COMS 552 HW2

Pseudocode Development

Version 1

Semaphore-based Solution

```
// Shared vars
mutex: semaphore := 1     // Control access to shared variables
bridge: semaphore := 1    // Control access to bridge
count[2]: int[] := [0, 0] // Stores number of vehicles crossing from end 0
and end 1


procedure cross(int i)
begin
    P(mutex)

    if count[i] = 0 then
        P(bridge)

    count[i] := count[i] + 1
    V(mutex)

    // Cross bridge

    P(mutex)
    count[i] := count[i] - 1

    if count[i] = 0 then
        V(bridge)

    V(mutex)

end cross
```

Monitor-based Solution

```
type bridge_traffic = monitor
begin
    count[2]: int[] := [0, 0] // Stores number of vehicles crossing from
end 0 and end 1
    OKtocross[2]: condition[] // Stores condition marking whether ok to
```

```
cross bridge

    procedure start_cross(int i)
    begin
        if count[1 - i] > 0 then
            OKtocross[i].wait

        count[i] := count[i] + 1

        OKtocross[i].signal
    end

    procedure end_cross(int i)
    begin
        count[i] := count[i] - 1

        if count[i] = 0 then
            OKtocross[1 - i].signal

    end

end bridge_traffic
```

Version 2

P/V-based Solution

```
// Shared vars
s0: sempahore := N0          // Number of vehicles trying to cross the
bridge from end 0
s1: sempahore := N1         // Number of vehicles trying to cross the
bridge from end 1
bridge: semaphore := 1     // 1 means free, 0 means occupied


procedure cross(int i)
begin
    if i = 0 then
        SP(s0, 1, 1) // A waiting vehicle from end 0 is being processed

        // Acquire the bridge and check if there are no active vehicles
from end 1
        SP(bridge, 1, 1; s1, N1, 0)

        // Cross bridge

        // Add back to s0 and give back bridge control
        SV(bridge, 1, 1; s0, 1, 1)
```

```
    else

        // Process vehicle from end 1 if there are no waiting or crossing
vehicles from end 0
        SP(s1, 1, 1; s0, N0, 0)
        SP(bridge, 1, 1) // Acquire bridge

        // Cross bridge

        // Add back to s1 and give back bridge control
        SV(bridge, 1, 1; s1, N1, 1)

end cross
```

Monitor-based Solution

```
type bridge_traffic = monitor
begin
    count[2]: int[] := [0, 0] // Stores number of vehicles crossing from
end 0 and end 1
    waiting[2]: int[] := [0, 0] // Number of vehicles waiting
    OKtocross[2]: condition[] // Stores condition marking whether ok to
cross bridge

    procedure start_cross(int i)
    begin

        waiting[i] := waiting[i] + 1

        if i = 0 then
            // Priority given to vehicle from end 0

            if count[1] > 0 then
                OKtocross[i].wait

            waiting[0] := waiting[0] - 1

            count[0] := count[0] + 1

            OKtocross[0].signal

        else
            if count[i] > 0 or waiting[i] > 0 then
                OKtocross[i].wait

            waiting[1] := waiting[1] - 1

            count[1] := count[1] + 1
```

```
            OKtocross[0].signal

    end

    procedure end_cross(int i)
    begin
        count[i] := count[i] - 1

        if count[i] = 0 then

            // Priority given to end 0
            if waiting[0] > 0 then
                OKtocross[0].signal
            else if waiting[1] > 0 then
                OKtocross[1].signal

    end

end bridge_traffic
```

Version 3

Monitor-based Solution

```
type bridge_traffic = monitor
begin
    count[2]: int[] := [0, 0] // Stores number of vehicles crossing from
end 0 and end 1
    waiting[2]: int[] := [0, 0] // Number of vehicles waiting
    turn: int := 0              // Whose turn it is when both sides have
waiting vehicles
    OKtocross[2]: condition[] // Stores condition marking whether ok to
cross bridge

    procedure start_cross(int i)
    begin

        waiting[i] := waiting[i] + 1

        // Wait if opposite direction is crossing and
        // there are vehicles waiting on both sides or it is not our turn
        if count[1 - i] > 0 and (waiting[1 - i] > 0 or turn != i) then
            OKtocross[i].wait

        waiting[i] := waiting[i] - 1
        count[i] := count[i] + 1

        // Signal next vehicle from same direction if no one waiting on
```

```
opposite side
        if waiting[1 - i] = 0 then
            OKtocross[i].signal



    end

    procedure end_cross(int i)
    begin
        count[i] := count[i] - 1

        if count[i] = 0 then

            // When last vehicle crosses, if there are any left on other
side waiting, then signal
            if waiting[1 - i] > 0 then
                turn := 1 - i // Switch sides
                OKtocross[i].signal

            // Otherwise if more waiting on current side, signal them
            else if waiting[i] > 0 then
                OKtocross[i].signal

    end

end bridge_traffic
```

Seralizer-based Solution

```
bridge_traffic: serializer
var
    waiting_queue[2]: queue    // Queues for vehicles waiting and end0 and
end1
    crossing_crowd[2]: queue     // Crowds for vehicles crossing from end0
and end1
    turn: int := 0              // Whose turn it is when both sides have
waiting vehicles

procedure cross(int i)
begin
    // Entry Event

    // Establish Event
    enqueue(waiting_queue[i]) until
        empty(crossing_crowd[1 - i]) and
        (empty(waiting_queue[1 - i]) or turn = i)
```

```
    joincrowd(crossing_crowd[i]) then
        // Cross the bridge (hollow region)
    end

    // Leave-crowd event

    if not empty(waiting_queue[0]) and not empty(waiting_queue[1]) then
        turn := 1 - i

    // Exit event

end cross
```