NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

MIDTERM TEST – LUMINUS QUIZ

AY2020/21 Semester 1

CS2106 – INTRODUCTION TO OPERATING SYSTEMS

SOLUTIONS AND MARKING SCHEME

September 2020 Time Allowed: **1 hour**

Student Number:

Part A: MCQ questions.

Questions 1-22 from this section should be answered by circling the **letter** of the correct option out of the options provided. Select ONLY one choice.

1. [1 mark] Ubuntu 20.04 (on our SoC Compute Cluster nodes) is using a monolithic kernel.
   A. True.
   B. False.
   C. It has elements of both microkernel and monolithic kernel.
      Ubuntu has elements of microkernel, but it is using a monolithic kernel.

2. [1 mark] Process A creates a process B that runs an infinite loop. A performs the `wait()` system call. Considering the generic five-states process model, process A may transit between the following states:
   (i) Running -> Blocked
   (ii) Running -> Terminated
   (iii) Running -> Ready
   (iv) Blocked -> Running

   A. Only (i) is possible.
   B. Only (i) and (ii) are possible.
   C. Only (i) and (iii) are possible.
      Accepted because it is not clear if wait() has completed.
   D. Only (i), (ii), (iii) are possible.
   E. All (i), (ii), (iii) and (iv) are possible.

3. [1 mark] Process A calls `exec*()` and receives `SIGTERM` (at the same time). Considering the generic five-states process model, process A transits between the following states:
   A. Running -> Blocked
   B. Running -> Terminated
   C. Running -> Ready
   D. Blocked -> Terminated
   E. None of the options.

4. [1 mark] Assume a single processor (core) system and no simultaneous multi-threading (ignore this term if you do not know what it is). Consider the generic five-states process model. How many processes can be in BLOCKED state at the same time? How many processes can be in RUNNING state at the same time?

   A. Exactly one process in BLOCKED state and one process in RUNNING state.
   B. The operating system decides on the fly how many processes are in BLOCKED and RUNNING state.

C. Exactly one process is in RUNNING state, while there might be no process in BLOCKED state.

D. Any number of processes can be in BLOCKED state, but at most one process in RUNNING state.

E. Multiple processes can be in BLOCKED state and multiple processes can be in RUNNING state.

5. [2 marks] A program P has two functions `func1()` and `func2()`. `func2()` is invoked by `func1()`. Which of the following statements are TRUE regarding stack frames during `func2()`'s invocation by `func1()`.

   (i) `func1()`'s stack frame is torn down and `func2()`'s stack frame is setup instead.
   (ii) `func2()`'s stack frame is setup on top of `func1()`'s stack frame.
   (iii) `func1` ()'s PC, SP, and FP values are saved in `func1()`'s stack frame.
   (iv) Because `func1()` and `func2()` belong to the same program, their stack frame will have the same height (size in memory).

   A. Only (ii)
   B. Only (ii) and (iii)
   C. Only (ii) and (iv)
   D. Only (i) and (iii)
   E. Only (i) and (iv)

6. [2 marks] Consider two threads executing once the following C code, and accessing shared variables `a, b` and `c`:

Line#	Code Snippets
1	**Initialization**
2	`int a = 3;`
3	`int b = 0;`
4	`int c = 0;`
5	**Thread 1**
6	`b = 10;`
7	`if (a > 0)`
8	`c = b - a;`
9	
10	**Thread 2**
11	`a = -4;`
12	`if (a <= 0)`
13	`c = b + a;`

You can assume that each statement can be written using store and load operations (and these operations are not interrupted while running; they are atomic). Furthermore, the order of execution of statements within each thread is preserved by the C compiler and the hardware so that it matches the code above. What are the possible values for variable `c` **after both threads complete**?

A. -4, 6, 7, 14
B. 6, 14
C. -4 -3, 3, 4, 6, 7, 10, 14
D. -4 -3, 4, 6, 7, 14
E. None of the options.

7. [2 marks] Consider the following C code:

Line#	Code Snippets
1	`int i = 0, status;`
2	`if (fork()==0)`
3	`      i++;`
4	`else`
5	`      sleep(10);`
6	`wait(&status);`
	`printf("%d\n",i);`

What is printed by this code?

A. The program terminates and prints `0\n` only
B. The program terminates and prints `1\n` only
C. The program terminates and prints `1\n 0\n` only
D. Nothing is printed.
E. The program does not terminate and prints `0\n` only.

8. [1 marks] For a deadlock to occur, a program must meet certain conditions. Which of the following is **NOT** needed to observe a deadlock:

A. Only one thread can use a resource* at a time – accepted because formulation is not clear and can be misinterpreted.
A. refers to mutual exclusion, which is needed to observe a deadlock.
B. A resource can be held, then blocking whilst waiting for more resources
C. Resources cannot be forcibly taken away from process holding it
D. A circular chain of 2 or more processes, each of which are waiting for a resource held by the next member in the chain.
E. All options are needed for a deadlock to occur. – All A-D are needed to observe a deadlock.

* By resource we mean any type of shared variable protected by a lock, mutex, etc.

9. [1 marks] Suppose there are three processes P1, P2 and P3 where P1 accepts some user input and uses message passing to distribute the input to P2 and P3. P2 and P3 should wait for the input from P1, process the input and message the results back to P1. Finally, once P1 has received the results from both P2 and P3, P1 will print the results and exit. Which statements are TRUE regarding the above scenario?
(i) With a blocking `receive()`, the results from P2 and P3 can be printed in a pre-defined order.
(ii) A non-blocking `receive()` allows P2 and P3 to work on some other task (if any) while waiting to receive the user input from P1.
(iii) A blocking `send()` is required in P1 to enforce P2 and P3 to block themselves until the input is passed from P1.

A. Only (i) and (ii) are TRUE.
B. Only (i) and (iii) are TRUE.
C. Only (ii) and (iii) are TRUE.
D. All (i), (ii), and (iii) are TRUE.
E. All (i), (ii) and (iii) are TRUE.

10. [2 marks] Which of the following statements are FALSE regarding process context switching?
(i) Context switching always involves one process transitioning from READY state to RUNNING state.
(ii) Context switching always involves one process transitioning from RUNNING state to READY state.
(iii) Context switching always involves one process transitioning from RUNNING state to BLOCKED state.
(iv) Before switching, the current program counter (PC) should be saved in the PCB to restore the process execution when the process gets the CPU back.
(v) Disabling interrupts will disable context switching.

A. Only (ii), (iii), and (iv)
B. Only (iv) and (v)
C. Only (i), (iv) and (v)
D. All (i), (ii), (iii), (iv), and (v)
E. Only (ii) and (iii)
Since (v) is FALSE, there is no ideal choice in the list. Thus, award 2 marks to everyone. FALSE statements are (ii), (iii) and (v).

11. [1 mark] A signal handler is a special routine that is executed when a signal is received by a process. A signal handler cannot wait or signal a semaphore.
A. TRUE - sem_wait is not async_safe (according to man file)
B. FALSE
Marks awarded for everyone because question did not mention the scope. (i.e POSIX)

12. [1 mark] A thread can acquire more than one mutex (binary semaphore).
    A. TRUE
    B. FALSE
13. [1 mark] Synchronization problems can be avoided using:
    i) Message passing.
    ii) TestAndSet CPU instructions.
    iii) Semaphores.
    iv) Mutexes.
    v) Signals.

    A. Only (ii), (iii), (iv)
    B. Only (iii) or (iv)
    C. Only (iii)
    D. Only (i), (ii), (iii), (iv)
    E. All (i), (ii), (iii), (iv), and (v).

14. [2 marks] Assume that there is another system call `fork_exec()` provided to you that creates a new process and executes a new program referred by a pathname (`fork + exec` in one system call). Which advantage the new system call implementation might bring over the well-known `fork + exec` mechanism?
    A. The new `fork_exec()` system call incurs less overhead because no new memory has to be used for the new process.
    B. The new `fork_exec()` system call incurs less overhead because the memory for the new process had to be manipulated only once.
    C. The new `fork_exec()` system call gives more flexibility in process creation and execution.
    D. The new `fork_exec()` system call does not have any advantage over the traditional `fork + exec` mechanism.

15. [2 marks] In a terminal, `command1 && command2` has the following behavior: `command2` will be executed if (and only if) `command1` returns exit status zero. In a terminal, the following command is run:

```
> ls /a/b/c && ps
```

What is the minimum number of `fork()` calls that needs to be done by the terminal process to execute the command?

    A. Exactly 2 `fork()` calls.
    B. 1 or 2 `fork()` calls.
    C. 0, 1, or 2 `fork()` calls.
    D. Exactly 3 `fork()` calls.
    E. 1 or 2 or 3 `fork()` calls.

16. [2 marks] In a terminal, `command1 && command2` has the following behavior: `command2` will be executed if (and only if) `command1` returns exit status zero.
    In a terminal, the following command is run:

    ```
    > ls /a/b/c && ps
    ```

    Which statement is TRUE?

    A. The number of `exec*()` function calls is the same with the number of `wait/waitpid` calls made to execute the command.
    B. The output from `ls` command must be redirected to `ps` command standard input.
    C. The process executing `ls` must check the exit status of the process running `ps` before calling `exec`.

17. [1 mark] In batch processing, processes are allowed to do I/O operations.
    A. True
    B. False

18. [2 marks] Priority inversion is a common problem that appears in priority scheduling. Choose the statements that are TRUE:
    (i) Assume the lottery scheduling is implemented such that processes with higher priority receive more lottery tickets. However, lottery scheduling cannot suffer from priority inversion.
    (ii) A system with all jobs having exactly two priorities does not suffer from the priority inversion.
    (iii) A system with all jobs having the same priority does not suffer from the priority inversion.

    A. Only (i) and (iii)
    B. Only (ii)
    C. Only (ii) and (iii) – Priority inversion refers to a medium priority job (M) making progress in execution despite other higher priority job (H) waiting to progress. The higher priority job should be able to preempt the medium priority job. However, priority inversion appears when H is blocked waiting (on a resource) for a low priority job (L). M can progress and complete before H because L cannot preempt M to release the resource.
    D. Only (iii) – allowed because definition of priority inversion is not clear in the lecture notes.
    E. All (i), (ii), and (iii)

19. [1 marks] A process A spawns process B using `fork()`. B runs `exec*()`. Which statements are TRUE?
    (i) B forgets that A is its parent after the `exec*()` call because the image of the process is replaced.
    (ii) A forgets that B is its child, and B becomes an orphan.

(iii) The operating system will update the child information in the process A's PCB.

A. Only (i)

B. Only (ii)

C. Only (i) and (iii)

D. Only (iii) (Someone might consider what happens when both fork() and exec() is called. The intended focus of the question is only exec().)

E. None

20. [2 marks] How many processes will be created by the following code snippet? Assume `fork()` is successful. Note that `break` is used to exit from the smallest enclosing `for` loop, if any.

Line#	Code Snippets
1	`int i = 0;`
2	`for (i = 0;i<5;i++)`
3	`  if (fork()>0)`
4	`     break;`

A. 15 processes

B. 5 processes

C. 32 processes

D. It's a fork bomb

21. [1marks] In a round-robin scheduling policy, it is possible to have two processes (alternatively running on the same CPU) using different CPU time.

A. True

B. False

22. [2 marks] Assume you want to use the atomic `TestAndSet` instruction to synchronize among multiple threads (processes). `TestAndSet` exchanges data between a memory location (address) and a register. To make things faster, you are considering changing the `TestAndSet` implementation to exchange data between two general purpose registers (GPR) instead. Choose the TRUE statement about this approach.

A. The approach would work correctly, but it is not possible to create such an implementation.

B. The approach will work correctly, but it will not work faster.

C. The approach will work correctly, and it will run faster because the access to registers is faster.

D. The approach will not work correctly and cannot be used for synchronization.

Part B. Short Questions [13 marks]

Answer each of the following questions briefly (in one or two sentences). State your assumptions, if any.

23. [1 mark] Why are interrupts needed in the operating system? Answers that just define an interrupt will not receive marks.

    General Comments:
    Marks were awarded for answers that explained why interrupts are needed in general. Answers that described a concrete example were also awarded marks. (just giving the definition of interrupt is not sufficient)

    Some accepted answers:
    - To disrupt the execution flow
    - To get control for a user
    - For preemption
    - For fault handling
    - .....

24. [2 mark] We mentioned that scheduling is done separately for CPU and I/O devices. Explain the need for different schedulers to be used by the operating system.

    General comments:
    - Devices have different characteristics/properties; scheduling must be tailored for the characteristics of the device.
    - Devices operate independently and can be used simultaneously, thus they need separate queues such that processes can queue for each device separately.
    - With independent scheduling and a separate queue, we can account for the usage per device independently by processes. Eg. If a process uses a lot of CPU, that should not affect its IO priority, and vice versa.
    - The best answers focused on why IO devices are different or should be treated differently or independently from the CPU. The focus should be on the devices, not the processes' behaviour.
    - Using different schedulers or scheduler parameters for IO-intensive versus CPU-intensive processes is not what the question is asking about at all.
    - The claim "if CPU and IO were not scheduled separately, then it would not be able to run a process on the CPU and do IO simultaneously" some students made is not true.
    - There were badly phrased answers like "CPU needs X and IO needs Y" or "X is better for CPU and Y is better for IO". Marks were given with benefit of doubt if we could substitute "CPU/IO-bound processes" or "CPU/IO devices" to get an answer that makes sense, but most of the time that was not the case.

Marking Scheme for Q24: (Your answer has been given a category (2A, 2B etc) in Luminus Gradebook Remarks.)

2 marks

- **2A** - **Devices have different characteristics/properties**; scheduling can be **tailored for the characteristics of the device**
- **2B** - Devices **operate independently** and can be **used simultaneously**
- **2D** - So we can **account** for **different usage per device independently**

1 mark

- **1B** - "To allow for **different prioritization** of IO-bound and CPU-bound processes on **each device**".
- **1C** - Some reference to CPU and IO devices being different, but no/wrong reasoning as to why we should then have separate schedulers for them
- **1D** - Any answer implying that without separate scheduling, it would not be possible to use CPU and IO simultaneously. (Please note that separate scheduling isn't necessary to achieve simultaneous use of CPU and IO.)
- **1E** - Any answer talking about "scheduling devices" or devices needing different scheduler parameters. (Schedulers do not schedule devices, but rather they schedule **processes' use** of devices.)

0 marks

- **0A** - Any answer that merely restates the goals of an OS e.g. "so that we can use resources optimally / efficiently / without starvation / fairly / …".
- **0B** - Any answer that merely restates the question e.g. "IO and CPU scheduler have different algorithms / parameters / behaviour".
- **0D** - Any answer that merely states that CPU and IO devices are different in some manner without any further relevant explanation.
    - It is not sufficient to say that IO and CPU resources are independent
    - Two CPU cores are also independent, but we don't use entirely separate scheduling for multiple cores.
- **0E** - Any answer talking about using **different schedulers/scheduler parameters** for **IO-intensive vs CPU-intensive processes** (or otherwise different kinds of processes) e.g. "use X for IO-intensive processes, Y for CPU-intensive processes" / "CPU and IO-bound processes have different requirements/criteria".
    - The question is not talking about scheduling CPU and IO-bound processes using different schedulers, but rather about using **different schedulers** to schedule processes' **use of CPU** and **of devices**.
- **0G** - Any answer merely mentioning that processes may require different amounts of time on IO or CPU / the existence of CPU-bound / IO-bound processes.
    - Please elaborate on why this means we should schedule them separately. (E.g. answer 2D above)

- **0H** Any answer claiming that separating the scheduling will allow it to be run on multiple cores.
    - On multicore systems, OSes may run the same scheduler with the same queue on multiple cores with no problem.
- **0X** Answer is blank.
- **0Y** Could not make sense of the answer, or the answer is entirely irrelevant.

25. [1 marks] An application has two concurrent activities. Give one reason for choosing the use of two processes rather than two threads.

    General Comments:
    This question tested the understanding on the objective of using threads vs processes. Marks were given for highlighting any of the following points.

    - Protection
    - Isolation
    - Large memory space needed
    - Synchronization problems when using threads
    - Behaviour of threads in system calls (wait(), exec())
    - Non thread safe system calls
    - User threads result in blocking for the other user thread

    Answers that were not accepted:

    - Merely stating memory is independent for processes. Answers should have stated why independent memory is needed.
    - Stating that if one thread is blocked, the other thread will also be blocked. This is not true for kernel threads. If the assumption is made about use of user threads, marks were awarded.

26. [2 marks] System calls are used to transit from user mode to kernel mode. Give another (different) example of a situation when the execution transits from user to kernel mode.

    General Comments:
    Marks were awarded for correctly identifying a situation that the system transits from user mode to kernel mode apart from using system calls. Answers of which system calls were directly a part of the solution were not awarded marks.

    Common answers that were awarded full marks:
    - Context switch
    - When Interrupt/exception handling

- Handling signals (such as SIGTERM)
- In hybrid thread model, creating/binding/switching kernel threads

Partial marks were awarded for answers that hinted above expected answers but lacked clarity. Common unaccepted answers:

- Syscalls (this is the example given in the question)
- Library calls that result in syscalls
- Bare metal implementation of programs/libraries that bypass the OS
- Message passing (which directly involves syscalls)
- Creating shared memory
- Thread switching (unless the answers clearly explains with respect to kernel threads)
- Importing a library
- IO operations
- Process getting blocked
- Simultaneous multi-threading
- Working in safe mode (like Windows safe mode)
- Using sudo
- Traps (the question is actually asking for situations other than traps)

27. [3 marks] The following code sequence should write “This is an easy midterm!” to file CS2106.txt. What lines of code would you add? **You are not allowed to use additional functions for writing in a file** (`write`, `fprintf`, etc)**, but you can open files**. Write the missing code below. (points are not deducted for wrong syntax if the understanding is conveyed, and the right function calls are used).
Fill in TODO1 [2 marks] and TODO2 [1 mark] in the following code snippet.

Line#	Code Snippets

1
2
3
…
 	
10
11
…
	`int main()`
	`{`
	`    \\ TODO1 [2marks]: add missing code, if any`
	`    close(STDOUT);`
	`    open(“CS2106.txt”, O_WRONLY | O_CREAT | O_TRUNC,`
	`0644);`
	
	`    printf("This is an easy midterm!\n");`
	
	`    \\ TODO2 [1mark]: add cleanup code, if any`
	
	`    \\No need to restore STDOUT, it’s ok if done`
	
	`    return 0;`
	`}`

General Comments:

The idea is to close STDOUT and open CS2106.txt such that the most recently close file descriptor (STDOUT) is assigned to the newly opened file. Thereafter, any strings directed towards STDOUT are sent to the file. An alternate solution would be to use dup2() to replace STDOUT with a new file descriptor assigned to CS2106.txt. No need to do anything for cleanup (TODO2) to get full marks.

Partial marks were awarded for the following cases:

- Open a file but redirect STDOUT to this file incorrectly.
- Use unpermitted commands like write().
- Printf comes before mapping the file with STDOUT.

Those where did not answer part(i) correctly were not awarded marks For part (ii).

28. [4 marks] Consider the following GCC atomic function:
    `bool _sync_bool_compare_and_swap (int* ptr, int old, int new)`

Description: Atomically compare a referenced location `ptr` with a given value `old`. If equal, replace the contents of the location with a new value `new`, and return 1 (true), otherwise return 0 (false).

Implement a lock using `_sync_bool_compare_and_swap.` You are allowed to use busy waiting. Fill in TODO1[1 mark], TODO2[2 marks] and TODO3[1 mark] in the following code snippet.

Line#	Code Snippet
1	`void lock_init (int *lock){`
2	`\\ TODO1[1 mark]: add missing code, if any`
3	
…	`*lock = 1;`
	`}`
11	`void lock_acquire(int *lock){`
…	`\\ TODO2[2 marks]: add missing code, if any`
	`while(!_sync_book_compare_and_swap(lock, 1, 0))`
	`//busy wait while we acquire the lock.`
	`}`
21	`void unlock(int *lock){`
…	`\\ TODO3[1 mark]: add missing code, if any`
	`*lock = 1;`
	`}`

General comments:

The idea is to busy wait until compare_and_swap atomic function manages to swap the value to the function (int old) with the value in *lock. You may initialize *lock to a desired value and use their own convention. However, the values used must be consistent across all three parts in the question. Our model solution assumes *lock = 1 represents empty critical section and *lock = 0 represents occupied critical section.

Marking Scheme for Q28(i): (Your answer has been given a category (A, B etc) in Luminus Gradebook Remarks.)

0 mark:

A. Blank, almost blank answers or totally wrong answers
   - lock is not dereferenced correctly. (lock = 1, lock -> 1)

B. *lock is not initialized to a deterministic value.
C. *lock is not initialized.
D. Using while loop to check value instead of setting initial value.
E. Initializing a mutex or pipe instead.

1 mark:

A. Correct answers.
   - *lock = 1
   - compare_and_swap(lock, *lock, 1)

B. Minor mistakes (eg. int *lock = 1)

Marking Scheme for Q28(ii): (Your answer has been given a category (A, B etc) in Luminus Gradebook Remarks.)

0 mark:

A. Blank, almost blank answers or totally wrong answers
B. Implementation done without using compare_and_swap.
C. Not using atomic operations/ mutual exclusion not guaranteed.
D. Wrong syntax in while statement
   - while (compare_and_swap(&lock, 1, 0 == 0) { } ;

E. Wrong logic in while statement. Process cannot enter critical section even when its empty. Note that int old needs to 1 (representing empty C.S.) instead of 0.
   - while (compare_and_swap(lock, 0, 1) != 1)

F. Wrong logic in while statement. Negation operator '!', '== 0' or '!= 1' is not present.
   - while (compare_and_swap(lock, 1, 0)){ }

G. Using mutexes, signals or pipes to implement the answer.
H. *lock is not changed to a value representing occupied state of critical section. Mutual exclusion will not be guaranteed.
   - while(compare_and_swap(lock, 1, 1));

I. Running compare_and_swap only once without a while loop. (No busy wait)

1 mark:

A. Busy waiting is done in recursion instead of iteration. (Can result in stack overflow)
B. Answer has correct logic but unnecessarily calls additional functions
C. Passing *lock instead of lock as a parameter.
D. Using additional variables that are not part of method parameters. If understanding is shown in comments or extra initialization statements, 1 mark was awarded.
    - while (!compare_and_swap(old, temp, new));

2 marks:

A. Correct answer. Usage of '== 0' or '!= 1' instead of negation operator '!' is okay.
    - while(!compare_and_swap(lock, 1, 0))
    - while(compare_and_swap(lock, 1, 0) == 0);
    - while (true) { if (compare_and_swap(lock, 1, 0)) break; }
    - while(!compare_and_swap(lock, 1, 0)) { sleep(1); }

Marking Scheme for Q28(iii): (Your answer has been given a category (A, B etc) in Luminus Gradebook Remarks.)

0 mark:

A. Blank, almost blank answers or totally wrong answers
B. *lock value is not consistent with part (i).
C. Part (i) and part (ii) do not show consensus of which value of *lock represents empty state of critical section, setting a fixed value in part(iii) is not meaningful. So, a mark is not awarded.
D. Using mutexes, signals or pipes, etc to implement the answer. Not allowed.
E. Syntax error by not dereferencing lock correctly. (If no syntax errors in part (i) or (ii))

1 mark:

A. Correct answer.
    a. *lock = 1
    b. compare_and_swap(lock, 0, 1)
B. Wrong syntax in dereferencing lock but penalty is applied in part (i) or (ii)

--End of paper--