The University of Nottingham

SCHOOL OF COMPUTER SCIENCE

A LEVEL 2 MODULE, AUTUMN EXAM 2022-2023

OPERATING SYSTEMS AND CONCURRENCY

Deadline for submission: see the Moodle submission box

Open-book examination.

Answer THREE OUT OF FOUR questions.
The first three questions written will be marked, any remaining to be ignored.

Duration of exam 3 hours.

This open-book examination will be marked out of 37.5, to be submitted no later than the DATE/TIME indicated in the moodle dropbox.

You may write/draw by hand your answers on paper and then scan them to a PDF file, or you may type/draw your answers into electronic form directly and generate a PDF file. Guidance on scanning can be found through the Faculty of Science Moodle Page Guidance for Remote Learning.

Your solutions should include complete explanations and should be based on the material covered in the module. Make sure your PDF file is easily readable and does not require magnification. Make sure that each page is in the correct orientation. Text/drawing which is not in focus or is not legible for any other reason will be ignored.

Use the standard naming convention for your document: [Student ID] [Module Code] [Academic Year]. Write your student ID number at the top of each page of your answers.

Although you may use any notes or resources you wish to help you complete this open-book examination, the academic misconduct policies that apply to your coursework also apply here. You must be careful to avoid plagiarism, collusion or false authorship. Please familiarise yourself with the Guidance on Academic Integrity in Alternative Assessments, which is available on the Faculty of Science Moodle Page Guidance for Remote Learning. The penalties for academic misconduct are severe.

Staff are not permitted to answer assessment or teaching queries during the period in which your examination is live. If you spot what you think may be an error on the exam paper, note this in your submission but answer the question as written.

Question 1 (Processes)

[**12.5 marks in total**]

a. Hyperthreaded cores have multiple sets of registers. Knowing this, would you expect a thread control block in an operating system to have space to store multiple sets of registers **(yes / no)**? **Briefly explain** your answer.

[1.5 marks]

b. Consider the code in Figure 1. **How many** child processes are created? **Briefly explain** your answer.

[1.5 marks]

```c
#include <unistd.h>
#include <stdio.h>

int main() {
  int iPID = fork();
  if(iPID == 0) {
    execl("./helloWorld.o", NULL);
    fork();
  } else {
    fork();
  }
  fork();
  return 0;
}
```

Figure 1: Fork

c. A multi-programmed system with a single CPU has a very large number of CPU bound processes running on it that all have the same priority. Could you think of a CPU scheduling algorithm that would combine the lowest possible response times with the highest possible throughput and no starvation **(yes / no)**? **Briefly explain** your answer.

[2 marks]

d. Consider the code in Figure 2 which increments a counter in parallel in two separate threads.

i. Will the code generate logically correct results if the thread library implements kernel threads and the code runs on a single-core single CPU architecture **(yes / no)**.

[0.5 marks]

ii. Will the code generate logically correct results if the thread library implements user threads and the code runs on a multi-core single CPU architecture **(yes / no)**. **Briefly explain** your answer.

[1.5 marks]

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

int sum = 0;
void * calc(void * number_of_increments)
{ int i;
  for(i = 0; i < *((int*) number_of_increments);i++) {
    sum++;
    sched_yield();
  }
}
int main()
{ int iterations = 500000;
  pthread_t tid1,tid2;
  if(pthread_create(&tid1, NULL, calc, (void *) &iterations) == -1) {
    printf("unable to create thread");
    exit(0);
  }
  if(pthread_create(&tid2, NULL, calc, (void *) &iterations) == -1) {
    printf("unable to create thread");
    exit(0);
  }
  pthread_join(tid1,NULL);
  pthread_join(tid2,NULL);
  printf("the value of sum is: %d\n", sum);
}
```

Figure 2: Incrementing a shared counter.

e. Assume that threads A and B are competing for a shared (mutually exclusive) resource. Are deadlocks due to inversion of control possible on:

   i. A Windows 7 system with A and B running as "realtime" threads **(yes / no)**?

   [0.5 marks]

   ii. A Windows 7 system with A and B running as "variable" threads **(yes / no)**?

   [0.5 marks]

   iii. A Linux system using a "completely fair scheduler" with A and B running as "time sharing tasks" **(yes / no)**

   [0.5 marks]

f. This question is about process and thread scheduling. It assumes the latest version of the Windows scheduler for which the documentation is available here. The characteristics of the processes and threads to schedule are listed in Table 1. The relevant process and thread priorities are listed in Table 2. You can assume that:

   - The time starts at 0.
   - The FCFS order of the threads is A1, A2, B1, B2, C1.

- All processes / threads run on the same core.
- The overhead for context switching is negligible.
- The time slice is 20ms and is independent of the priority level.
- The priority boost interval is 500ms (the counter starts at time 0).
- The processes / threads do not require user input.

Answer the following questions:

i. **Illustrate** and **list** the times when the processes start running, finish running, and when the context switches take place.

[2 marks]

ii. Calculate the average response time for the different threads. **Show** your working.

[1 mark]

iii. Calculate the average turnaround time for the different threads. **Show** your working.

[1 mark]

Process	Process Priority Class	Thread	Thread Priority Class	Arrival Time (ms)	Burst Time (ms)
A	NORMAL_PRIORITY_CLASS	A1	THREAD_PRIORITY_NORMAL	0	110
		A2	THREAD_PRIORITY_NORMAL	0	90
B	BELOW_NORMAL_PRIORITY_CLASS	B1	THREAD_PRIORITY_HIGHEST	100	60
		B2	THREAD_PRIORITY_NORMAL	100	60
C	REALTIME_PRIORITY_CLASS	C1	THREAD_PRIORITY_TIME_CRITICAL	60	20

Table 1: Processes and threads to schedule.

Process Priority	Thraead Priority	Priority Level
BELOW_NORMAL_PRIORITY_CLASS	THREAD_PRIORITY_IDLE	1
	THREAD_PRIORITY_LOWEST	4
	THREAD_PRIORITY_BELOW_NORMAL	5
	THREAD_PRIORITY_NORMAL	6
	THREAD_PRIORITY_ABOVE_NORMAL	7
	THREAD_PRIORITY_HIGHEST	8
	THREAD_PRIORITY_TIME_CRITICAL	15
NORMAL_PRIORITY_CLASS	THREAD_PRIORITY_IDLE	1
	THREAD_PRIORITY_LOWEST	6
	THREAD_PRIORITY_BELOW_NORMAL	7
	THREAD_PRIORITY_NORMAL	8
	THREAD_PRIORITY_ABOVE_NORMAL	9
	THREAD_PRIORITY_HIGHEST	10
	THREAD_PRIORITY_TIME_CRITICAL	15
REALTIME_PRIORITY_CLASS	THREAD_PRIORITY_IDLE	16
	THREAD_PRIORITY_LOWEST	22
	THREAD_PRIORITY_BELOW_NORMAL	23
	THREAD_PRIORITY_NORMAL	24
	THREAD_PRIORITY_ABOVE_NORMAL	25
	THREAD_PRIORITY_HIGHEST	26
	THREAD_PRIORITY_TIME_CRITICAL	31

Table 2: Process and thread priority classes and levels.

Question 2 (Concurrency)

[**12.5 marks in total**]

a. This question relates to semaphores and mutexes:

   i. Mutexes in modern operating systems such as Linux (and others) are implemented using a hybrid spin-lock/sempahore approach. That is, a mutex follows three steps (in chronological order): (1) it tries to acquire the lock; (2) if that fails, it spins if the thread that "owns the lock" is running; (3) if that still fails, the thread goes to sleep. **Briefly explain** what the reason(s) might be for step 2 and step 3.

   [2 marks]

   ii. Can an operating system implement a generic semaphore by disabling interrupts **(yes / no)**? **Briefly explain** your answer.

   [1.5 marks]

   iii. Assume a single-core single CPU architecture and code that uses user threads. Can synchronisation of critical sections be achieved by disabling interrupts. **(yes / no)**? **Briefly explain** your answer.

   [1.5 marks]

b. The code in Figure 3 implements mutual exclusion between processes i and j using Peterson's solution. You can assume that memory access at the hardware level is elementary. Demonstrate (e.g. by giving an example) that a satisfactory solution for mutual exclusion cannot be obtained when:

   i. The turn variable is removed from the implementation?

   [1 mark]

   ii. The flag variables are removed from the implementation?

   [1 mark]

```
do {                                      do {
  flag[i] = true;                           flag[j] = true;
  turn = j;                                 turn = i;
  while (flag[j] && turn == j);             while (flag[i] && turn == i);

  // CRITICAL SECTION                       // CRITICAL SECTION

  flag[i] = false;                          flag[j] = false;
} while (...);                            } while (...);
```

Figure 3: Peterson's Solution

c. Consider the code in Figure 4. Will the `printf("The value of counter is: %d\n", *map)` instruction print a logically correct result on the screen **(yes / no)**? **Briefly explain** your answer.

[1.5 marks]

d. The code in Figure 5 shows a solution for the dining philosophers problem with maximum parallelism.

   i. Provide an example that demonstrates that the critical section in the `take_forks()` function is necessary.

   [1.5 mark]

   ii. What is the effect of moving the instruction `state[i. ] = THINKING` from 31 after the `test(right)` instruction on line 33.

   [1 mark]

e. The code in Figure 6 is for the readers before writers problem. Given that the critical section for the readers is short, would it be better to use a synchronisation approach based on busy waiting instead of the sync semaphore **(yes / no)**? **Briefly explain** your answer.

[1.5 marks]

```c
#include <stdio.h, stdlib.h>
#include <sys/shm.h, sys/mman.h, fcntl.h>
#include <semaphore.h>

#define SHARED_MEMORY_NAME "/G52OSCSHAREDMEMORYGDM123"
#define SIZE_OF_MEMORY sizeof(long)
#define NUMBER_OF_PROCESSES 4

int main() {
  sem_t s;
  sem_init(&s, 0, 1);
  int i, status, iFD;
  pid_t pid[NUMBER_OF_PROCESSES];

  if((iFD = shm_open( SHARED_MEMORY_NAME, O_RDWR|O_CREAT|IPC_CREAT, S_IRUSR | S_IWUSR)) == -1) {
    perror( "shm_open() failed");
    exit(1);
  }
  if(ftruncate(iFD, SIZE_OF_MEMORY) == -1) {
    perror( "ftruncate() failed");
    exit(1);
  }
  int * map = mmap( NULL, SIZE_OF_MEMORY, PROT_READ | PROT_WRITE, MAP_SHARED, iFD, 0);
  if (map == MAP_FAILED) {
    close(iFD);
    perror("Error mmap() failed");
    exit(1);
  }

  for(i = 0; i < NUMBER_OF_PROCESSES; i++) {
    pid[i] = fork();
    if(pid[i] < 0) {
      printf("Could not create process\n");
      exit(1);
    } else if(pid[i] == 0) {
      int i;
      for(i = 0; i < 10000;i++) {
        sem_wait(&s);
        *map = *map + 1;
        sem_post(&s);
      }
      exit(0);
    }
  }
  for(i = 0; i < NUMBER_OF_PROCESSES; i++)
    waitpid(pid[i], &status, WUNTRACED);
  printf("the value of counter is: %d\n", *map);
  if (munmap(map, SIZE_OF_MEMORY) == -1)
    perror("Error munmap()");
  shm_unlink( SHARED_MEMORY_NAME );
  shmctl(iFD, IPC_RMID, 0);
}
```

Figure 4: Incrementing a shared counter.

```c
#define N 5
#define THINKING 1
#define HUNGRY 2
#define EATING 3

int state[N] = {THINKING, THINKING, THINKING, THINKING, THINKING};
sem_t phil[N];  // sends philosopher to sleep
sem_t sync;

void test(int i) {
  int left = (i + N - 1) % N;
  int right = (i + 1) % N;
  if(state[i] == HUNGRY && state[left] != EATING && state[right] != EATING) {
    state[i] = EATING;
    sem_post(&phil[i]);
  }
}

void take_forks(int i) {
  sem_wait(&sync);
  state[i] = HUNGRY;
  test(i);
  sem_post(&sync);
  sem_wait(&phil[i]);
}

void put_forks(int i) {
  int left = (i + N - 1) % N;
  int right = (i + 1) % N;
  sem_wait(&sync);
  state[i] = THINKING;
  test(left);
  test(right);
  sem_post(&sync);
}

void * philosopher(void * id) {
  int i = *((int *) id);
  while(1) {
    printf("%d is thinking\n", i);
    take_forks(i);
    printf("%d is eating\n", i);
    put_forks(i);
  }
}
```

Figure 5: Parallel Philosophers

```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>

sem_t rwSync;
sem_t sSync;

int iReadCount = 0;

void * reader(void * arg) {
  while(1) {
    sem_wait(&sSync);
    iReadCount++;
    if(iReadCount == 1)
      sem_wait(&rwSync);
    sem_post(&sSync);

    printf("reading record\n");

    sem_wait(&sSync);
    iReadCount--;
    if(iReadCount == 0)
      sem_post(&rwSync);
    sem_post(&sSync);
  }
}

void * writer(void * writer) {
  while(1) {
    sem_wait(&rwSync);
    printf("writing\n");
    sem_post(&rwSync);
  }
}

int main() {
  sem_init(&rwSync, 0, 1);
  sem_init(&sSync, 0, 1);
  pthread_t tid1, tid2, tid3;
  pthread_create(&tid1, NULL, reader, (void *) NULL);
  pthread_create(&tid2, NULL, reader, (void *) NULL);
  pthread_create(&tid3, NULL, writer, (void *) NULL);
  pthread_join(tid1, NULL);
  pthread_join(tid2, NULL);
  pthread_join(tid3, NULL);
}
```

Figure 6: Readers before Writers

Question 3 (Memory Management)

[**12.5 marks in total**]

a. Consider a memory system for which the free list indicates the following blocks of memory (a) 10KB, (b) 26KB, (c) 12KB, (d) 27KB, (e) 30KB, (f) 9KB, (g) 16KB (in that order).

Assume that requests for blocks of memory of 12Kb, 9Kb, 25Kb and 7kb come in (in that order). Which partitions would be allocated for the first fit, next fit, best fit and worst fit algorithm? If a block of memory is not fully occupied by the request, the remainder remains in the same place in the free list.

[2 marks]

b. Fixed partitioning and paging both result in internal fragmentation. However, paging is favoured in modern memory systems. Why is this?

[1 mark]

c. Assume an operating system that uses virtual memory. It has 64 pages of 4096 bytes each, and 32 frames of 4096 bytes each.

i. How many addresses will exist for the virtual and physical memory.

[1 mark]

ii. How many bits will the virtual and physical addresses contain?

[1 mark]

iii. What is the maximum number of entries that an inverted page table may contain for the memory "architecture" above? Briefly explain your answer.

[1 mark]

d. Assume a system that has 5 physical frames. The order in which a process references its pages is given below. Illustrate your answer.

Page references: 5, 4, 1, 2, 7, 9, 5, 1, 3, 2, 9, 8, 1, 4, 8

i. How many page faults would occur when the first in first out page replacement algorithm is applied?

[2 marks]

ii. How many page faults would occur when the optimal replacement algorithm is applied?

[2 marks]

iii. For both algorithms, highlight where the page faults occur.

[1 mark]

e. With virtual memory, explain the role of a paging daemon, and why certain pages cannot simply be swapped out if they are not currently in use. What may need to happen before this?

[1.5 marks]

Question 4 (File Systems and virtualisation)

[**12.5 marks in total**]

a. Assume a 32-bit disk address space, a block size of 2 kilobytes, a hard drive of 200 gigabytes, and that the size of directory metadata is negligible. If we have an i-node structure with 10 direct block pointers, one single indirect pointer, one double indirect pointer and one triple indirect, how many disk blocks will be spent to keep track of the block locations of a file of 32 megabytes? Write down your calculations.

[3 marks]

b. Assuming a File System based on i-nodes, choose the true statement below, and explain why the others are false:

  i. If 'file' is a hard link that is located in the directory "/usr/student/", the OS will need to access 6 times to disk to open the i-node.
  ii. If 'file' is a symbolic link that is located in the directory "/usr/student/", the OS will need to access 6 times to disk to open the i-node.
  iii. Creating a symbolic link will not require the use of a new i-node.
  iv. Creating a hard link will imply the use of a new i-node.

[4 marks]

c. Consider a disk with 200 tracks, and the following sequence of requested tracks 50, 74, 21, 55, 182, 145, 152, 24, 164, 184. Calculate the number of tracks for the algorithms below. Assume that the disk starts at position 75 and is moving towards position 200 (i.e., 0 to 200). Show your calculations.

  i. Shortest seek time first

[2 marks]

  ii. Circular scan algorithm (i.e. the arm does not move to the boundaries of the disk if not required), moving in the up position

[2 marks]

d. Briefly explain:

  i. A reason why a company might be interested in virtualisation. What type of company do you think would most benefit from virtualisation?
  ii. The main approaches to Virtualisation.
  iii. The main differences between Hypervisors Type 1 and Type 2.

[1.5 marks]