Question 1

Part (c) [1 MARK]

If two threads access a shared variable at the same time, there will be a concurrency error.

Question 2

Question 3. [2 MARKS]

Consider the following excerpt from the solution to the reader/writer problem using condition variables:

```
// assume all variable are correctly initialized
// assume writing will only ever hold the values 0 or 1

pthread_mutex_lock(&mutex);
while(writing) {
    pthread_cond_wait(&turn, &mutex);
}
readcount++;
pthread_mutex_unlock(&mutex);
```

Which of the following statements are true?

- ☐ `writing` must be 0 when `pthread_cond_wait` returns.
- ☐ The mutex is held by this thread while it is blocked in `pthread_cond_wait`
- ☐ The mutex is held by this thread whenever it checks the value of `writing`
- ☐ The mutex is held by this thread when the while loop terminates.

Question 3

Q3. (10 marks) (Conceptual + Reasoning) [15 minutes] Answer the following short questions.

a) [1 marks] Which of the following scheduling algorithms may cause starvation? Circle all that apply.
a. FCFS
b. SJF
c. Round-Robin
d. MLFQ (final revision)

b) [2 marks] What happens if a thread executes a pthread_cond_signal on a condition variable *cv1*, if no other thread is currently waiting on *cv1*? Explain in detail.

c) [2 marks] What's the difference between an interrupt and a system call?

d) [2 marks] Kernel-level threads are typically faster than user-level threads. Do you agree with this statement? Explain your rationale in detail.

e) [3 marks] A program called "run_stuff" uses 1 fork() system call and 2 exec() system calls. The exec system calls execute the following commands "ls -l" and "cat /proc/cpuinfo", respectively. Depending on how these system calls are issued, how many processes are likely to be spawned by the "run_stuff" program (other than the main program itself)? You must explain your rationale to get marks for this question.

Question 4

c) (6 marks) The bank hires you to change the `transfer_amount` implementation so that the amount can be transferred only if there is at least that amount in the account. If the account were to be left with a negative balance as a result of the transfer, then the operation waits until the account gets sufficient money (from some other transfer).
You may modify the account structure as you see fit, and add in it any other synchronization variables you may need (no need to worry about initializing them, as long as it is clear what your intent is). Your solution should ensure *no deadlocks* occur. You may *NOT declare any global variables for synchronization purposes.*
Note: efficiency does matter (particularly in terms of achievable parallelism)!

```
typedef struct acct {

      float balance;



} account;


void transfer_amount(account *a1, account *a2, float amount) {









}
```

Question 5

Q5. (6 marks) Scheduling (Reasoning) [10 minutes]
Consider a **process scheduling** algorithm which prioritizes processes that have used the least processor time in the recent past. Answer the following questions and explain *in detail* your rationale.

a) Will this algorithm favour either CPU-bound processes or I/O-bound processes?
b) Can this algorithm permanently starve either CPU-bound or I/O-bound processes?

Note: Do not feel compelled to use this entire page, but be specific and elaborate on your thought process!

Question 6

Q1. (1 mark each) True/False [5 minutes]
Indicate below, for each statement, whether it is (T)rue or (F)alse. Circle the correct answer.

T/F Whenever the processor is in kernel mode, interrupts should be disabled.

T/F One of the ways a context switch is triggered is when a process calls yield() voluntarily.

T/F User-level threads are not visible to the OS for scheduling purposes.

T/F A process will go back to the Running state immediately after receiving a SIGCONT signal.

T/F A process is placed in the blocked state when it fails to acquire a spinlock.

T/F In the round-robin scheduling algorithm, the quantum assigned to a process should be roughly equal
e context switch time.

Q2. (2 marks each) (Conceptual) [5 minutes]
Explain briefly the following concepts or terms, in the context of this course:

a) Spinlock

b) Scheduling quantum

c) System call

Question 7

Q1. (1 mark each) True/False [5 minutes]

Indicate below, for each statement, whether it is (T)rue or (F)alse. Circle the correct answer.

T/F Upon receiving a system call interrupt, the OS is responsible for switching the mode bit to allow privileged instructions to occur.

T/F The Process Control Block of a process stores all its information needed to carry out a context switch.

T/F Kernel-level threads are more lightweight than user-level threads.

T/F One of the roles of an OS is to take care of protection domains.

T/F An atomic operation which modifies the contents of a shared variable may leave this variable in an inconsistent state if an interrupt arrives midway through the operation.

T/F The Ready queue contains only those processes which finished their allocated time quantum and got descheduled.

Q2. (2 marks each) (Conceptual) [5 minutes]

Explain briefly the following concepts or terms, in the context of this course:

a) Test-and-set

c) Preemptive scheduling

Question 8

Q3. (16 marks) (Conceptual + Reasoning) [15 minutes] Answer the following short questions.

a) [1 mark] Which of these scheduling algorithms minimize average wait time? Circle all that apply.

a. FCFS
b. SJF
c. Round-Robin
d. MLFQ

b) [3 marks] Compare and contrast condition variables and semaphores. Describe a situation where condition variables may be preferred over semaphores.

c) [2 marks] Does disabling interrupts ensure mutual exclusion is achieved? Explain why or why not.

d) [2 marks] Explain why it's necessary to validate user pointers for system calls.

e) [2 marks] Describe one possibility of using both kernel threads and user-level threads in a hybrid way. Explain why this would work well and in what situations.