Systems Programming and Operating Systems	**Assignment 3**
CS414, Summer 1996	**July 24, 1996; Due on August 1, 1996**

Problem 1 (20 points)

In the assymmetric solution, all the odd philosphers first attempt to take the fork on their left and all the even philosphers attempt to take the fork on their right. We shall prove that no deadlock occurs as there cannot be circular waiting. Note that circular waiting is possible only if every philospher waits for the next philospher along the entire circle. In fact we shall show that there cannot be a path containing more than **three** philosophers in the resulting holt graph.

Assume that the following scenario (a section of the holt graph) is possible –

$$P_0 \rightarrow F_1 \rightarrow P_1 \rightarrow F_2 \rightarrow P_2 \rightarrow F_3 \rightarrow P_3$$

where the P's are philosophers and the F's are forks.

In the above scenario, P_2 holds F_2 and so must have must have taken F_3 (because P_2 is even) before it took F_2 and hence it cannot be waiting for P_3, a contradiction.

Problem 2 (20 points)

```
type read-write = monitor
begin
   int n_readers = 0;    /* number of readers reading */
   int n_writers = 0;    /* number of writers writing -- max is 1 */
   int n_r_waiting = 0;  /* number of readers waiting */
   int n_w_waiting = 0;  /* number of writers waiting */
   cond read_q;          /* queue of waiting readers */
   cond writ_q;          /* queue of waiting writers */

procedure enter_read()
begin
   if (n_writers != 0 || n_w_waiting != 0)
   begin
      /* some writer is writing or waiting. We shall wait
       * to preserve mutual exclusion and fairness
       */
      n_r_waiting++;
      read_q.wait;
      n_r_waiting--;
   endif

   n_readers++;
end

procedure enter_write()
begin
   if (n_readers != 0 || n_writers != 0)
   begin
      /* Somebody is either reading or writing -- we better
```

```
        * wait to preserve mutual exclusion. Note that no
        * reader can be waiting unless somebody is reading or writing
        */
       n_w_waiting++;
       writ_q.wait;
       n_w_waiting--;
    endif

    n_writers++;
end


procedure exit_read()
begin
    n_readers--;
    if (n_readers == 0 && n_w_waiting > 0)
    begin
       /* If we are the last reader and some writer is waiting
        * let him in
        */
       writ_q.signal;
    endif
end

procedure exit_write()
begin
    n_writers--;
    if (n_r_waiting > 0)
    begin
       /* If any readers are waiting let them in first to be fair */
       int n = n_r_waiting;

       while (n > 0) do
          read_q.signal;
          n--;
       endwhile
    endif
    elseif (n_w_waiting > 0)
    begin
       /* a writer is waiting -- let him in */
       writ_w.signal;
    endif
end

end monitor;
```

Problem 3 (20 points)

In the sleeping barber problem we model each customer as a process *customer* and the barber as the process *barber*.

```
semaphore barbers_chair = 1;         /* used to simulate the barbers chair */
semaphore customer_available = 0;    /* used to wakeup the sleeping barber */
semaphore customer_ready = 0;        /* customer ready to get haircut */
semaphore haircut_over = 0;          /* customer waiting for haircut to finish  */
semaphore mutex        = 1;          /* for mutual exclusion */
int       n_customers  = 0;          /* number of customers  */

Process barber()
{
   while(1) {
      wait(mutex);

      if (n_customers == 0) {
          signal(mutex);             /* release the mutex */
          wait(customer_available);  /* sleep till a customer is available */
      } else
          signal(mutex);

      signal(barbers_chair);         /* let a customer sit in the chair */
      wait(customer_ready);          /* wait till customer is seated */
      CUT_HAIR();
      signal(haircut_over);          /* tell the customer haircuts done */
   }
}

Process customer()
{
   wait(mutex);

   if (n_customers == N) {
        /* leave if all the chairs are taken */
        signal(mutex);
        LEAVE_SHOP();

    } else {
        if (n_customers == 0) {
            /* wake up the barber if you are the first customer */
            signal(customer_available);
        }

        n_customers++;
        /* release mutex as you are going to wait in queue */
        signal(mutex);
```

```
        /* Take a seat and wait till barbers chair is free */
        wait(barbers_chair);

        wait(mutex);
        n_customers--;
        signal(mutex);

        /* get a haircut */
        signal(customer_ready);
        wait(haircut_over);

        LEAVE_SHOP();
    }
}
```