Calculators may be used in this examination provided they are not capable of being used to store alphabetical information other than hexadecimal numbers

UNIVERSITY OF BIRMINGHAM

School of Computer Science

LI Operating Systems and Systems Programming

Main Summer Examinations 2023

Time allowed: 2 hours

[Answer all questions]

Note

Answer ALL questions. Each question will be marked out of 20. The paper will be marked out of 80, which will be rescaled to a mark out of 100.

Question 1

(a) The code below is intended to launch ten threads with the integers 1 through 10 inclusive. However, when the code is executed, it does not print the anticipated results, can you explain why? (note: the anticipated output is printing the numbers from 1 through 10 e.g, 1235789 10 6 4)

```
 1 #include <pthread.h>
 2 void * myfunc(void * arg) {
 3   int i = * ((int * ) arg);
 4   printf("%d ", i);
 5   pthread_exit(NULL);
 6 }
 7 int main() {
 8   int i;
 9   pthread_t tid;
10   for (i = 1; i < 11; i++) {
11     pthread_create( & tid, NULL, myfunc, & i);
12   }
13 }
```

[4 marks]

(b) Modify the code in a) to correct the inconsistency so that the intended results are displayed correctly. Briefly, explain your logic. **[8 marks]**

(c) The insert C function below inserts a new element into a singly link lists to the end of a list. Modify the insert() function to an enqueue() function, which inserts nodes into a priority queue by priority. Assume that each node has a priority field. Note: A priority queue is a (singly) link list ordered by priority, with high priority values in front. In a priority queue, nodes with the same priority are ordered First-In-First-Out (FIFO). Briefly explain your logic via commenting your code.
Use the following signature for the enqueue function:

```
int enqueue(NODE **queue, NODE *p)
```

```
 1 void insert(NODE **list, NODE *p)
 2 {
 3     NODE *q = *list;
 4     if (q == 0)
 5         *list = p;
 6     else{
 7         while (q->next)
 8             q = q->next;
 9         q->next = p;
10         }
11     p->next = 0;
12 }
```

[8 marks]

Question 2

The following code represent thread-unsafe stack implementation:

```
 1 #define STACK_SIZE 20
 2 int count;
 3 double values[STACK_SIZE];
 4 void push(double v) {
 5   values[count++] = v;
 6 }
 7 double pop() {
 8   return values[--count];
 9 }
10 int is_empty() {
11   return count == 0;
12 }
```

(a) Based on your understanding of the above code, which function(s) in the above stack implementation are thread-unsafe and briefly explain why the data structure is an inconsistent state? Support your answer with one example. **[5 marks]**

(b) A candidate 'solution' for the thread-unsafe stack implementation is shown below. However, the solution contains shortcoming(s), Identify and explain the error(s) may accord.

```
 1 #define STACK_SIZE 20
 2 int count;
 3 double values[STACK_SIZE];
 4 pthread_mutex_t m1 = PTHREAD_MUTEX_INITIALIZER;
 5 pthread_mutex_t m2 = PTHREAD_MUTEX_INITIALIZER;
 6 void push(double v) {
 7   pthread_mutex_lock( & m1);
 8   values[count++] = v;
 9   pthread_mutex_unlock( & m1);
10 }
11 double pop() {
12   pthread_mutex_lock( & m2);
13   double v = values[--count];
14   pthread_mutex_unlock( & m2);
15   return v;
16 }
17 int is_empty() {
18   pthread_mutex_lock( & m1);
19   return count == 0;
20   pthread_mutex_unlock( & m1);
21 }
```

[8 marks]

(c) Update the code in b) so the stack implementation will become thread-safe. Briefly, explain your logic. **[7 marks]**

Question 3

(a) What is a context switch? Why is it important for the operating system to minimise the number of context switches? **[4 marks]**

(b) A multi-user system used to work well, with low response times and good throughput. Now many users use a package for automatic program verification, and as a result the response time is high, and throughput low. How would you distinguish between overloaded CPU, thrashing and overused disk as a possible reason? **[4 marks]**

(c) A webserver for an e-commerce shop is serving several kinds of requests. The first kind consists of rendering large images, which is computationally expensive and can happen in the background. The second kind is a preview of the list of items for sale, which needs to be fast. These preview requests are computationally inexpensive but use lots of I/O. The third kind are purchase requests which are computationally inexpensive and use a moderate amount of I/O. Assume the system is highly loaded.

(i) Describe the effects of using each of FCFS, Round Robin and priority scheduling strategy in this scenario. **[9 marks]**

(ii) Which of the three scheduling strategies mentioned in part (i) would you choose for this scenario? Justify your answer. **[3 marks]**

Question 4

(a) Why is it important that critical sections in kernel code take as little time as possible? **[6 marks]**

(b) Games programmers would like to issue commands to the graphics card directly, bypassing the operating systems. What are the effects of this for stability and security? **[6 marks]**

(c) Consider the following kernel code fragment:

```
 1  struct messageList {
 2      char *message;
 3      struct messageList *next;
 4  };
 5
 6  struct messageList *msgList = NULL;
 7  int msgSize = 0; // the total size of all messages
 8
 9  // Called as part of a system call.
10  // removes message from list and copies it into buffer.
11  int removeMessage (char *buffer, size_t length) {
12      // buffer is pointer to user space
13      memcpy(buffer, msgList->message, length);
14      msgList = msgList->next;
15      msgSize = msgSize - length;
16      return length;
17  }
18
19  // Called as part of a system call.
20  // adds message from buffer to message list.
21  int addMessage (const char *buffer,
22                  // buffer is pointer to user space
23        size_t length, // size of the buffer
24        loff_t offset) {
25
26      memcpy(msgList->message, buffer, length);
27      msgSize = msgSize + length;
28      return length;
29  }
```

This code is part of a device driver for a character device. The intention is that the driver stores a list of messages. The procedures `removeMessage` and `addMessage`

are called as part of a system call whenever a message should be removed from and added to this list, respectively.

This code compiles but does not work as intended. It contains errors not necessarily limited to memory management and concurrency. Identify the errors and suggest a remedy. Your solution should maximise the degree of concurrency. For critical sections, it is enough to indicate the beginning and end of a critical section, and whether you would use semaphores or spinlocks. **[8 marks]**

End of Paper

Do not complete the attendance slip, fill in the front of the answer book or turn over the question paper until you are told to do so

Important Reminders

- Coats/outwear should be placed in the designated area.
- Unauthorised materials (e.g. notes or Tippex) must be placed in the designated area.
- Check that you do not have any unauthorised materials with you (e.g. in your pockets, pencil case).
- Mobile phones and smart watches **must** be switched off and placed in the designated area or under your desk. They must not be left on your person or in your pockets.
- You are not permitted to use a mobile phone as a clock. If you have difficulty seeing a clock, please alert an Invigilator.
- You are not permitted to have writing on your hand, arm or other body part.
- Check that you do not have writing on your hand, arm or other body part – if you do, you must inform an Invigilator immediately
- Alert an Invigilator immediately if you find any unauthorised item upon you during the examination.

Any students found with non-permitted items upon their person during the examination, or who fail to comply with Examination rules may be subject to Student Conduct procedures.