1. Consider the following program:

```
co   < await  (x  >=  3)   x =  x - 3; >
//   < await  (x  >=  2)   x =  x - 2; >
//   < await  (x  ==  1)   x =  x + 5; >
oc
```

(a) For what initial values of x does the program terminate assuming that scheduling is weakly fair? What are the corresponding final values? (b) Write a proof outline to demonstrate your claim.

2. Suppose we have to write a busy-wait and weakly fair implementation for semaphores for a machine that has atomic increment and decrement instructions. INC(x) atomically adds 1 to integer variable x, and DEC(x) atomically subtracts 1 from integer variable x. Both INC(x) and DEC(x) return the modified value of x. (a) What is correct, and (b) what is not correct with the following solution? (c) What is bad with it performance wise? (d) How would you correct it and improve its performance? (e) What would be the requirement for a strongly fair implementation?

```
/* P(s): */
while (DEC(s) < 0) {
       INC(s);  #undo decrement
       }

/* V(s): */
INC (s)
```

3. Dining philosopher problem: Five philosophers sit around a round table spending their lives thinking and eating. There are five forks on the table one between every two philosophers. For eating a philosopher needs exclusive access to both of the forks beside him. A naïve solution assigns one semaphore for each fork:

```
sem fork[5]= { 1, 1 , 1 , 1,  1};
process Philosopher [i = 0 to 4] {
       while (true)  {
              P(fork[i]); P(fork[(i+1)%5]);
              eat;
              V(fork[i]); V(fork[(i+1)%5]);
              think;
       }
}
```

(a) What is wrong with this? Demonstrate your claim with a trace. (b) Fix the solution based on the idea that the number of philosophers allowed to eat at the same time is limited. Use one or more additional semaphores for this. (c) Formulate carefully an argumentation that your solution satisfies the following correctness requirements: 1. Safety: When eating, the philosopher should have exclusive access to the two forks on each side. 2. No deadlock can occur, 3. No philosopher would starve, i.e. the solution is fair even if the semaphores are not strongly fair, like e.g. FIFO.

4. Sleeping barber problem: A barber has a small shop -where only one person can move at a time- with one barber chair and a waiting bench. When no customer is in the shop the barber waits sleeping. When customer enters and finds the barber sleeping he awakens the barber, sits in the chair and falls to sleep waiting for the barber to finish the haircut. If barber is already busy, the customer waits sleeping on the bench. After giving the haircut the barber opens the door for the customer to leave and awakens him and waits for the customer exit. After it he starts with a new customer or if there is no one waiting he falls to sleep waiting for a new one to arrive. Consider the following monitor solution:

```
monitor Barber_Shop
  int barber=0, chair=0, open=0;
  cond barber_available;   # signaled when barber>0
  cond chair_occupied;     # signaled when chair>0
  cond door_open;          # signaled when open>0
  cond customer_left;      # signaled when customer==0

  procedure get_haircut() {
     while (barber == 0) wait(barber_available);
     barber= barber - 1;
     chair= chair + 1; signal(chair_occupied);
     while (open == 0) wait(door_open);              # customer is served here
     open= open-1; signal(customer_left)
  }

  procedure get_next_customer() {
     barber= barber+1; signal(barber_available);
     while (chair == 0) wait(chair_occupied);
     chair = chair-1;                                #
  }                                                  # service is done here
                                                     #
  procedure finished_cut() {                         #
     open = open + 1; signal(door_open);
     while (open > 0) wait (customer left)
  }
}
```

(a) Write down the monitor invariant and explain it briefly. (b) Some of the while loops can be replaced by if -statements. Determine which ones, and modify the monitor accordingly assuming signal-and-continue (SC) semantics, and (c) signal-and-wait (SW) semantics. (d) Modify the solution to cover the possibility for several barbers (each having his own chair) in the shop. Pay special attention to problem of how the barber remembers the customer identity from get_next_customer() to finished_cut(), i.e. when finishing haircut, the barber has to wake the right customer. How is the invariant changed? Use SC-semantics.

5. Choose only one of the following:

a) What does conversational continuity mean in the context of client server systems? How it relates to the stateless server concept like in NFS? What are the pros and cons of statelessness?

b) What is meant with centralized, symmetric and ring type solutions in peer-to-peer systems? Compare their pros and cons.

Please, be as precise and concise as possible. Do not use more than half a page for your answer.