CS 5348 Operating Systems, Homework #2

1. Consider the following program. Find a counter example (execution sequence) that demonstrates that this software solution is incorrect in terms of mutual exclusion violation.

```plaintext
var blocked: array[0..1] of boolean;
var turn: 0..1;  // could be 0 or 1

function p (id: integer)
{ repeat
    blocked[id] := true;
    while (turn != id) do
    { while blocked[1–id] do;  // do nothing
        turn := id;
    }
    < critical section >
    blocked[id] := false;
    < remainder code for p>
    until false;
}

main ()
{ blocked[0] := false; blocked[1] := false;
    create thread to execute P(0);
    create thread to execute P(1);
}
```

Currently turn = 0
P1: set blocked[1] = true;
P1: check and find turn is not 1 ( while (turn != id) do )
P1: check and find blocked[0] = false, exit inner loop   ( while (blocked[1–id]) do; )
P0: set blocked[0] = true;
P0: check and find turn is 0, exit outer loop   ( while (turn != id) do )
P0: enter critical section
P1: set turn = 1;
P1: check and find turn is 1, exit outer loop   ( while (turn != id) do )
P1: enter critical section

2.

1) x = y+z;  -- x should be computed before it is used in 4) and 5)
2) w = y*z;  -- w should be computed first before it is used in 3) and 5)
3) u = w–3;  -- u, v, k are not used in later statements, so does not matter
4) v = x+t;
5) k = x+w;

(a)
lock lx = true;   // locked, done before any execution
lock lw = true;   // locked, done before any execution
x := y+z; unlock(lx);
w := y*z; unlock(lw);
lock(lw); unlock(lw);  u = w–3;
lock(lx); unlock(lx);  v := x+t;
lock(lx); unlock(lx);  lock(lw); unlock(lw);  k = x+w;

(b)
semaphore sx = 0;
semaphore sw = 0;

x = y+z;  signal (sx); signal (sx);
w = y*z;  signal (sw); signal (sw);
wait(sw);  u = w–3;
wait(sx);  v = x+t;
wait(sx);  wait(sw);  k = x+w;

3.
var cust: semaphore := 0;
bus: semaphore := 0;
busReady: semaphore := 0;
mutex := 1;

procedure visitor;
begin
  signal (cust);
  wait (bus);
  visitor gets in bus;
  wait (busReady);
  drive around the park;
  visitor gets off bus;
end;

procedure bus;
begin
  wait (mutex);
  signal (bus);
  signal (bus);
  wait (cust);
  wait (cust);
  signal (busReady);
  signal (busReady);
  signal (mutex);
  drive around the park;
  drop visitors;
end;

*** a bus needs to wait for two customers in mutex, otherwise, two waiting customers may get on two
different buses.

4. Implement a monitor solution for the bakery problem. Assume that you have N salesmen and customers
arrive at arbitrary times. Note: neither the customers nor the salesmen should have a busy waiting.

Monitor
{  int sales_count, initialized to N
5. When we implement the code for realizing the monitor, there is one important issue. Monitor can only allow one active thread in it. When signaling a condition variable, we need to decide whether to let the signaler or the signalee to continue. We have discussed how to implement the monitor with the choice of letting the signaler continue. Now, discuss the high-level implementation idea if we choose to let the signalee continue.

Assume that the monitor is protected by a semaphore “mutex”. Either the signaler or the signalee continues, we need to maintain a counter “temp_count” and have a semaphore “temp” to hold the one that has to wait.

If the signalee proceeds and the signaler waits, we need to let the signaler do “temp.wait”. When the signalee finishes using the monitor or when it goes for cond.wait on some condition, it needs to do the following:
If temp-count > 0, temp.signal; else signal mutex;
6. Customer process:
   send (custport, customer);
   take a seat in the entertainment center and wait;
   get service from the hairdresser;
   send (payport, payment);

Hairdresser process:
   repeat select
     send (dresserport, hairdresser);
     provide hair styling service;
   until false;

Manager process:
   numcust = 0;
   numserv = 0;

   repeat select
     when numcust < M
       { receive (custport, customer);
         numcust = numcust + 1;
         give the customer a seat in the entertainment center;
       }
     when numcust > 0 and numdresser > 0  // gray part: can be omitted
       { receive (dresserport, hairdresser);
         pair customer and dresser;
         numcust = numcust – 1;
         numserv = numserv + 1;
         numdresser = numdresser – 1;
       }
     when numserv > 0
       { receive (payport, payment);
         process payment;
         numserv = numserv – 1;
         numdresser = numdresser + 1;
       }
   until salon-close-time;