Programação Paralela e Distribuída

Aula 3

Profa.: Liria M. Sato

Conteúdo

- Exclusão mútua
- Semáforos binários
- monitor
- Semáforos contadores
- eventos

Exclusão Mútua

Processamento exclusivo de processos é necessário para que recursos possam ser compartilhados sem interferências mútuas.

Consideremos o seguinte exemplo:

Em um programa um contador (COUNT) é compartilhado e incrementado por mais de um processo.

COUNT = COUNT + 1

LD COUNT

ADD 1

STO COUNT

exclusão mútua

Se 2 processos executarem esta seqüência, pode ocorrer:

```
LD COUNT     {processo 1}
LD COUNT     {processo 2}
ADD 1        {processo 2}
STO COUNT    {processo 2}
ADD 1        {processo 1}
STO COUNT    {processo 1}
```

Tem-se, então, que COUNT é incrementado apenas de 1.

A solução é encerrar a seqüência de instruções em uma seção crítica.

Seção crítica: é uma seqüência de códigos executada ininterruptamente, garantindo que estados inconsistentes de um dado processo não sejam visíveis aos restantes. Isto é realizado utilizando mecanismos de exclusão mútua.

Implementação de exclusão mútua:

se o sistema é monoprocessador: desabilita interrupções;
e o sistema tem processadores múltiplos: a desabilitação não é suficiente, sendo necessária a aplicação de protocolos de sincronização:
- adquire o controle exclusivo
- seção crítica
- libera o controle exclusivo

implementação com semáforos binários

Definições de semáforo

- Semáforo fraco (“weak semaphore”)

O semáforo S é um tipo de dados composto por 2 campos:

V: do tipo inteiro não negativo

L: do tipo conjunto de processos

Operações P(S) e V(S): atômicas

P(S)

Se S.V>0

    decrementa S.V de 1

Senão

    insere processo no conjunto S.L

    processo é bloqueado

V(S)

Se S.L está vazio

    incrementa S.V de 1

Senão

    escolhe um q arbitrário de S.L

    retira de S.L

    coloca q em estado de pronto para execução (“acorda q”)

- Semáforo Forte (“strong semaphore”)

Aqui, S.L é substituída por uma fila (Q).

P(S)

Se S.V>0

    decrementa S.V de 1

Senão

    insere processo no fim fila S.Q

    processo é bloqueado

V(S)

Se S.Q está vazio

    incrementa S.V de 1

Senão

    q:= primeiro da fila S.Q

    retira q da fila

    coloca q em estado de pronto para execução (“acorda q”)

- Semáforo espera ocupada (“busy-wait semaphore”)

P(S)

espera até S>0

decrementa S de 1

V(S)

incrementa S de 1

Semáforos

Em CPAR:

declaração de semáforo: global ou local a macrotarefa

shared Semaph nome_semaforo;

Ex: shared Semaph X;

criação : create_sem(&nome_semafor,valor_inicial);

Ex: create_sem(&X,1);

remoção: rem_semaph(&nome_semaforo);

Ex: rem_semaph(&X);

operação P: lock(&nome_semaforo);

Ex: lock(&X);

operação V: unlock(&nome_semaforo);

Ex: unlock(&X);

Semáforo Binário

```
shared Semaph X;

create_sem(&X,1);

lock(&X);

unlock(&X);

rem_sem(&X);
```

Exemplo: seção crítica

```
task body tarefa()
{shared Semaph A;
  int i;
  create_sem(&A,1);
  forall i=0 to 99
    { ....
      lock(&A);
      COUNT=COUNT+1;
      unlock(&A);
      .....
     }
   ......
   rem_sem(&A);
  }
```

Exemplo e exercício

Exercicio:

- soma dos elementos de um vetor
- soma_vet.cpar
- soma_vet1.cpar (exercício): dar uma solução melhor para a soma

get_mi_id( ) : retorna id da microtarefa

Exercício para casa

- Implementar uma função que provê uma barreira.

barreira(int num): cada processo espera num processos chegarem na barreira para continuar a execução.

Variáveis A,B,C compartilhadas

tarefa1, tarefa2 e tarefa3: simultãneas

tarefa1:	tarefa2:	tarefa3:
A=2	B=3	C=10
Barreira(3)	barreira(3)	barreira(3)
X=A+B+C*2	Y=B-A+C	Z=A+C-B
Imprime X	imprime Y	imprime Z

Exercício para casa

- cálculo do desvio dos elementos de a matriz a[1000][1000]
- matriz: compartilhada global
- inic : tarefa que inicia a matriz (a[i][j]=i)
- desvio: calcula o desvio
- imprimir o resultado

Monitor

- acesso a recursos compartilhados por processos distintos em uma mesma seqüência de códigos: solução é encerrar a seqüência em uma seção crítica
- acesso a recursos compartilhados por processos distintos em seqüências distintas de código: a solução é utilizar um semáforo binário para o controle do acesso

monitor

Exemplo:

```
task 1          task2          task 3          main()
----            ----           ----            {
lock(&s);       lock(&s)       lock(&s);        create_sem(&s,1);
B=A+1;          A=A+1;          A=A+1;          A=0;
unlock(&s);     unlock(&s)     unlock(&s);      -----
                                                create 1,task1();
                                                 create 1,task2();
                                                 create  1,task3();
                                                 ------
                                                 rem_semaph(&s);
```

Monitor: oferece ao usuário uma forma fácil e segura para efetuar acessos a recursos compartilhados

Monitor

monitor: um monitor é escrito como um conjunto de variáveis globais seguidas por um conjunto de procedimentos. A entrada do monitor é permitida somente a um processo de cada vez. Quando um processo encerra a execução de um procedimento do monitor, este é liberado e um outro processo que estava esperando ganha o acesso ao monitor.

variáveis compartilhadas: com controle de acesso por um monitor, devem ser declaradas neste monitor.

monitor

monitor : é constituído de:

- declarações de variáveis
- funções de monitor: acessíveis pela entrada no monitor, sendo o seu acesso permitido a um processo de cada vez.
- iniciação do monitor: seqüencia de comandos que é executada quando o programa é iniciado. Permite desta forma a iniciação das variáveis compartilhadas.

monitor

```
monitor nome_monitor
  declarações de variáveis
{
   tipo
    nome_funcao_1(parametros)
     declaracoes dos parametros
     {
       declaracoes
       ----
       }
    -----
```

```
tipo nome_funcao_n(parametros)
       {---
        }
}
{
  iniciacao do monitor
 }
```

Exemplo

```
monitor buffer
  shared char c;
{
  char leia()
    {
      char ch;
      when(c!=null)
        { ch=c;
          c= NULL;
         }
        return ch;
      }
```

```
void escreva(carac)
    char carac;
    {
       when (c==NULL)
          {
            c=carac;
          }
         return ch;
       }
}
{
   c=NULL;
 }
```

Exemplo (continuação)

```
task spec produz();
task body produz()
{
  char x;
  do
      { x=getchar();
        buffer.escreva(x);
       } while (x!='.');
  }
task spec consome();
```

```
task body consome()
{
  char x=NULL;
  while ((x=buffer.leia())!='.')
     putchar(x);
  }
void main()
 {
  create 1,produz();
  create 1,consome();
 }
```

Exercício

monitor contador:

- variável compartilhada (inteiro): A
- funções: incrementa(): incrementa A
- int leia(): retorna valor de A

tarefa1, tarefa2 e tarefa3:

para i de 1 a 10:

- chama função incrementa do monitor
- chama função leia do monitor
- imprime valor retornado

Semáforos Contadores: exemplo

Exemplo: (escrita e leitura em um buffer)

```
#include <stdio.h>
shared char buffer[5];
shared int pointer;
/* posicao do prox. item a ser lido e removido */
shared int count;
/* total de itens no buffer */
shared Semaph excl;
shared Semaph full;
shared Semaph empty;
task spec produz();
task spec consome();
```

```
void escreve(carac)
   char carac;
   {
      lock(&empty);
      lock(&excl);
    buffer[(pointer+count)%5]=carac;
      count=count+1;
      unlock(&excl);
      unlock(&full);
   }
```

```
char leia()
{ char x;
    lock(&full);
    lock(&excl);
    x=buffer[pointer];
    count=count-1;
     pointer=(pointer+1)%5;
  unlock(&excl);
  unlock(&empty);
  return x;
}
```

```
task body produz()
{
  char k;
  do{  k=getchar();
       escreve(k);
    } while (k!='.');
}
task body consome()
{
  char k;
  while ((k=leia())!='.')
      putchar(k);
}
```

```
void main()
{ create_sem(&full,0);
  create_sem(&empty,5);
  create_sem(&excl,1);
  pointer=0;
  count=0;
  alloc_proc(2);
  create 1,produz();
  create 1,consome();
  wait_all();
  rem_sem(&full);
  rem_sem(&empty);
  rem_sem(&excl);
}
```

Eventos

Evento é um mecanismo que bloqueia um número arbitrário de processos até que uma determinada ação aconteça em que o bloqueio é finalizado e os processos possam continuar. É uma espécie de semáforo de tráfego.

Os eventos podem ser iniciados, ativados, apagados e esperados.

Eventos

Em CPAR tem-se:

declaração:

shared Event A;

funções:

create_ev(&A) : cria evento A (estado inicial: apagado)

rem_ev(&A) : remove evento A

set_ev(&A) : ativa evento A

res_ev(&A) : apaga evento A

wait_ev(&A) : espera até que A seja ativado

bool read_ev(&A) : retorna o estado de evento

```
shared Event A;
task spec exem1();
task body exem1()
{  _____
   _______
   wait_ev(&A);
   _______
}
task spec exem2();
task body exem2()
{
   _______
   _______
   wait_ev(&A);
   _______
}
```

```
 task spec exem3();
task body exem3()
{ _____
   _____
   set_ev(&A);
   _____
}
void main()
{
  create_ev(&A);
  create 1,exem1();
  create 1,exem2();
  create 1,exem3();
  wait_all();
  rem_ev(&A);
}
```

Exemplos

- **Exemplo:**

Simulação de um sistema de semáforos de tráfego que operam sincronamente.

São utilizados eventos para simular os mecanismos de sincronização.

Exercicio (casa)

- Implementar exemplo de pipeline dado na aula.