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

Solutions for Tutorial Session N°3: 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;
    canDecrement.signal; // Wake up a waiting decrementer if any
  End;
```

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

```
Begin
   Count:=0;
End;
```

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

Exercise 2: (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.

SOLUTION

```
Program Bridge
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 3:

A car service center has a **limited number of parking spots**. This limit is represented by the number **P** of parking spaces available in the center. **There are two types of cars: regular and large. Regular cars need one spot; however, large cars occupy two parking spots** instead of one (They must wait until **two spots** are available):

- A car that arrives waits in a **waiting lane** if no parking spot is available.
- Before parking, each car must go through a **car wash**; there are only **three (03) washing stations**, each for **single use**.
- Once the car is washed, it parks in an available spot (or two spots for large cars).
- As soon as the car leaves the parking spot, it is freed for another car.

Cars can be considered as parallel processes, and the parking spots and washing stations as shared resources.

SOLUTION

Program PARKINK;	
Process Car-i (size) // size=1 for regular, 2 for large **begin** P.enterParking(size); wash-car(); P.leaveWash(); park-car (); P.releaseParking(size); **end;**	**//monitor declaration** **Monitor** P **VAR** freeSpots: integer; freeWashStations: integer; waitForWash: condition; waitForParking: condition; **const** P=…..;
procedure enterParking (size: interger) **begin** **while** freeSpots < size **then** waitForParking.wait; freeSpots = freeSpots – size; **if** freeWashStations == 0 **then** waitForWash.wait; freeWashStations = freeWashStations – 1; **end;**	**procedure leaveWash()** **begin** freeWashStations = freeWashStations + 1; waitForWash.signal; **end;**

procedure releaseParking (size: integer) **begin** freeSpots = freeSpots + size; waitForParking.signal; **end;**	**// monitor initialization** **Begin** freeSpots = P; freeWashStations = 3; **end;**
// main program **Begin** **parbegin** car1 (size); car2(size); …..; car-i(size); **parend** **end**.	

Exercise 4: (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**.
- The trains traveling **from B to A** have **priority**.

SOLUTION

Program train;	
Processus TAB-i, **Begin** M.enterA(); crossing(); M.exitA() ; **End** ;	**Processus** TBA-j, **Begin** M.enterB(); crossing(); M.exitB() ; **End** ;
Monitor M; **Var** countA, countB: integer ; CA, CB:condition;	
Procedure enterA(); **Begin** **while** (countB>0) **then** CA.wait; countA++; CA.signal; **end**;	**Procedure** enterB(); **Begin** countB++; **while** countA>0 **then** CB.wait; CB.signal; **end**;

Procedure exitA(); **Begin** countA--; **if** (countA==0) CB.signal; **End**;	**Procedure** exitB(); **Begin** countB--; **if** (countB==0) **then** CA.signal; **End**;
Begin countA:=0; countB:=0;; **end**;	
Begin **Parbegin** TAB-1;…; TAB-i; TBA-1;…; TBA-j; **parend** ; **End**.	