրոցեսի հետ կապված բոլոր ռեսուրսները;
- ՕՀ-ից կախված ավարտված ծնող-պրոցեսի որդի-պրոցեսները ավարտվում են:

Բազմաթիվ անգամներ կատարվող գործողություններ

Պրոցեսի կանգառ.

- տեղի է ունենում ընդհատում ;
- պահպանվում են հրամանների հաշվիչը, ռեգիստրները պրոցեսի պահունակում;
- ղեկավարությունը տրվում է ընդհատումը մշակող ծրագրին;
- պրոցեսն անցնում *պատրաստ է* վիճակի:

Բազմաթիվ անգամներ կատարվող գործողություններ

Պրոցեսի արգելափակում

- տեղի է ունենում, երբ պրոցեսը չի կարող շարունակվել մինչև որևէ իրադարձություն տեղի ունենալը;
- օպերացիոն համակարգը կատարում է սիստեմային կանչ (օրինակ, մուտքի-ելքի գործողության կանչ);
- պրոցեսը ավելացվում է պրոցեսների հերթին;
- նորոգվում է PCB-բլոկը;
- պրոցեսը *“կատարվում է“* վիճակից անցնում է *“սպասում է“* վիճակի:

Բազմաթիվ անգամներ կատարվող գործողություններ

Պրոցեսի ապատրգելափակում

- տեղի է ունենում համակարգում որևէ իրադարձության դեպքում;
- ՕՀ-ը ստուգում է պրոցեսի վիճակը (*ապատու՞մ է*) և պրոցեսն անցնում է *պատրաստ է* վիճակի:

Պրոցեսների կառավարում

Պրոցեսը ներկայացվում է **պրոցեսի ղեկավարման բլոկով** *PCB* (Process Control Block)**,** որը պարունակում է հետևյալ ինֆորմացիան.

- պրոցեսի վիճակը;
- պրոցեսի ծրագրային հաշվիչը (IP);
- պրոցեսի ռեգիստրները;
- պրոցեսների պլանավորման և հիշողության կառավարման համար անհրաժեշտ ինֆորմացիա (պրոցեսի նախապատվություն, հասցեային տարածության ծավալ/հասցե և այլն);
- պրոցեսի հաշվառման տվյալներ (նույնականացման համար, ով է թողարկել պրոցեսիը, պրոցեսի օգտագործած ժամանակը և այլն);
- տվյալներ օգտագործվող մուտքի-ելքի սարքերի և ֆայլերի մասին:

Պրոցեսի կոնտեքստ (PCB-բլոկ)

- ռեգիստրային կոնտեքստ
- սիստեմային կոնտեքստ
- օգտատիրոջ կոնտեքստ (հասցեային տարածությունում գտնվող ծրագրային կոդը և տվյալները)

Զուգահեռ պրոցեսներ

- անկախ զուգահեռ պրոցեսներ՝ փոփոխականների բազմությունները չեն հատվում: Մի պրոցեսը չի ազդում մյուսի աշխատանքի արդյունքների վրա:

- փոխկապակցված զուգահեռ պրոցեսներ՝ օգտագործում են ընդհանուր փոփոխականներ: Մի պրոցեսը ազդում է մյուսի աշխատանքի վրա:

- մրցակցող՝ պայքարում են ընդհանուր ռեսուրսների համար

- համագործակցող՝ փոխանակվում են տվյալներով

Օրինակ. մրցակցող պրոցեսներ

Պրոցես P1

1. R1:=X
2. R1:=R1+X
3. X:=R1

Պրոցես P2

4. R2:=X
5. R2:=R2+1
6. X:=R2

P1 և P2 պրոցեսների կատարման հնարավոր սցենարներ

(1)R1:=X; (2) R1:=R1+X; (3) X:=R1;
(4) R2:=X ; (5) R2:=R2+1; (6) X:=R2;
արդյունք՝ X=2X+1

(4) R2:=X ; (5) R2:=R2+1; (6) X:=R2;
(1)R1:=X; (2) R1:=R1+X; (3) X:=R1;
արդյունք՝ X=2X+1

(1)R1:=X; (4) R2:=X; (2) R1:=R1+X;
(5)R2:=R2+1; (3) X:=R1; (6) X:=R2;
արդյունք՝ X=X+1

Հոսքեր

Հոսքը պրոցեսորի կողմից կատարվող հրամանների հաջորդականություն է:

Մեկ պրոցեսի շրջանակում կարող է կատարվել

- մեկ՝գլխավոր հոսք (*մեկհոսքային պրոցեսներ*)
- մի քանի տարբեր հոսքեր (*բազմահոսքային պրոցեսներ*)

Հոսքերը ստեղծվում են պրոցեսում համապատասխան համակարգային կանչերի միջոցով (օրինակ,CreateThread):

Գլխավոր հոսքը ստեղծվում է լռությամբ՝ պրոցեսի ստեղծման ժամանակ:

Պրոցեսի շրջանակում կատարվող հոսքն ունի

- իր սեփական պահունակը,
- իր հրամանների հաշվիչը,
- իր ռեգիստրների արժեքները:

Հոսքն օգտվում է պրոցեսի հասցեային տարածությունից և ռեսուրսներից:

Պրոցեսների/հոսքերի փոխբացառում

Փոխբացառում – քանի դեռ մի պրոցեսը/հոսքը չի ավարտել դիմումը ընդհանուր փոփոխականներին /ռեսուրսներին՝ բացառել մեկ այլ պրոցեսի/հոսքի դիմումը այդ փոփոխականներին/ ռեսուրսներին :

Կրիտիկական տիրույթ

Կրիտիկական տիրույթ - ծրագրի այն հատվածը, որտեղ կատարվում է դիմում ընդհանուր փոփոխականներին (ռեսուրսներին):

Պահանջներ կրիտիկական տիրույթների նկատմամբ

- կամայական պահի միայն մեկ պրոցես կարող է գտնվել իր կրիտիկական տիրույթում;
- ոչ մի պրոցես չի կարող անվերջ գտնվել իր կրիտիկական տիրույթում;
- ոչ մի պրոցես չի կարող անվերջ սպասել իր կրիտիկական տիրույթ մտնելուն;

- եթե որևէ պրոցես գտնվում է իր կրիտիկական տիրույթում և ավարտում է իր աշխատանքը, ապա փոխբացառման ռեժիմը պետք է անջատվի, որպեսզի այլ պրոցեսներ հնարավորություն ստանան մտնելու իրենց կրիտիկական տիրույթներ ;

- իր կրիտիկական տիրույթից դուրս գտնվող պրոցեսը չի կարող արգելափակել մեկ այլ պրոցեսի;

- ծրագիրը գրելիս հաշվի չեն առնում պրոցեսորների քանակը և արագագործությունը;

Կրիտիկական տիրույթի պաշտպանում

- ընդհատումների արգելում, երբ պրոցեսը մտնում է կրիտիկական տիրույթ, և ընդհատումների թույլատրում՝ կրիտիկական տիրույթից դուրս գալուն պես
- բլոկավորման փոփոխականներ՝
  0- ոչ մի պրոցես չի գտնվում իր կրիտիկական տիրույթում
  1- որևէ պրոցես գտնվում է իր կրիտիկական տիրույթում
- խիստ զուգորդում (turn – հերթի փոփոխական)

turn=0՝ 0-րդ պրոցեսի հերթն է մտնել կրիտիկական տիրույթ

turn=1՝ 1-ին պրոցեսի հերթն է մտնել կրիտիկական տիրույթ

Ակտիվ սպասման սխեմա

Պրոցես 0

```
while (true)
{
while (turn);
critical_region();
turn=1;
non_critical_region();
}
```

Պրոցես 1

```
while (true)
{
while (!turn);
critical_region();
turn=0;
non_critical_region();
}
```

Սինխրոնիզացման միջոցներ

Պրոցես 0

{ ոչ կրիտիկական տիրույթ

enter_critical_region(0);

կրիտիկական տիրույթ

leave_critical_region(0);

ոչ կրիտիկական տիրույթ

}

Պրոցես 1

{ ոչ կրիտիկական տիրույթ

enter_critical_region(1);

կրիտիկական տիրույթ

leave_critical_region(1);

ոչ կրիտիկական տիրույթ

}

Պրոցեսների սինխրոնիզացման միջոցներ

- Հիշողության բլոկավորում (Պետերսոնի/Դեկկերի ալգորիթմ)
- TSL(Test and Set Lock) հրաման
- Սեմաֆորներ և մյուտեքսներ
- Մոնիտորներ
- Փոստարկղեր

Պետերսոնի/Դեկկերի ալգորիթմ

```
#define N 2 // պրոցեսների քանակ
int turn; // հերթի փոփոխական
int interested[N]; //բոլոր սկզբնական արժեքները FALSE են
 void enter_region(int process); // proces-ն ունի 0կամ 1 արժեք
 { int other;//մյուս պրոցեսի համարը
     other = 1 – process;
 interested[process] = TRUE; // հետաքրքրված է  կրիտիկական տիրույթ
     //մտնելու
turn = process; // պրոցեսի համարը
 while (turn == process && interested[other] == TRUE);
}
void leave_region(int process) // կրիտիկական տիրույթից դուրս եկող
     //պրոցես
 { interested[process] = FALSE; //կրիտիկական տիրույթից դուրս գալու
     //հայտանիշ }
```

TSL(Test and Set Lock) հրաման

enter region:

```
TSL REGISTER,LOCK
CMP REGISTER,#0
JNE enter_region
RET
```

leave region:

```
MOVE LOCK,#0
RET
```

Կատարվում է երկու չընդհատվող գործողությամբ

1. LOCK -> REGISTER
2. LOCK- ում գրվում է ոչ զրոյական արժեք

Սինխրոնիզացիայի պրիմիտիվներ

sleep() - արգելափակում է այն պրոցեսը, որը կանչում է այս ֆունկցիան

wakeup(process) - ապաարգելափակում է (‘արթնացնում է ’) արգումենտում նշված պրոցեսը

Այս պրիմիտիվները բացառում են ակտիվ սպասման սխեման՝ քնեցնելով այն պրոցեսը, որին չի թույլատրվում մտնել իր կրիտիկական տիրույթը

Օպերացիոն համակարգերի դասական խնդիրներ
(արտադրողի և սպառողի խնդիրը)

Այս խնդիրն անվանում են նաև ՞Սահմանափակ բուֆերի խնդիր՞:

Տրված է ահմանափակ երկարությամբ բուֆեր, որի մեջ Արտադրողը տեղադրում է նոր տարր, իսկ Սպառողը բուֆերից վերցնում է տարր:

Խնդիրն իրականացվում է երկու պրոցեսի միջոցով՝ արտադրողի պրոցես և սպառողի պրոցես:

Արտադրողի և սպառողի խնդիրը (արտադրողի պրոցես՝ producer)

```
#define N 100 //բուֆերի չափը՝ N=100
int count = 0; //գրառումների քանակը բուֆերում
 void producer(void) {
 int item;
while (TRUE) {
item = produce_item( );
 if (count == N) sleep( );
 insert_item(item);
count = count + 1;
if (count == 1) wakeup(consumer);
     }
}
```

Արտադրողի և սպառողի խնդիրը
(սպառողի պրոցես՝ consumer)

```
void consumer(void) {
int item;
while (TRUE) {
if (count == 0) sleep();
item = remove_item( );
count = count− 1;
if (count == N − 1) wakeup(producer);
   consume_item(item);
}
}
```

Սեմաֆորներ

Դեյկստրա (1965թ.)

Ամբողջատիպ ոչ բացասական արժեքներ ընդունող փոփոխական

Գործողություններ.

down(sem)

up(sem)

Գործողություններ սեմաֆորների նկատմամբ

- down(sem) - եթե sem սեմաֆորի արժեքը 0 է, ապա այս գործողությունը կարարող պրոցեսն արգելափակվում է; հակառակ դեպքում sem-ի արժեքը փոքրանում է 1-ով:

up(sem) - sem սեմաֆորի արժեքը մեծանում է 1-ով:

Արտադրողի պրոցեսը սեմաֆորների կիրառմամբ

```
#define N 100
    typedef int semaphore;//սահմանվում է semaphore տիպը
    semaphore mutex = 1; //երկուական սեմաֆոր
     semaphore empty = N; //բուֆերում դատարկ տեղերի քանակ
    semaphore full = 0; // բուֆերում լցված տեղերի քանակ
void producer(void) {
     int item; while (TRUE) {
    item = produce_item( );
    down(&empty); //արգելապակում է պրոցեսը կամ
    //փոքրացնում է բուֆերում դատարկ տեղերի քանակը
    down(&mutex); // պաշտպանում է կրիտիկական տիրույթը
    insert_item(item); //կրիտիկական տիրույթ
    up(&mutex);
    up(&full);// ավելացնում է բուֆերում լցված տեղերի քանակը
}
 }
```

Սպառողի պրոցեսը սեմաֆորների կիրառմամբ

```
void consumer(void) {
    int item;
    while (TRUE) {
    down(&full); //արգելապակում է պրոցեսը կամ փոքրացնում է
    //բուֆերում լցված տեղերի քանակը
    down(&mutex); // պաշտպանում է կրիտիկական տիրույթը
    item = remove_item( ); //կրիտիկական տիրույթ
    up(&mutex);
    up(&empty); // ավելացնում է բուֆերում լցված տեղերի
    քանակը
    consume_item(item);//մշակում է բուֆերից վերցրած տարրը
}
 }
```

Մյուտեքսներ

mutex=0 – ոչ բլոկավորված վիճակ (մյուտեքսն ազատ է)

mutex=1 - բլոկավորված վիճակ (մյուտեքսը սեփականացված է)

Օպերացիոն համակարգերի պրիմիտիվներ՝

mutex_lock()- կանչվում է կրիտիկական տիրույթ մտնելիս (սեփականացնել մյուտեքսը)

mutex_unlock()- կանչվում է կրիտիկական տիրույթից դուրս գալուց (ազատել մյուտեքսը)