Aula 13

Uso de Semáforos em problemas de síncronia de programação concorrente

Implementação de canais com monitores

```
monitor Channel3 {
int[3] n;
int used = 0;
void put(int x){
if (used < 3)
        { n[used] = x; used++;
}
    }
int get(){
int tmp;
if (used > 0) {
tmp = n[0];

for (int j=1; j < used; j++){
n[j-1]=n[j];}
used--;
}
return tmp; }
}
```

Implementação de canais com monitores

```
Channel3 buff;
void produce() {
buff.put(2);
buff.put(3);
buff.put(5);
buff.put(7);
}
void consume(){
int myPrime;
    myPrime=buff.get();
    println ("My prime number is "+ myPrime);

 }

mainAgent { start(produce()); start (consume())
}
```

- O consumidor pode consumir 0
- O produto não fica bloqueado se o canal estiver cheio
- Temos de garantir as condições:
  - que os dados estão disponiveis ($Dataavailable$)
  - e que há espaço ($RoomAvailable$)

```
monitor Channel3 { int[3] n;
int used = 0;
condition roomAvailable with (used < 3);
condition dataAvailable with (used > 0);
void put(int x){
waitForCondition(roomAvailable);
n[used] = x; used++;
signal(dataAvailable);}
int get(){
int tmp;
waitForCondition(dataAvailable);
tmp = n[0];
for (int j=1; j < used; j++){
n[j-1]=n[j];}
used--;
signal(roomAvailable);
return tmp;
}}
```

Semáforo em PseuCo

Construção dum semáforo com um monitor

```
monitor Semaphore {
int value;

condition valueNonZero with (value>0));
void init(int v)
{ value = v;}

void up()
{++value;
signalAll(valueNonZero);
}

void down () {
```

```
waitForCondition(valueNonZero)
 --value;
}

}
```

Rendez-vous

Supor dois processos A e B cada um com duas instruções a_1, a_2 e b_1, b_2, respectivamente. Pretende-se garantir que a_1 ocorra antes de a_2.

- corresponde a um canal síncrono.
- Dois semáforos: um para A outro para B com value=0;
- A: assinala que chegou e fica à espera de B
- B: assinala que chegou e fica à espera de A

Rendez-vous

```
void worker(Semaphore X, Semaphore Y, string id){
    println("1"+id);
    Y.up();
    X.down();
    println("2"+id);
}

mainAgent{
    Semaphore A,B;
    A.init(0); B.init(0);
   start(worker(A,B,"b"));
   start(worker(B,A,"a"));
}
```

Ficheiro:`https://pseuco.com/#/edit/remote/w5mq4jyl0lh8x8mio5j1`

***Mutex*: exclusão mútua**

Dados dois processos A e B garantir que ambos não acedem simultaneamente a uma zona crítica (p.e ambos têm uma instrução $n++$, para um n variável inteira global).

- Criar um semáforo inicializado em 1.
- Se 1, o processo pode progredir
- Se 0 tem de esperar

```
void p(int i,Semaphore A){
     println("não ",i);
    A.down();
     println("entrar",i);
    count=count+1;
     println("sair",i);
    A.up();
}
mainAgent{
    Semaphore A;
    A.init(1);
    start(p(1,A));
    start(p(2,A));
}
```

Ficheiro:https://pseuco.com/#/edit/remote/yvo471h9vwrcks1h7ynu

Multiplex

Generalizar o problema anterior para o caso de no máximo max processos podem aceder a uma zona de código.

- inicializar um semáforo com max

```
int max =4;
void p(int i,Semaphore A){
     println("não ",i);
    A.down();
     println("entrar ",i);
     println("sair ",i);
    A.up();
}
mainAgent{
    int i;
    Semaphore A;
    A.init(max);
    for(i=0;i<=10;i++){
        start(p(i,A));}
}
```

Ficheiro:https://pseuco.com/#/edit/remote/y31el8eh5gvpoq0aro8h

Barreiras (Barriers)

- Generalizar o *Rendezvous* para n processos.
- Nenhum processo pode chegar à zona crítica antes de efectuar o *rendezvous*.
- Sugestão: usar um mutex/lock para incrementar um contador que indique quantos processos chegaram ao *rendezvous*.
- n processos
- *count* no rendezvous
- semáforos: M mutex e A inicializado em 0
- ultimo $A.up()$: cada processo assinala que mais um pode passar
- experimentar incluir o if na barreira

```
int count =0;
int n=4;
void worker(int i,Semaphore A,Semaphore M){
    M.down();
    count ++;
    M.up();
    println(i,"rendezvous");
    if (count ==n){
    A.up();}
    A.down();
    A.up(); //porque?
    println("entrar",i);}
mainAgent{
    int i;
    Semaphore A,M;
    A.init(0);
    M.init(1);
    for(i=1;i<=n;i++){
        start(worker(i,A,M));}}
```

Ficheiro:`https://pseuco.com/#/edit/remote/asvqwspem28pps1m32t6`

Barreira reusável

- Permitir que uma barreira seja reusada
- Tem de se decrementar o contador

- Para isso usar outro semáforo (mutex): agora que assegure que todos os processos entraram na zona crítica.

```
        void worker(int i,Semaphore A,Semaphore A1, Semaphore M){
    int j;
    M.down();
    count ++;
      println(i," rendezvous");
      if (count ==n){A1.down();
          A.up();}
    M.up();
    A.down();
    A.up();
    println("entrar ",i);
    M.down();
    count --;
      if (count ==0)
        { A.down(); A1.up();}
    M.up();
    A1.down();
    A1.up();
}

mainAgent{
    int i;
    Semaphore A,A1,M;
    A.init(0);
    M.init(1);
    A1.init(1);
    for(i=1;i<=n;i++){
        start(worker(i,A,A1,M));}
}
```

Ficheiro:https://pseuco.com/#/edit/remote/wpvmvx76t9xt2kfc2a9j