UNIVERSITY OF BIRMINGHAM

School of Computer Science

Second Year Undergraduate

06-30203

30203 LI Systems Programming in C and C++

Resit Examinations 2022

[Answer all questions]

Exam paper

Question 1

(a) The following data structure in C defines dynamically allocated matrix:

```
struct matrix_t {
  int * elems;
  int n_cols, n_rows;
};
```

Assume that the member `elems` points to a sufficiently large array to contain `n_rows*n_cols` int elements. Consider a function that sets all elements with the value `val`, implemented as follows:

```
void fill(struct matrix_t * m, int val) {
  for (int j = 0; j < m->n_cols; j++)
    for (int i = 0; i < m->n_rows; i++)
      set(m, i, j, val);
}
```

The function `fill` relies on a function `set` with the following signature:

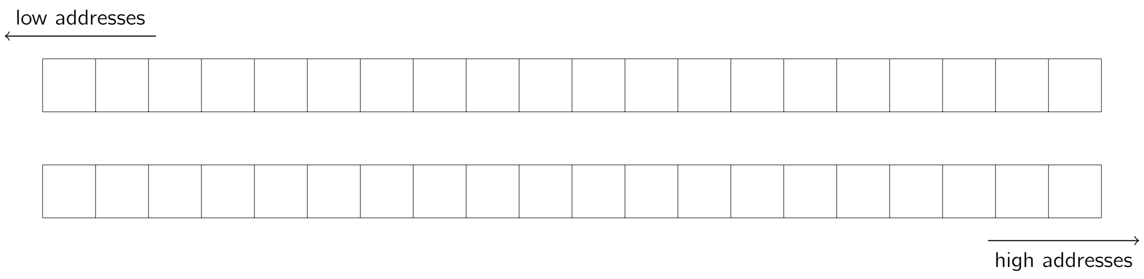
```
void set(struct matrix_t * m, int row, int col, int val);
```

Provide a C based implementation of the `set` function that sets the element at coordinates `row` and `col` to value `val` and that makes `fill` be as cache efficient as possible. **[6 marks]**

(b) Show a valid memory layout and content on the stack for the variables a,b, and c at the end of execution of the following C code:

```
char a = 10;
int b[4];
int * c = b;
while (c != b + 4)
  *(c++) = a++;
printf("%lu\n", (unsigned long) b);
```

Assume that the value printed on screen is 2000, that numbers are represented in little endian, and that we are on a 64 bits machine. Each cell of the grid below corresponds to one byte in memory, bytes are contiguous from left to right and continue in the lower grid. Note that the cells available below are more than necessary, so it is up to you to decide where to begin the sequence of bytes. For simplicity, write in each cell the decimal representation of the respective byte.



[7 marks]

(c) Interpret the struct of point (a) as a C++ struct definition and consider the code below (assume `N_ROWS` and `N_COLS` are some global integer positive constants)

```
void f(matrix_t b) {
  /* this modifies some elements of b */
  ...
}

int main() {
  matrix_t a(N_ROWS, N_COLS);
  f(a);
}
```

(i) Define a constructor with arguments `int n_rows, int n_cols` that allocates an appropriate amount of memory. Also, define an appropriate destructor that prevents the code above from leaking. **[3 marks]**

(ii) Define a copy constructor that ensures an appropriate pass-by-value to function f, that is, `b` equals to `a` when entering f but modifications to `b` do not affect `a`. **[2 marks]**

(iii) Assuming that pass-by-value works as intended, describe how to modify the code above so that changes to `b` directly apply to `a`. Briefly, justify your answer. **[2 marks]**

Question 2

(a) You have a multi-threaded application that is running on a multicore system. You have just added some code to implement mutual exclusion on some critical sections of the code. Now the system behaves correctly but real-time performance is 50% slower than before. Provide at least two reasons that are affecting performance. **[5 marks]**

(b) A computer lab is utilised to teach the Systems Programming module as well as perform CPU-intensive operations in the background for research purposes. The CPU was generally in use for nearly 100% of the time before updating the memory-intensive IDE that is used for programming, and the response time was short. The response time has increased greatly since the change, and throughput has decreased significantly. Give a possible cause for this behaviour and a solution that doesn't require any additional hardware. **[6 marks]**

(c) The following **device_write** function is part of a device driver implementation for a character device which implements a simple way of message passing. The device driver writes to the device to store the message in kernel space and adds it to the list if the message is below the maximum size, and the limit of the size of all messages wouldn't be surpassed with this message. If the message is too big, -EINVAL is returned, and if the limit of the size of all messages was surpassed, -EAGAIN is returned. This kernel code compiles correctly, but does not work as intended. Identify these errors and suggest remedies. If you think critical sections are required, provide an implementation of the critical section. Briefly justify your answer. **[9 marks]**

```
static ssize_t device_write(struct file * filp,
  const char __user * buff, size_t len, loff_t * off) {
  char * kernelBuffer;
  struct msg_t * msg;
  printf(KERN_INFO "write %d\n", (int) len);

  if (len > MSGSIZE) {
    return -1;
  }

  // prepare new list element first
  kernelBuffer = malloc(len, GFP_KERNEL);
  if (!kernelBuffer) {
    return -ENOMEM;
  }

  msg = malloc(sizeof(struct msg_t), GFP_KERNEL);
  if (!msg) {
    kfree(kernelBuffer);
    return -ENOMEM;
  }

  msg -> buf = kernelBuffer;
  if (copy_to_user(msg -> buf, buff, len)) {
    kfree(msg);
    kfree(kernelBuffer);
    return -EFAULT;
  }
  msg -> next = NULL;
  msg -> size = len;

  // add element to the list
  if (len + overall_size > max_size) {
    kfree(msg);
    kfree(kernelBuffer);
    return -1;
  }
  msg -> next = messages;
  messages = msg;
  if (lastMsg == NULL) lastMsg = msg;
  overall_size = overall_size + len;
  return len;
}
```

Question 3

(a) Give three examples of when a context-switch between processes is performed. What are the actions taken by a kernel during the context-switch? **[4 marks]**

(b) Consider three compute bound processes with 10, 6 and 4 units of burst time. Let us assume that these processes arrive at 0, 2 and 6 units of time respectively. For the Shortest Remaining Time First (SRTF), how many context switches are needed? Do not consider the context switches at time 0 or at the end. **[3 marks]**

(c) The C program shown below is compiled and run on a UNIX machine. Modify the program such that instead of the parent, the child process executes the execlp() while the parent process waits for the child process to complete. Provide the updated code. **[3 marks]**

```
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <unistd.h>
int main()
{
   int pid;
   /* fork a child process */
   pid = fork();
   if (pid < 0) { /* error occurred */
      fprintf(stderr, "Fork Failed");
      return 1;
   }else{
      /* Parent Process  */
      execlp("/bin/ls","ls",NULL);
   }
   return 0;
}
```

(d) Consider a concurrent system with three processes A, B and C. The system receives multiple requests, and places them in a request queue that is accessible by all the three processes A, B, and C. For each request, we would like to enforce an order such that the request must first be processed by A, then C, then A again and finally B before it can be removed and discarded from the queue. A (semaphore based) solution to synchronize A, B and C is given in (Table 1). Discuss whether the proposed solution is correct. If not, then provide a deadlock free solution. **[5 marks]**

Table 1: A Semaphore based solution

Process A:	Process B:	Process C:
signal(a1done)	wait(a1done)	signal(cdone)
signal(a2done)	wait(a2done)	

(e) Consider a system with three frames of memory, and the following sequence of page accesses: 1,2,3,4,1,2. When do page faults occur using FIFO, LRU and Optimal Page replacement algorithms? Briefly justify your answer. **[5 marks]**