The University of British Columbia

Midterm: Processes, Threads, and Synchronization – March 1st 2023

COSC 315, 2022W T2

ANSWER KEY – DO NOT SHARE

Closed book examination Time: 90 minutes

Last Name: _____ **First Name:** _____ **SID:** _____

Instructor: Reece Walsh

Special Instructions:

1. A cheat sheet can be referenced during this exam. The sheet must be a single sided, 8.5x11" piece of paper. Notes on this paper must be written by hand.
2. Explain your answers clearly and concisely. Do not write long essays.
3. Show your work. Partial credit is possible, but only if you show intermediate steps.
4. You've got this! Good luck.

Rules Governing Examination

- Each candidate must be prepared to produce, upon request, a UBCcard for identification.
- Candidates are not permitted to ask questions of the invigilators, except in cases of supposed errors or ambiguities in examination questions.
- No candidate shall be permitted to enter the examination room after the expiration of one-half hour from the scheduled starting time, or to leave during the first half hour of the examination
- Candidates suspected of any of the following, or similar, dishonest practices shall be immediately dismissed from the examination and shall be liable to disciplinary action.
  - Having at the place of writing any books, papers or memoranda, calculators, computers, sound or image players/recorders/transmitters (including telephones), or other memory aid devices, other than those authorized by the examiners.
  - Speaking or communicating with other candidates.
  - Purposely exposing written papers to the view of other candidates or imaging devices. The plea of accident or forgetfulness shall not be received.
- Candidates must not destroy or mutilate any examination material; must hand in all examination papers; and must not take any examination material from the examination room without permission of the invigilator.
- Candidates must follow any additional examination rules or directions communicated by the instructor or invigilator.

Page	Points	Max
2		6
3		4
4		13
5		7
6		10
7		10
8/9		20
Total		70

Points: _____ SID: _____

Section 1: Multiple Choice (10 points)

1 mark per question

B___ 1. What is an action that an OS can perform in kernel mode and not user mode?
(a) Run processes
(b) Use instructions to manipulate memory state
(c) Cause a trap
(d) Execute a system call

A___ 2. What is one main requirement for Banker's Algorithm?
(a) A fixed set of resources must exist
(b) All resources must be assigned
(c) Processes must be assigned to the same CPU core
(d) The algorithm must be non-pre-emptive

C___ 3. Select a significant difference between traps and interrupts from the OS's perspective.
(a) Traps can be generated by hardware or software while interrupts can only be generated by the hardware
(b) Traps can suddenly interrupt user code while interrupts predictably occur
(c) Interrupts can never occur from user code while traps can occur from user code
(d) Interrupts are always caused by user error

A___ 4. For valid atomic read-modify operations, multiprocessors must allow for:
(a) Cache invalidation on other cores
(b) Parallelization of busy waiting
(c) Multi-core value comparison
(d) Single-core operation to prevent race conditions

D___ 5. Which of the following options is not contained within a process control block?
a) Username of the process owner
b) Queue pointers for state queues
c) Scheduling information
d) File permissions

C___ 6. In a Unix system when a process is forked, what is the main difference between a child process and a parent process?
a) The child process takes parameters from the parent as input to initialize itself
b) The parent process receives a return code of zero from the fork call
c) The child process receives a return code of zero from the fork call
d) The child process receives the parent process's PID

Points: _____ SID: _____

A___ 7. The main purpose of a stack pointer is:
a) To keep track of last element on the stack
b) To keep track of data stored in a process
c) To track the current program address
d) To store the address for the return point of a function call

B___ 8. How many of the following points regarding semaphores and monitors are true:
- Monitors record a history of their usage
- Both semaphores and monitors require mutual exclusion to access state variables
- Semaphores can encounter a no-op on a wait call if a process is the first to use it

(a) 0 (b) 1 (c) 2 (d) 3

D___ 9. What is the first operation performed by the OS when a trap is detected?
(a) The trap vector is placed in the appropriate space in memory
(b) Control is transferred to the trap handler
(c) The trap vector is indexed
(d) The current process's state is saved

B___ 10. When using a shortest job first scheduling policy, one notable property is that:
(a) Only Pre-emptive schedulers can take advantage of this policy
(b) I/O-bound jobs are favoured
(c) It is optimal with respect to average completion time
(d) Only non-pre-emptive schedulers can take advantage of this policy

Points: _____ SID: _____

Section 2: Short Answer (30 points)

Write between one sentence and a paragraph in response to each of the following.

---

11. The OS uses a process control block (PCB) to maintain the state of a process and a thread control block (TCB) to maintain the state of a thread. What differentiates a PCB from a TCB, and what impact does this difference have on context switching? [8 points]

The following can be contained within a PCB and not a TCB [each difference counts for 1 mark up to a maximum of 4, slightly different wording referring to the same concept is acceptable]:

- Process ID, Process Group ID, Parent Process, Child Processes, heap pointer, system resource permissions, open files, process threads (an array of pointers to TCBs).

When context switching [+4 for something along the following lines]:
The main difference comes from a memory address space that all threads within a process share (e.g. a shared heap) vs. different memory address spaces individual processes do not share. Context switching between threads is faster than switching between processes mainly because the address space does not need to be changed.

[If the student mentions user-level threads and their differences (for the +3 marks above) this is fine as well and worth the same amount of marks]

12. To which type of jobs do scheduling policies for interactive systems give priority and why? [5 points]

Any of the following points can be included for marks, up to a maximum of 5. Responses can roughly match (within reason):

- Scheduling policies for interactive systems will typically give priority to shorter jobs from foreground or interactive processes [+2].
- Shorter jobs can include I/O operations, user interfaces, or other core components that contribute towards the responsiveness [+2, doesn't need to be exact].
- Shorter jobs are prioritized to ensure a responsive and interactive user experience. [+1]
- When a user interacts with an application, they expect it to respond quickly and smoothly. [+2]
- OS schedulers in interactive systems prioritize shorter jobs over long running jobs, since these are usually not immediately critical. [+2]

The main message should be that shorter jobs are prioritized to make user-facing actions responsive. Long running jobs are typically deprioritized to make way for immediate user needs in an interactive OS.

Points: _____ SID: _____

13. What is the purpose of synchronization? [3 points]

Marks for any of the following or similar, up to 3:

- Synchronization ensures that multiple processes or threads can coordinate on many or a single task while also using shared resources. [+1.5]
- Without synchronization, conflicts (such as a data race or deadlock) can cause issues when multiple processes or threads attempt to access shared resources at the same time. [+1.5]
- Locks, semaphores, and monitors ensure that access to shared resources is performed in a coordinated and efficient manner, ensure data integrity and preventing deadlock. [+1.5]

14. What advantage does the atomic instruction "test&set" have over atomic loads and stores? [4 points]

Marks for any of the following or similar, up to 4:

- test&set is a single atomic operation that combines a load and a store operation into one. Using test&set is simpler than implementing two separate atomic operations (a load and a store). [+2]
- test&set is typically faster than using separate load and store operations because it avoids the overhead of acquiring and releasing a lock for each operation. [+2]
- test&set is a low-level operation that provides direct access to hardware-level synchronization mechanisms. This takes advantage of hardware-level optimization and also makes it suitable for implementing low-level synchronization primitives. [+2]

Points: _____ SID: _____

15. Consider a variant of Round Robin scheduling in which the usual ready queue (in which PCBs are linked together) is replaced by a queue of pointers to PCBs.

a) What is the effect of putting two pointers to a PCB on the queue? [5 points]

Marks for something similar to the following points:

- A Round Robin scheduler attempts to fairly assign a time slice to the process at the front of the ready state queue. [+1 for remark about how Round Robin works]
- Putting two pointers to the same PCB on the modified ready state queue would effectively double the representation for the process [+2 for mentioning how this would affect the queue]
- If representation for a task is doubled on the ready state queue, the time a process can run is doubled, meaning that a process with two pointers would get two time slices worth of time to run on the CPU [+2]

b) What is one advantage and one disadvantage of this approach? [5 points]

Possible advantages [+2.5 max for any valid advantage]:

- Simplification of PCB management: A PCB's pointer only needs to be manipulated to access and move it from one queue to another.
- Context switching efficiency: Using pointers instead of copying the entire PCB increases efficiency during context switching. Moving a pointer is faster than moving a PCB.

Possible disadvantages [+2.5 max for any valid disadvantage]:

- Reduces fairness: Allowing for a process to be represented multiple times on the ready state queue undercuts the fairness of the round robin scheduling algorithm.
- Increased memory use: Pointers need to be created, thus, taking up extra memory. A pointer + the PCB is greater memory-wise than just the PCB.
- Increased processing time: When using pointers to examine a given PCB, the pointers need to be dereferenced to examine values.

The above list is not exhaustive. If a student provides a valid reason not listed above, marks can be given [Up to the 2.5 max for an advantage/disadvantage. No bonus marks].

Points: _____ SID: _____

Section 3: Long Answer (30 points)

---

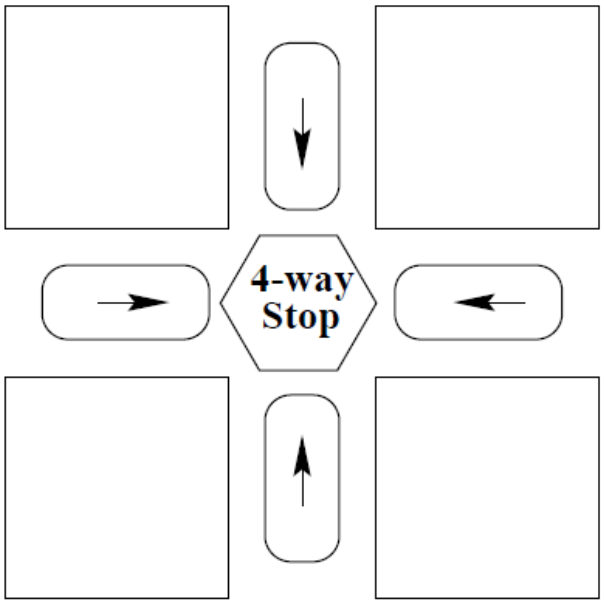
16. Using the fork(), waitpid(), exit() and kill() system calls, write a program in which a parent creates a child and the child creates a grandchild. The grandchild prints "I am a grandchild" and sleeps for 60 seconds. The child then kills the grandchild and exits. The parent waits for the child to finish and prints "Child finished" and exits. You may use pseudo-code to write your program. [10 points]

```c
int main() {
    pid_t cpid, w;
    int wstatus;
    int ret = fork(); // First fork +2
    if (ret == 0) { // Child detection +1
        int cret = fork(); // Second fork +2
        if (cret == 0) { // Grandchild detection +1
            // Correct grandchild implementation +1
            printf("I am a grandchild\n");
            fflush(stdout);
            sleep(60);
        } else { // Child
            // Child gives grandchild time +1
            sleep(1);
            // Child kills grandchild +1
            kill(cret, SIGKILL);
        }
        exit(0);
    } else { // Parent
        // Correct parent implementation +1
        waitpid(ret, &wstatus, 0);
        printf("Child finished\n");
    }
    return(0);
}
```

This image is a working implementation of the code described in the above question. Marks for each part are noted in the image. Students do not need to replicate this code exactly.

The following should be taken into account when marking:
-Pseudo code is okay for this question
-No deductions for syntax errors, within reason. Not including a semi-colon is fine, however, non-sensical or unreadable code is not
-No marks deducted for lack of error checking
-Noting the necessary imports is not needed for marks
-Students **do not** need to include fflush. No marks should be deducted if it is not present.

Points: _____ SID: _____



17. Consider this 4-way stop sign, with traffic from all four directions.

a) Does the rule "yield to the car on the right" prevents any synchronization issue? Why or why not? [4 points]

This rule does not prevent synchronization issues and can cause deadlock. [+1.5]

If all cars arrive at the same time, each car will be yielding to the one on the right, causing a circular dependency. [+2.5]

[The answer should mainly state that the rule doesn't prevent deadlock and explain why. Marks can be given as long as the explanation is reasonable and acknowledges the deadlock occurrence.]

b) What is the difference between Mesa and Hoare style monitors? [4 points]

Mesa-style: The thread that signals for a waiting thread keeps the lock. Waiting threads have to wait for the signalling thread to give up the lock and then contend for it. [+2]

Hoare-style: The thread that signals gives up the lock and the first waiting thread gets the lock. Once the waiting thread is done, the lock is given back to the signalling thread. [+2]

c) Instead of the traditional stop sign, you must design an electronic system based on Monitors that allows only one car to enter the intersection at once. [12 marks]

Below is the code skeleton for one possible solution. You must include the lock->Acquire and lock-> Release in each of your Monitor procedures. *D*:0...3 signifies the four directions from which a car can arrive. (Marks will not be deducted for syntax errors; pseudo-code version of solution is acceptable). Use the following page for your answers.

```
class IntersectionMonitor {
  public:
    void Arrive&Stop (d: 0..3);
    void Go (d: 0..3);
  private:
    lock = FREE: Monitor Lock;
    atLight[0..3] = False: Condition Variable;
}
Intersection::Arrive&Stop(d: 0..3){

}
Intersection::Go(d: 0..3){{

}
```

Implementation on next page…

Points: _____ SID: _____

```
IntersectionMonitor::Arrive&Stop(d: 0..3) {
    lock->Acquire(); // +2 for correct acquire/release position
    if (atLight[d]) { // +2 cond. var. check
                      // Wait if a vehicle is waiting in this
                      // direction.
                      // The above if cond. is Hoare-style.
                      // Could also be Mesa-style
                      // e.g. while (atLight[d]) {
        atLight[d]->Wait(lock); // +2 wait on cond. var.
    }
    atLight[d] = true; // Note that we're at the light
    lock->Release();
    Go(d); // Attempt to drive through the light
}
IntersectionMonitor::Go(d: 0..3) {
    lock->Acquire(); // +2 correct acquire/release position
    atLight[d] = false; // +1 Light reset

    // Assume the car has driven through at this point...

    // If there are any waiting cars, the system ensures
    // that at least one gets a signal to go +3
    if (atLight[(d+1)%4] || atLight[(d+3)%4] || atLight[(d+2)%4]) {
        // Loop until a car is found waiting at any direction
        while (!atLight[next_direction] &&
               !atLight[(next_direction+1)%4] &&
               !atLight[(next_direction+3)%4]) {
            next_direction = (next_direction+1)%4; // Move to the next direction
        }
        atLight[next_direction]->Signal(lock); // Signal the waiting car to proceed
    }
    lock->Release();
}
```

Main Idea:
When cars arrive and stop, the need to acquire the monitor lock and check if there are any cars waiting at the light already by using the condition variable. If a car is there, they wait. If not, they don't. After bypassing the condition variable check, they set the condition variable to true, release the lock, and attempt to Go.

When cars attempt to Go, the reacquire the condition variable and set the condition variable to false (signaling they went through the intersection). At this point, the Go function attempts to find any waiting cars and signal for them to go. After this, it releases the monitor lock.

Notes:

- The above is one possible solution. Try to fairly judge other possibly solutions.
- If students don't include the Go function in their arrive&stop, no marks deducted.
- Pseudo-code is acceptable (within reason). If you can't understand their logic or what they're trying to call, no marks.