Utor ID:		Name:	
Utor ID:		Name:	
Utor ID:		Name:	

Question 1: (Note, this question was probably a little too hard for a midterm.)

Using locks and condition variables I attempted to implement the following function: `do_exchange(void *arg),` which allows pairs of threads to exchange values. After two processes have called `do_exchange`, they swap the values of the arguments. The function should continue to operate correctly with successive pairs of callers. The following code is my attempt at implementing the this function. Assume that the lock and condition variable are initialized correctly before `do_exchange` is called.

```
struct lock *entry;
struct cv *got_first;

/* first and second hold the
values to exchange */

int first = -1;
int second = -1;
int got_one = 0;

void *do_exchange(void *arg){
  int value = *(int *)arg;
  lock_acquire(entry);

  if(!got_one) {

    got_one = 1;
    first = value;
    cv_wait(got_first, &entry);
    *(int *)arg = second;

  } else {
    got_one = 0;
    second = value;
    cv_signal(got_first)
    *(int *)arg = first;
  }

  lock_release(entry);
  fprintf(stderr,"%d -> %d\n",
          value, *(int *)arg);

  return NULL;
```

If I set up a program that creates 6 threads that all call exchange, then I expect the following output:

```
1 -> 0
0 -> 1
3 -> 2
2 -> 3
5 -> 4
4 -> 5
```

but I get

```
1 -> 0
3 -> 2
0 -> 3
2 -> 3
5 -> 4
4 -> 5
```

a) Explain carefully why this happens. *(This is really an exercise in tracing multi-threaded code.)*

b) Does this behaviour follow either Hoare or Mesa semantics? Explain your answer.

c) Describe in English how you could modify the code to fix the problem. Assume that the behaviour of the locks and condition variables is the same as shown in the output of the above program.

2. Scheduling

a) The Round Robin scheduling algorithm does not give preference to processes with higher priority. Propose and describe two different schemes to extend Round Robin scheduling to handle priorities.

b) The Multi-Level Feedback Queue scheduling algorithm allows processes to move between queues. Give two criteria by which a process might move to a higher priority queue. (Saying that we assign it higher priority will not receive marks.)

c) Give one criteria by which a process would move to a lower priority queue.

d) Describe two factors would you use to determine the length of a quantum for a Round Robin type scheduling algorithm.

3. Processes and files

Consider the following two programs. Assume that they run to completion correctly. Recall that the `read(int fd, char *buf, int num)` system call reads `num` bytes from the open file referred to by `fd`.

```
/* Program A*/
int main() {
  char buf[100];  int n;

  if(fork()) {
    int fd = open("text.txt",
O_RDONLY);
    n = read(fd, buf, 2);
    buf[2] = '\0';
    fprintf(stderr, "Parent %s",
buf);
    close(fd);
  } else {
    int fd = open("text.txt",
O_RDONLY);
    buf[2] = '\0';
    n = read(fd, buf, 2);
    fprintf(stderr, "Child %s",
buf);
    close(fd);
  }
    return(0);
}
```

```
/* Program B*/
int main() {
  char buf[100];  int n;
  int fd = open("text.txt",
O_RDONLY);

  if(fork()) {
    n = read(fd, buf, 2);
    buf[2] = '\0';
    fprintf(stderr, "Parent %s",
buf);
    close(fd);
  } else {
    buf[2] = '\0';
    n = read(fd, buf, 2);
    fprintf(stderr, "Child %s",
buf);
    close(fd);
  }
  return(0);
}
```

The file `text.txt` contains

```
a
b
c
```

a) What is the output of program A? (There may be more than one correct answer.)

b) What is the output of program B? (There may be more than one correct answer.)

c) Explain how the kernel data structures must be set up to support this behaviour.

4. List four different types of operations that might cause a running process to block.