UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

INFR10079 OPERATING SYSTEMS

Wednesday 12th May 2021

13:00 to 15:00

INSTRUCTIONS TO CANDIDATES

Answer any TWO of the three questions. If more than two questions are answered, only QUESTION 1 and QUESTION 2 will be marked.

All questions carry equal weight.

CALCULATORS MAY NOT BE USED IN THIS EXAMINATION

Year 3 Courses

Convener: D.Armstrong
External Examiners: J.Bowles, S.Rogers, H.Vandierendonck

THIS EXAMINATION WILL BE MARKED ANONYMOUSLY

1. (a) Describe the differences between:

   i. an *operating system* and a *hypervisor* (*virtual machine monitor*) [*2 marks*]

   ii. para-virtualisation and hardware virtualisation [*2 marks*]

   (b) What is the output of the following snippet of code? Write down what you expect to be printed to the console by the `printf()` functions, and explain your reasoning in detail.

   Assume that all library calls (`fork()`, `getpid()`, and `printf()`) return without any error, and that the main process has PID 1234. The OS will assign the next PIDs sequentially (at increments of one), and there are no other processes or threads running. [*5 marks*]

```
1  int main () {
2    int x = 1, y = 2;
3    int pid;
4
5    x = fork();
6    if (x != 0) {
7      printf("x = %d y = %d\n", x, y);
8      pid = getpid();
9      printf("I am process: %d\n", pid);
10   }
11
12   y = fork();
13   if (y != 0) {
14     printf("x = %d y = %d\n", x, y);
15     pid = getpid();
16     printf("I am process: %d\n", pid);
17   }
18
19   return 0;
20 }
```

(c) In general, multithreading improves the overall performance and efficiency of programs in execution.

i. Provide two examples where multithreading **improves** the performance of a single-threaded solution. Explain why each example can improve the performance. [*2 marks*]

ii. Provide two examples of programs where multithreading **fails** to improve on the performance of a single-threaded solution. Explain why each example **fails** to improve the performance. [*2 marks*]

iii. If an application contains 20% of code that is inherently serial, what will the maximum speed-up be if run on a multicore system with four processors? Explain your reasoning. [*3 marks*]

(d) Consider the following C declarations and function definition:

```
1  int global_positives = 0;
2
3  typedef struct list {
4    struct list *next;
5    double val;
6  } list;
7
8  void count_positives(list *l)
9  {
10   list *p;
11   for (p = l; p; p = p->next)
12     if (p->val > 0.0)
13       ++global_positives;
14 }
```

Now consider the scenario in which there are two threads: A and B.

Thread A performs:

```
1  count_positives(<list containing only one positive value>);
```

Thread B performs:

```
1  ++global_positives;
```

Considering the possible implementations of the `++` operator (either atomic or not), describe the possible values that `global_positives` may assume **after** the execution, based on the interleaving of the two different threads. [*4 marks*]

(e) From the following code snippet, identify a sequence of code lines that when executed may result in a deadlock. Write this sequence as a list of line numbers. Explain why this sequence causes the deadlock to occur.

Note: There are multiple sequences that may cause a deadlock, you only have to correctly identify *one* such sequence.

Assume the `down()` and `up()` operations are atomic. The code has three semaphores: two counting semaphores (`full` and `empty`), and one mutex (`mutex`). Assume that the consumer starts executing first (from line 16). (This is an instance of the producer-consumer problem with one consumer task, and one producer task.) [*5 marks*]

```
1  const int N = 3;
2  semaphore full = N, empty = 0;
3  semaphore mutex = 1;
4
5  Producer Function:
6    while (1) {
7      produce an item A;
8      down(mutex);
9      down(full);
10     insert item;
11     up(mutex);
12     up(empty);
13   }
14
15 Consumer Function:
16   while (1) {
17     down(empty);
18     down(mutex);
19     remove item;
20     up(mutex);
21     up(full);
22     consume an item;
23   }
```

2. (a) Of I/O-bound, and CPU-bound programs:

   i. Which sort of program is more likely to incur voluntary context switches (i.e. purposefully give back control of the CPU to the OS), and which is more likely to incur involuntary context switches (i.e. the OS forcibly interrupts the program's execution)? Explain why this is the case. [*2 marks*]

   ii. What is the key advantage of concurrently executing I/O-bound and CPU-bound programs on the same machine? Please explain as well as providing an example. [*2 marks*]

   (b) Consider the following set of processes, with their corresponding execution times (in arbitrary time units) and priorities (the lower the number, the higher the priority):

Process	Execution Time	Priority
P1	5	4
P2	2	1
P3	1	2
P4	6	3
P5	3	3

   The processes are assumed to have arrived in the order P1, P2, P3, P4, P5, all at time 0.

   i. Draw five Gantt charts (with any software tool, taking a photo of a hand-drawing is also fine) that illustrate the execution of these processes using the following scheduling algorithms:

   - *First-come-first-served* (FCFS)
   - *Non-preemptive shortest job first* (SJF)
   - *Preemptive shortest job first* (SJF)
   - *Round-robin* (with a time quantum of 2 units)
   - *Non-preemptive priority*

   [*5 marks*]

   ii. For each of the scheduling algorithms in 2(b)i, what is the turnaround time of each process? [*2 marks*]

   iii. For each of the scheduling algorithms in 2(b)i, what is the waiting time of each process? [*2 marks*]

   iv. Which of the algorithms in 2(b)i results in the minimum average waiting time (over all processes)? [*2 marks*]

(c) The following processes are being scheduled using a preemptive, priority-based, round-robin scheduling algorithm – i.e. multilevel queue scheduling.

Process	Priority	Execution time	Arrival
P1	8	15	0
P2	2	25	0
P3	4	20	20
P4	4	20	25
P5	6	5	45
P6	6	20	55

Each process is assigned a numerical priority, with a lower number indicating a higher relative priority. The scheduler will execute the highest priority process first. For processes with the same priority, a round-robin scheduler will be used with a time quantum of 10 units. If a process is preempted by a higher priority process, the preempted process is placed at the end of the queue.

Draw a Gantt chart that shows the scheduling order of these processes. [*5 marks*]

(d) Mechanical disks (HDDs), or simply disks, still play an important role in storage systems despite the introduction of SSDs. Consider a disk drive with 2,048 cylinders, numbered from 0 to 2,047. The request queue has the following composition:

```
1040 751 943 879 1364 1686 1234 660 1676 1987
```

If the current position is 1167, and the previous request was served at 1250, compute the total distance (in cylinders) that the disk arm would move for each of the following algorithms: FIFO, SSTF, SCAN, and C-SCAN scheduling. [*5 marks*]

3. (a) What is the difference between

   i. a *translation lookaside buffer (TLB) miss*, and a *page fault*? [*2 marks*]

   ii. a *virtual page*, and a *page frame*? [*2 marks*]

   (b) Suppose an OS implements virtual memory with paging. A virtual address is 32 bits long and a page comprises 2kB (2^{11} bytes). A hierarchical two-level page table is used. The first-level and second-level page tables have 1024 (2^{10}) and 2048 (2^{11}) entries respectively.

   If a process uses the first 2MB (2^{21}) of its virtual address space, how much space would be occupied by the page tables for that process, when:

   i. Each page table entry occupies 4 bytes? [*2 marks*]

   ii. Each page table entry occupies 8 bytes? [*2 marks*]

   (c) Consider a hypothetical machine with four pages of physical memory and seven pages of virtual memory (labelled `A`–`G`). Given the following page reference requests:

   `F E D C B A D B B A A G A B G D F F E`

   For both the FIFO and LRU page replacement policies:

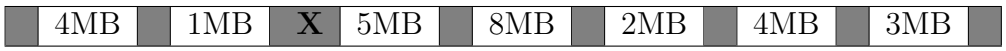
   i. Show the sequence of page swaps, and

   ii. compute the number of page faults. [*5 marks*]

(d) The diagram below shows an example of memory configuration under dynamic partitioning, after a number of placement and swapping-out operations have been carried out.

Addresses go from lowest to highest, left to right. Grey areas indicate blocks occupied by processes. White areas indicate free memory blocks.

The last process placed is 2 MB in size, and is marked with an X. Only one process was swapped out after that (thus, the memory is now free).



i. What was the maximum size of the swapped-out process? What was the size of the free block just before it was partitioned by X? [*2 marks*]

ii. A new 3 MB allocation request must be satisfied next. Indicate the intervals of memory where a partition will be created for the new process under the following four placement algorithms: best-fit, first-fit, next-fit, and worst-fit. For each algorithm, draw or clearly explain where the 3 MB of memory are going to be allocated. [*4 marks*]

(e) Suppose you have a file system where the block size is 2kB, a disk address is 4 bytes, and an i-node structure contains the disk addresses of: (a) 14 direct blocks, (b) a single indirect block, and (c) a double indirect block.

(In answering the following questions, you do not need to simplify arithmetic expressions, but you should show the math and explain your calculations.)

i. What is the largest file that can be represented by an i-node? [*3 marks*]

ii. Suppose you want to create a new file of the largest file size. How many free blocks on the disk would there need to be, in order to create that file? [*3 marks*]