Instructions:

1. No aids are permitted except non-programmable calculators with no persistent memory.
2. Turn off all communication devices. Communication devices must be stored with your personal items for the duration of the exam. Taking a communication device to a washroom break during this examination is not allowed and will be considered an academic offence.
3. Place all bags at the front or side of the examination room, or beneath your table, so they are inaccessible.
4. There are three (3) questions, each with multiple parts. Not all are equally difficult.
5. The exam lasts 60 minutes and there are 50 marks.
6. Verify that your name and student ID number is on the cover page and that your examination code appears on the bottom of each page of the examination booklet.
7. If you feel like you need to ask a question, know that the most likely answer is "Read the Question". No questions are permitted. If you find that a question requires clarification, proceed by clearly stating any reasonable assumptions necessary to complete the question. If your assumptions are reasonable, they will be taken into account during grading.
8. An API reference document is provided to accompany this exam. This will not be collected, so do not use it as extra space.
9. After reading and understanding the instructions, sign your name in the space provided below.

Signature

1 Concurrency & Processes [10 Marks Total]

1.1 Stack Storage [2 marks]

During process (thread) switch, we could use the Process (Thread) Control Block as a place to store the register data of the process (thread) being suspended. Your colleague proposes instead putting the register data on the stack of the thread being suspended. Would this solution work? Justify your answer.

1.2 Writers Board in Zone 1 [3 marks]

Part of being the operating system implementer is that it's possible to simply choose to give priority to certain classes of threads. Consider the pseudocode implementation of the readers-writers lock's unlock function below:

```
function rwlock_unlock( rwlock rwl )
  if ( rwl writer queue is empty AND rwl reader queue is empty )
    set light switch to 0
  else if ( rwl writer queue is empty and rwl reader queue is not empty )
    for each reader r in rwl reader queue
      unblock reader r
    end for
  else if ( rwl writer queue is not empty )
    unblock first writer in rwl writer queue
  end if
end function
```

Does the pseudocode above correctly implement the behaviour needed to have readers-writers locks where writers get priority (even at the risk of starving readers)? Explain your reasoning.

1.3 Sunrise [2 marks]

Following the principle of least privilege, system services that are not truly core to the kernel are started and run in user-mode, like regular users' processes. From a security and reliability perspective, why would it be a bad idea to run these in kernel mode?

1.4 Security [3 marks]

The operating system will validate, on a system call, whether the caller has the permissions to invoke the functionality. Recall that a system call like `read()` involves entering a library function which calls the trap instruction. Should the validation of permissions take place in the library or after the trap (in the kernel)? Justify your answer.

2 Memory [24 Marks Total]

2.1 Double Free [3 marks]

It is almost certain you have experienced what happens in a C program when the same memory is deallocated twice: the program crashes with an error message, immediately. Speculate as to how a double-free can be detected immediately; your answer should include your assumptions about how memory allocation data is maintained.

2.2 Dynamic Memory Allocation [5 marks]

Write down a pseudocode implementation of how to implement the *Best Fit* memory allocation routine. Your pseudocode needs to be sufficiently detailed and account for the scenarios where no sufficiently sized block is found (return NULL) and situations where there are multiple blocks of the same size (can return any).

2.3 Virtual Memory [6 marks]

Part 1. [4 marks] For this virtual memory scheme, at some point in time, we have the following page table for the process currently executing:

Page Table for Process Pn

Page #	Present	Frame #
0	1	3
1	0	–
2	1	8
3	1	4

Assuming the page table is in memory, complete the table below. To do so, determine the physical address corresponding to the virtual addresses. If the address is valid, write "OK" in the third column. If the address is valid but results in a page fault, write "Page Fault" in the third column. If the address is not valid and results in a segmentation fault, put N/A in the physical address and write "Segmentation Fault" in the third column.

Virtual Address	Physical Address	Evaluation
0x00001701		
0x00004886		
0x10001C41		
0x000033BC		

Part 2. [2 marks] In lecture, we discussed how page sizes are usually 4 KB as this is the block size of a hard drive. Suppose you bought a new generation hard drive and it has 8 KB block sizes. Show a diagram of how it a virtual memory address would be laid out if you moved to 8 KB pages.

2.4 Shared Memory [7 marks]

In ECE 252, we learned about shared memory as one inter-process communication mechanism. Explain how shared memory could be implemented in a 32-bit system with a virtual memory setup like that discussed in class. Your answer should include what happens on creation of a shared memory segment, attaching to and detaching from one, and deleting a segment. Your answer should also discuss the role of frames in this situation.

2.5 Caching [3 marks]

Modern 64-bit processors may have separate L1 caches for instructions and data. Explain why this is beneficial, making reference to what we know about memory access patterns.

3 Scheduling [16 Marks Total]

3.1 Fairness is an Opinion [6 marks]

Using the concept of *Fairness* as a scheduling criterion is vague, because what is "fair" depends on the needs of the system and users' opinion. Some might consider a purely Round-Robin scheduler to be fair, but maybe that is not the kind of fairness that we want for the department run servers for undergraduate courses (used by hundreds of students every term).

Part 1. Reasoning [3 marks] Give three reasons why a strict Round-Robin approach is not appropriate for the kind of system discussed above.

Part 2. Improvement [3 marks] Suppose you can make minor changes to the scheduling algorithm; suggest an improvement and justify why it would help. Note that your improvement does have to be *an improvement* and you cannot simply say replace Round-Robin with a different algorithm.

3.2 Lottery Scheduling [10 marks]

Recall that the scheduling algorithm of the lottery is based on the idea of choosing a random "ticket" that is used to identify which thread will get the resource right now. In this question, we will say there is only one resource (1 CPU) and each thread can have one ticket.

Threads that are ready to run are kept in a linked list that you need to manage. A ticket will be drawn in the range $[0, \texttt{max}]$ by calling `draw_ticket( int max )`; that ticket indicates the position in the linked list of the thread that is selected to run. If a thread is created, gets unblocked, or its timeslice expires, it will be added to the list with `add`. If a thread gets blocked or exits while running, no action is needed on our part in this implementation. If a thread is cancelled or killed while in the ready state, it is removed from the ready queue with `remove`.

You can assume the linked list is called `list` and has the properties `head, tail, size`. Elements in the list are called `node`, and each of those consists of a `thread_id tid` and a `node* next` pointer.

Write pseudocode implementations of the `scheduler` and `add` functions below to make it work as described.

```
/* Select the number in the range 0..max that indicates which thread in the ready queue */
int draw_ticket ( int max ) {
 // Implementation not shown -- you can assume it works correctly
}

/* Return the thread ID of the next thread selected to run */
thread_id scheduler( ) {









}

/* Add this thread id to the linked list (can add at front or back)*/
void add( thread_id id ) {









}
/* Remove this thread id from the linked list */
void remove( thread_id id ) {
        // Implementation not shown -- you can assume it works correctly.
}
```