Mohamed Boudiaf University – M'sila
Department of Computer Science
3rd year, Computer Science - SI

Solutions for Tutorial Session N°4: Monitors

Objective: Understand and apply process synchronization using monitors through exercises on managing shared resources, coordinating threads, and solving classic concurrency problems.

Propose a monitor-based synchronization algorithm for each of the following problems:

Exercise 1: (Positive Shared Counter)

Consider a shared integer counter that is initially zero. Two types of processes can access the counter: incrementers, which increase its value, and decrementers, which decrease its value. A decrementer may only execute if the counter is greater than zero; otherwise, it must wait until the counter becomes positive.

SOLUTION

```
Program PositiveCounter;
Process IncrementingProcess_i;
Begin
  CounterMonitor.Increment();
End;
```

```
Process DecrementingProcess_j;
Begin
  CounterMonitor.Decrement();
End;
```

```
Monitor CounterMonitor
  Var
    count : integer       // Shared counter
    canDecrement : Condition;     // Condition variable for decrementers
  Procedure Increment();
  Begin
    count := count + 1;
    // Wake up a waiting decrementer if any
    canDecrement.signal;
  End;
```

```
  Procedure Decrement();
  Begin
    // Wait if counter is zero
    while count = 0 do
      canDecrement.wait;
    count := count - 1;
  End;
Begin
   Count:=0;
End;
```

```
Begin
  ParBegin
```

```
    IncrementingProcess_1;……..; IncrementingProcess_i;
    DecrementingProcess_1;…….; DecrementingProcess_j;
  ParEnd;
End.
```

Exercise 2: (Barber Shop)

Consider a barber shop with **N chairs** for waiting customers, and a **chair C** for the customer being served, and a **barber B**.

- ✓ If there are no customers, the barber **B** sleeps in chair **C**.
- ✓ When a customer arrives while the barber **B** is sleeping, the customer wakes him up, sits in **C**, and gets a haircut.
- ✓ If a customer arrives while the barber is busy: if one of the **N** chairs is free, the customer sits and waits; otherwise, the customer leaves.
- ✓ When the barber finishes a haircut and there are waiting customers, he serves the next customer.

SOLUTION

Program Barber Shop

```
Process Barber;
Begin
  Repeat
    ShopManager.TestBarber();
    CutHair();
  Until false;
End;
```

```
Process Customer_i;
Var access: Boolean;
Begin
  ShopManager.TestEntry(access);
  If not(access) then LeaveShop()
  Else
  Begin
    ShopManager.Enter();
    GetHaircut();
    ShopManager.Exit();
  End;
End;
```

```
Monitor ShopManager;
Const N = … ;          // Number of waiting chairs for customers
Var
  customerCount: Integer;        // Counter for customers
  CustomerQueue, BarberQueue: Condition;  // Waiting queues for customers and barber
```

```
Procedure TestBarber();
Begin
  // If there are no customers, the barber waits
  if customerCount = 0 then
    BarberQueue.wait;
End;
```

```
Procedure TestEntry(Var access: Boolean);
Begin
  If customerCount = N + 1 then
    access := false
  Else
    access := true;
End;
```

```
Procedure Enter();
Begin
   customerCount := customerCount + 1;
   if customerCount = 1 then
      BarberQueue.signal
   else
      CustomerQueue.wait;
End;
```

```
Procedure Exit();
Begin
   customerCount := customerCount - 1;
   if customerCount > 0 then
      CustomerQueue.signal;
End;
```

```
Begin
customerCount := 0;
End;
```

```
Begin
   ParBegin
      Customer_1; … ; Customer_j;
      Barber;
   ParEnd;
End.
```

Exercise 3: (Swimming Pool)

An Olympic swimming pool can receive swimmers from three synchronized swimming teams: **T1**, **T2**, and **T3**. To organize training sessions, the rule is: at any given time, the pool can accommodate any number of swimmers but from at most 2 teams. For example, swimmers from teams **T1** and **T3** can train simultaneously, but if a swimmer from team **T2** wants to enter, they must wait until all swimmers from one team have left the pool.

```
Program Olympic_Pool
```

```
Process T1-i
Begin
   M.enterT1();
   Train();
   M.exitT1();
End;
```

```
Process T2-j
Begin
   M.enterT2();
   Train();
   M.exitT2();
End;
```

```
Process T3-k
Begin
   M.enterT3();
   Train();
   M.exitT3();
End;
```

```
Monitor M;
nb1, nb2, nb3 : integer // representing the number of swimmers in teams T1, T2, and T3, respectively.
nbclub : integer // represents the number of teams in the pool (ranging from 0 to 2)
Pool : condition // condition to block the third club;
```

```
Procedure enterT1()
Begin
   If (nb1 = 0) and (nbclub = 2) then
      Pool.wait;
   If (nb1 = 0) then nbclub++;
   nb1++;
   Pool.signal;
End;
```

```
Procedure enterT2()
Begin
   If (nb2 = 0) and (nbclub = 2) then
      Pool.wait;
   If (nb2 = 0) then nbclub++;
   nb2++;
   Pool.signal;
End;
```

```
Procedure enterT3()
Begin
   If (nb3 = 0) and (nbclub = 2) then
      Pool.wait;
   If (nb3 = 0) then nbclub++;
   nb3++;
   Pool.signal;
End;
```

Procedure exitT1();
Begin
nb1--;
If (nb1 = 0) **then**
Begin
nbclub--;
Pool.signal;
End;
End;
	Procedure exitT2();
	Begin
	nb2--;
	If (nb2 = 0) **then**
	Begin
	nbclub--;
	Pool.signal;
	End;
	End;
		Procedure exitT3();
		Begin
		nb3--;
		If (nb3 = 0) **then**
		Begin
		nbclub--;
		Pool.signal;
		End;
		End;

Begin
nb1:=0; nb2:=0; nb3:=0; nbclub:=0;
end;		

Begin **Parbegin** T11 ;…….; T1i ; T21 ;…….; T2j ; T31 ;…….; T3k ; **Parend** **End**.		

Exercise 4: (Bridge Load)

A bridge can support a maximum load of **20 tons**. The bridge is crossed by **trucks weighing 20 tons** and **cars weighing 5 tons**. You are asked to manage access to the bridge so that:

- ✓ The maximum bridge load is respected.
- ✓ Priority is given to cars: when a car and a truck request access simultaneously, the car must be allowed first, provided the bridge's maximum capacity is not exceeded.

Program Bridge B : **monitor** ;	
Process cars(i) **Begin** B.enter_car(); Cross();; B.exit_car(); **End**;	**Process** trucks(j) **Begin** B.enter_truck(); Cross();; B.exit_truck(); **End**;

Monitor B;
CONST nbmax = 4; // maximum number of cars that can cross the bridge simultaneously
VAR T : boolean; // indicates whether a truck is crossing the bridge or not
count : integer; // car counter
truck, car : condition;	

```
Procedure enter_car();
Begin
   count := count + 1;
   while (T) or (count > nbmax) then car.wait;
   car.signal;
End;
```

```
Procedure exit_car();
Begin
   count := count - 1;
   If (count = 0) then truck.signal
   Else car.signal;
End;
```

```
Procedure enter_truck();
Begin
   If (T) or (count > 0) then truck.wait;
   T := true;
End;
```

```
Procedure exit_truck();
Begin
   T := false;
   If (count > 0) then car.signal
   Else truck.signal;
End;
```

```
Begin
   count:=0;
   T:=false;
End;
```

```
Begin
   Parbegin
      Car_1 ; Car_2 ;…….Car_i ;
      Truck_1 ; Truck_2 ;…….Truck_j ;
   Parend ;
End.
```

Exercise 5: (ATM Alternation)

An **ATM** can be used by only one person at a time. When there are women and men waiting, a strict alternation must be ensured: after a woman uses the **ATM**, it must be a man's turn (if a man is waiting), otherwise women continue. The same applies for men. Initially, the turn is given to women.

SOLUTION

```
Program ATM
```

```
Process woman-i;
Begin
   M.before-withdraw-W();
   use_ATM();
   M.after-withdraw-W();
End;
```

```
Process man-j;
Begin
   M.before-withdraw-M();
   use_ATM();
   M.after-withdraw-M();
End;
```

```
Monitor M;
Var W, M : condition;
   nbW, nbM : integer;
   turn: integer // 0 for woman and  1 for man
```

procedure before-withdraw-W() **Begin** nbW++ ; **if** (turn = 0 **AND** nbW>1) **OR** (turn = 1 **AND** nbM > 0) **then** W.wait **else** turn := 0; **end**;	**procedure** before-withdraw-M() **Begin** nbM++ ; **if** (turn = 1 **AND** nbM>1) **OR** (turn = 0 **AND** nbW > 0) **then** M.wait; **else** turn := 1; **end**;

procedure after-withdraw-W()
Begin
nbW--;
if (nbM>0) **then**
begin
turn:=1;
M.signal;
End
else W.signal;
end;
	procedure after-withdraw-M()
	Begin
	nbM--;
	if (nbW>0) **then**
	begin
	turn:=0;
	W.signal;
	End
	else M.signal;
	end;

Begin
nbW:=0; nbM:=0; turn:= 0;
end;	

Begin **Parbegin** Woman 1; Woman2,……..Woman-i ; Man1; Man2,……..Man-j ; **parend** ; **end**.	

Exercise 6: (single railway track)

Two cities **A** and **B** are connected by a **single railway track**. We consider two classes of processes: trains traveling **from A to B (T-AB)** and trains traveling **from B to A (T-BA)**. There may be an unspecified number of processes (i.e., trains). The traffic rules are as follows:

- The track must **never** be used simultaneously by two trains traveling in opposite directions.
- The track **may be used simultaneously** by one or more trains, as long as they are **all traveling in the same direction**.
- Trains traveling **from B to A** have **priority**.

SOLUTION

```
Program train;
```

```
Processus TAB-i,
Begin
   M.entrerA();
   franchir();
   M.exitA() ;
End ;
```

```
Processus TBA-j,
Begin
   M.entrerB();
   franchir();
   M.exitB() ;
End ;
```

```
Moniteur m;
Var nbA, nbB: integer ;
      CA, CB:condition;
```

```
Procedure entrerA();
Begin
    while (nbB>0) then CA.wait;
   nbA++;
   CA.signal;
  end;
```

```
Procedure entrerB();
Begin
     nbB++;
     if nbA>0 then CB.wait;
     CB.signal;
end;
```

```
Procedure exitA();
Begin
   nbA--;
   if (nbA==0) CB.signal;
End;
```

```
Procedure exitB();
Begin
   nbB--;
   if (nbB==0)  then CA.signal;
End;
```

```
Begin
     nbA:=0; nbB:=0;;
end;
```

```
Begin
   Parbegin
         TAB-1;…; TAB-i;
         TBA-1;…; TBA-j;
   parend ;
End.
```