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 to 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.

Question 9

Q4. (7 marks) Synchronization (Reasoning)

You work for a company called FameBook, which launches a social network application, allowing their users to establish groups of friends. The code behind this application uses some synchronization, as shown below.

You can assume that a *friend_list* is a data type that represents a basic linked list, where each node contains a *user* structure and a *next* pointer. The list supports the following operations:

```
void list_add (friend_list *l, user *u);

void list_remove (friend_list *l, user *u);

int user_is_in_list (friend_list *l, user *u);
```

The first operation adds a user to a given list. The second removes a user from a given list (if the user is already in the list). The last operation checks if a user is in a given list (returns an integer: 1=yes, 0=no).

The function *friend_someone()* is called whenever a user wants to befriend another user (you can assume the request is always granted). The function *defriend_someone()* is called when a user wishes to break a friendship with another user.

Your task is to decide whether the two functions are correct, or describe all the problems that may arise. Indicate below what your conclusions are, explaining in detail your reasoning. Give examples if necessary.

```
typedef struct _user {
   char *name;
   user_list *friends;
   pthread_mutex_t *lock;
} user;
```

```
void friend_someone(user *me, user *newfriend) {

   pthread_mutex_lock(me->lock);

   if ( ! user_is_in_list(me->friends, newfriend)) {
      list_add(me->friends, newfriend);

      pthread_mutex_lock(newfriend->lock);
      list_add(newfriend->friends, me);
      pthread_mutex_unlock(newfriend->lock);

      printf("%s is now connected to %s\n",
            me->name, newfriend->name);
      return;
   }
   pthread_mutex_unlock(me->lock);
}
```

```
void defriend_someone(user *me, user *oldfriend) {

   pthread_mutex_lock(me->lock);

   if ( user_is_in_list(me->friends, oldfriend) ) {
      list_remove(me->friends, oldfriend);

      pthread_mutex_lock(oldfriend->lock);
      list_remove(oldfriend->friends, me);
      pthread_mutex_unlock(oldfriend->lock);

      printf("%s is no longer connected to %s\n",
            me->name, oldfriend->name);
      return;
   }
   pthread_mutex_unlock(me->lock);
}
```

Question 10

Q5. (5 marks) Scheduling (Reasoning) [10 minutes]

Assume that we have a multi-level queue scheduler, with 2 queues Q0 and Q1, where Q0 is the highest priority queue. Both queues use a round-robin scheduling algorithm, with a time quantum of 100 nanoseconds.

New processes and processes returning from I/O start at the back of Q0. If a process returns from I/O at the exact time when a new process arrives, the new process gets enqueued first. When a process finishes its time quantum, it gets preempted and placed at the back of the lower queue (unless it is already in Q1, in which case it stays in Q1).

This scheduler is used on a typical Unix machine. A context switch on this machine is typically 2-3 microseconds. Most processes running on this system typically finish after 5 seconds of runtime.

After having a look over this scheduler, Linus Torvalds states that this scheduler is horrible.

a) Why would Linus state this? Do you agree? Explain your answer.

b) If you agree with Linus, how would you improve your scheduler, such that the system remains interactive (all processes make some progress)?

Question 11

TRUE	FALSE	Threads that are part of the same process can access the same TLB entries.
TRUE	FALSE	The **convoy** effect occurs when longer jobs must wait for shorter jobs.
TRUE	FALSE	Two processes reading from the same virtual address will access the same contents.
TRUE	FALSE	A TLB caches translations from virtual page numbers to physical addresses.
TRUE	FALSE	Threads that are part of the same process share the same page table base register.
TRUE	FALSE	Deadlock can be avoided by having only a single lock within a multi-threaded application.
TRUE	FALSE	The number of virtual pages is always identical to the number of physical pages.
TRUE	FALSE	The OS may not manipulate the contents of an MMU.
TRUE	FALSE	If a lock guarantees progress, then it is deadlock-free.
TRUE	FALSE	With dynamic relocation, hardware dynamically translates an address on every memory access.

Question 12

Question 2. General [5 MARKS]

1. Which of the following components are shared between threads in a multithreaded process? (Check all that apply)

   - ☐ heap memory
   - ☐ stack memory
   - ☐ code
   - ☐ program counter

2. For two processes accessing a shared variable, Peterson's algorithm provides (Check all that apply)

   - ☐ mutual exclusion
   - ☐ bounded waiting
   - ☐ no preemption
   - ☐ progress
   - ☐ preventing circular waiting

3. The program `xxd` outputs the byte sequence, `daa0 2e0e`. This sequence represents an integer in little-endian byte order. Which of the following shows the integer in "normal" order (big-endian)? (Check one)

   - ☐ `2e0e daa0`
   - ☐ `0e2e a0da`
   - ☐ `daa0 2e0e`
   - ☐ `a0da 0e2e`
   - ☐ `e0e2 0aad`

4. `xxd` outputs the following sequence of bytes: `6566 5768`. What string does it represent? (Check one) (Relevant ASCII values: '8' = 56, '9' = 57, 'A' = 65, 'B' = 66, 'D' = 68, 'K' = 75, 'L' = 76, 'V' = 86)

   - ☐ AB9D
   - ☐ D9BA
   - ☐ 9DAB
   - ☐ 8BKV
   - ☐ VKB8

5. Which of the following are stored in an ext2 inode? (Check all that apply)

   - ☐ file size
   - ☐ file name
   - ☐ array of pointers to blocks
   - ☐ type of the file
   - ☐ location of block bitmap
   - ☐ number of links

Part (e) [1 MARK]

What data structure allows the kernel to determine when a process is accessing an invalid memory area?

Part (f) [1 MARK]

In round-robin scheduling, new processes are placed at the end of the queue, rather than at the beginning. Suggest a reason for this.

Part (g) [2 MARKS]

The multi-level feedback queue (MLFQ) policy periodically moves all jobs back to the top-most queue. On a particular system this is done every 10 seconds. Consider the effects of shortening this time interval to 1 second.

i– [1 MARK] Explain a positive effect that may occur.

ii– [1 MARK] Explain a negative effects that may occur (other than increased overhead costs).

Part (h) [2 MARKS]

Is the following statement true or false? Explain your answer. *A longer scheduling time slice is likely to decrease the overall TLB miss rate in the system.*

Question 13

Question 5. Synchronization [5 MARKS]

Consider the following code for inserting into a shared linked list. Assume that `malloc` always succeeds. Only the line numbers of interest are labelled. If thread T executes line 1 we label it as T1. If the scheduler is run such that thread T executes the first line of the function and then thread S executes the first line of the function, we would write the schedule as T1 S1

```
struct node {
    int key;
    struct node *next;
}

void insert(struct node **head, int key){
1   struct node *new = malloc(sizeof(struct node));
2   new->key = key;
3   new->next = *head;
4   *head = new;
}
```

In each of the subquestions below, assume we have a variable `L` of type `struct node *` that points to a list two nodes with keys 3 and 4. Assume thread T calls `insert(&L, 2)` and thread S calls `insert(&L, 5)`. (Each question has the same initial state.)

Part (a) [1 MARK]

If the two threads run according to the following schedule, check the box that indicates the correct final state of the list: T1 T2 T3 T4 S1 S2 S3 S4

- ☐ 2 5 3 4
- ☐ 5 2 3 4
- ☐ 2 3 4 5
- ☐ 2 3 4
- ☐ 5 3 4

Part (b) [2 MARKS]

Write a schedule using the notation described above where the final state of the list is 2 3 4. Assume both threads run correctly to completion.

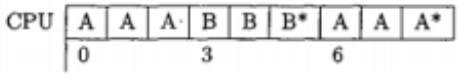
Part (c) [2 MARKS]

Add appropriate synchronization to the insert method to prevent the race condition illustrated above. Full marks only for solutions that maximize concurrency.

Question 14

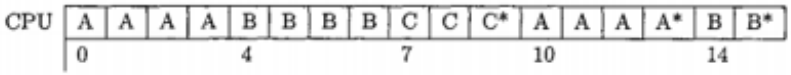
Question 7. Scheduling [9 MARKS]

For this question we will use diagrams to depict scheduling algorithms. For example, let's say we run job A for 3 time units, and then run job B for 3 times units and then run job A again for 3 time units. A diagram of this schedule is shown below. Note that an * is placed on the last execution unit of a process indicating that it is done and will not run again.



Part (a) [2 MARKS]

Assume the following schedule for a set of three jobs: A, B, and C.



Which of the scheduling disciplines below could allow this schedule to occur? Check a box if it is possible for the scheduling policy to lead to the schedule above.

- ☐ Round Robin (RR)
- ☐ First In First Out (FIFO)
- ☐ Shortest Remaining Time First (SRTF)
- ☐ Multi-level Feedback Queue (MLFQ)

Part (b) [2 MARKS]

Complete the picture of the non-preemptive shortest job first (SJF) scheduling with three jobs that arrive at the same time:

- A - 4 time units
- B - 2 time units
- C - 3 time units

Part (c) [5 MARKS]

Assume we have a multi-level feedback queue (MLFQ) scheduler with three levels of priority. The time-slice (quantum) for jobs with highest priority is 1 time unit, for jobs with the middle priority is 2 time units, and 3 for the lowest priority.

i– [2 MARKS] What is the benefit of assigning a smaller-time slice for higher priority jobs? Explain your answer.

ii– [3 MARKS] Assume jobs A and B arrive at the same time (A arrives just before B). They each have a run-time of 7 time units and are CPU-bound, making no I/O requests. Using the properties of the MLFQ scheduling policy as above, draw a picture of how the scheduler behaves. The queues are labeled below.