CSCI 3753: Operating Systems
Fall 2016
Problem Set Two (Solutions)

Please write your answers in the space provided.
Due date: Thursday, October 13 in class. No extensions will be given except at the instructor's discretion in documented cases of extreme hardship or emergencies. Please submit a hardcopy of your solutions.

Problem 1. [10 Points] Is the swap() function below thread-safe or not? Explain your reasoning.

```
int temp;

void swap(int *y, int *z)
{
   int local;

   local = temp;
   temp = *y;
   *y = *z;
   *z = temp;
   temp = local;

}
```

Swap is not thread-safe, because it is using a global variable temp. For example, suppose thread T1 called swap(&m,&n), where *m=1 and *n=2, and thread T2 called swap(&q,&r) where *q=7 and *r=4. Let T1 execute inside swap(), up to just before the line *z=temp. At this point, temp=*y=1, and T1 desires to set *z=1 to complete the swap. But if T1 is interrupted at this point, then T2 executes, then T2 sets temp=*y=7. Now suppose T1 context switches back in, and executes *z=temp=7. So the output of the swap() does not swap the values 1 and 2, but instead produces a value of 7 for one of the variables. Hence, the code is not thread safe.

Problem 2. [20 Points] Using TS( ) instruction, provide an implementation of semaphores.

```
typedef struct {
   PID pid;
   Boolean sleeping;
} list_item;

typedef struct {
   int value;
   boolean lock;
   struct list_item *list[ ];
} semaphore;

init (semaphore *s)
{
   value = 1; // Assume initial value to be 1; it can be set to a different value
              // depending on exact semantics
   initializes list to empty
   lock = FALSE;
}

wait (semaphore *s) {
   while (TS(&(s→lock)));
   s→value--;
   if (s→value < 0) {
      list_item x = <process_id, TRUE>
      add &x to s→list;
      lock = FALSE;
      sleep ( );
      x.sleeping = FALSE;
   }
   else lock = FALSE;
}

signal (semaphore *s) {
   while (TS(&(s→lock)));
   s→value++;
   if (s→value <= 0) {
      struct list_item *x = remove an adderss of list item <P, bool> from s→list;
      while (x→sleeping == TRUE)
      {
         wakeup (P);
         yield ( ); // preempt this process
      }
      lock = FALSE;
   }
   else lock = FALSE;
}
```

Problem 3. [35 Points] You have just been hired by Greenpeace to help the environment. Because unscrupulous commercial interests have dangerously lowered the whale population, whales are having synchronization problems in finding a mate. The trick is that in order to have children, *three* whales are needed, one male, one female, and one to play matchmaker --- literally, to push the other two whales together (*I am not making this up*!). Your job is to write three functions: *Male* ( ), *Female* ( ), and *Matchmaker* ( ). A male whale calls *Male* ( ), which waits until there is a waiting female and a matchmaker. A female whale calls *Female* ( ), which must wait until there is a waiting male and a matchmaker. Similarly, a matchmaker calls *Matchmaker* ( ), which must wait until there is a waiting male and a female. Once all three types of whales are present, all three return with one of them printing a message "A calf is born". Use semaphores to implement the required synchronization.

```
semaphore male = 0, female = 0, matchmaker = 0;
semaphore male_start = 0, male_end = 0;
semaphore female_start = 0, female_end = 0;
semaphore mutex = 1;

Male()
{
   signal(male);
   wait(male_start);
   wait(male_end);
   signal(matchmaker);
}

Female()
{
   signal(female);
   wait(female_start);
   wait(female_end);
   signal(matchmaker);
}

Matchmaker()
{
   wait(male);
   wait(female);
   wait(mutex);
   signal(male_start);
   signal(female_start);
     printf ("A calf is born\n");
   signal(male_end);
   signal(female_end);
   wait(matchmaker);
   wait(matchmaker);
   signal(mutex);
}
```

Problem 4. [35 Points] Unisex bathroom problem: CU wants to show off how politically correct it is by applying the U.S. Supreme Court's "Separate but equal is inherently unequal" doctrine to gender, ending its long-standing practice of gender-segregated bathrooms on campus. However, as a concession to tradition, it decrees that when a woman is in the bathroom, other women may enter, but no men, and vice versa. Also, due to fire code, at most N (N > 1) individuals may use the bathroom at any time.

You task is to write two functions: man_use_bathroom( ) and woman_use_bathroom( ). Provide a monitor-based solution that manages access to the bathroom. Your solution should be fair, starvation free and deadlock free.

```
void man_use_bathroom ( )
{
  unisex_bathroom.enter_bathroom_man ( );
      Use restroom
  unisex_bathroom.exit_bathroom_man ( );
}

void woman_use_bathroom ( )
{
  unisex_bathroom.enter_bathroom_woman( );
      Use restroom
  unisex_bathroom.exit_bathroom_woman ( );
}

monitor unisex_bathroom {
    int mc, mcw, fc, fcw;
    condition m_cond, f_cond;

    void enter_bathroom_man
    {
      // no women in the bathroom, no women waiting, and # of men in bathroom is < N, enter bathroom
        if (fc == 0 && fcw == 0 && mc < N) mc++;
        else {
      // have to wait
            mcw++;
            m_cond.wait( );
            mcw--; mc++;
     // check if more men waiting and mc < N; if so wake one man
            if (mcw > 0 && mc < N) m_cond.signal( );
      }
    }

void enter_bathroom_woman  is similar to enter_bathroom_man( )

void exit_bathroom_female
     {
        fc--; // one less woman in the bathroom now
      //no men waiting and at least one woman waiting
        if (mcw == 0 && fcw > 0) f_cond.signal( );
          // some men waiting and no more women in the bathroom
        else if (mcw > 0 && fc == 0) m_cond.signal( );
     }
```