Duration: 1 hour 15 min

Examiner: A. Goel

Please fill your student number and name below and then read the instructions below carefully.

Student Number: |__|__|__|__|__|__|__|__|__|__|

First Name: |__|__|__|__|__|__|__|__|__|__|__|__|__|__|

Last Name: |__|__|__|__|__|__|__|__|__|__|__|__|__|__|

Instructions

Examination Aids: No examination aids are allowed.

Do not turn this page until you have received the signal to start.

Do not remove any sheets from this test book. Answer all questions in the space provided. No additional sheets are permitted. Use the last blank page as scratch space. This page will not be marked.

This exam consists of 5 questions on 8 pages (including this page). The value of each part of each question is indicated. The total value of all questions is 60.

For the written answers, explain your reasoning clearly. Be as brief and specific as possible. Clear, concise answers will be given higher marks than vague, wordy answers. Marks will be deducted for incorrect statements in an answer. Please write legibly!

Work independently.

MARKING GUIDE

Q1:	_______	(26)
Q2:	_______	(6)
Q3:	_______	(6)
Q4:	_______	(12)
Q5:	_______	(10)
TOTAL:	_______	(60)

Question 1. Short Questions [26 MARKS]

Part (a) [8 MARKS] Circle the operations that require privileged instructions.

1 OS switches kernel threads

2 OS performs context switch

3 OS loads a program on disk into memory

4 OS acquires blocking lock on uniprocessor

5 Program invokes system call

6 Program acquires spinlock on multiprocessor

7 Program releases spinlock on multiprocessor

8 Device hardware interrupts CPU

Part (b) [6 MARKS] A process invokes the following system call. Circle the system call if the OS may move the process to the waiting state.

1 It invokes the read() system call

2 It invokes the getpid() system call

3 It invokes down() semaphore system call

4 It invokes up() semaphore system call

5 It invokes the thread_yield() system call

6 It invokes the kill() system call

Part (c) [6 MARKS] Circle the statements that are true of the Unix execve system call.

1 When the child starts, it uses the stack pointer value of the parent

2 Child's program counter is initialized to the start of the program

3 Child reuses the heap area of the parent

4 Child is assigned a new process ID

5 The system call never returns to the caller

6 Parent can pass a variable number of arguments to the child

Part (d) [6 MARKS] Consider a task implemented as either 2 single-threaded processes (Processes), or 2 kernel-level threads in a single process (Threads). For each of the following operations, indicate whether the operation will complete faster, or if the feature is easier to provide, with the Processes or with the Threads implementation by circling one of them. If both options are equally good, circle Both. Assume that the task is running on a 2-core processor.

Concurrently reading data from two different sections of a file	Processes / Threads / Both
Switching between the threads of the task	Processes / Threads / Both
Synchronizing threads using spinlocks	Processes / Threads / Both
Reducing memory corruption bugs	Processes / Threads / Both
Execution time for the matrix multiply operation	Processes / Threads / Both
Accessing variables on the stack	Processes / Threads / Both

Question 2. System Calls [6 MARKS]

The C code shown below invokes the `read` system call. We have also shown the corresponding x86-64 assembly code. Explain the purpose of each line of the assembly code shown below. If a C variable is being accessed at a line, mention the variable.

```
int
main()
{
        int ret;
        char buffer[20];
        int fd;

        ...
        ...
        ret = read(fd, buffer, 20);
        ...
}
```

```
1  lea      -0x20(%rbp),%rsi
2  mov      -0x4(%rbp),%edi
3  mov      $0x14,%edx
4  mov      $0x0,%eax
5  syscall
6  mov      %rax,-0x8(%rbp)
```

Line number	Explanation
1	load address of buffer variable into rsi register
2	load fd variable into edi register
3	load constant 20 into edx register
4	load 0, the system call number of read, into eax
5	invoke system/trap instruction to invoke system call
6	store return value of system call in ret variable

Question 3. Labs [6 MARKS]

Assume that the lab code shown below is run using preemptive threading. The program starts by running the `main` function. Also, recall that `thread_create` returns the thread ID of the newly created thread.

```
1  #define NTHREADS 16
2  static Tid wait[NTHREADS];
3  static int done = 0;
4
5  static void fn(int num) {
6          while (__sync_fetch_and_add(&done, 0) < 1) {}
7          if (num < (NTHREADS - 1)) {
8                  thread_wait(wait[num+1]);
9          }
10         unintr_printf("%d␣", num);
11 }
12
13 int main() {
14         for (long i = 0; i < NTHREADS; i++) {
15                 wait[i] = thread_create((void (*)(void *))fn, (void *)i);
16         }
17         __sync_fetch_and_add(&done, 1);
18         thread_wait(wait[0]);
19         unintr_printf("main␣");
20 }
```

Part (a) [1 MARK] What is the purpose of the `thread_wait` function?

the thread invoking thread_wait waits until the target thread exits

Part (b) [3 MARKS] Show the output of the program above. If multiple outputs are possible, show any two different outputs.

15 14 13 ... 2 1 0 main

Part (c) [2 MARKS] If you remove the `while` loop in the `fn` function (line 6), you find that the program produces different outputs. Clearly explain why this is the case.

A thread may invoke thread_wait on the next thread (with id = wait[num + 1]) before the next thread has been created (in the main thread). Then thread_wait will return an error (which is ignored), and this thread will print its num value before the next thread runs. So the order shown in Part (b) will not be maintained.

Question 4. Synchronization [12 MARKS]

You sketch an implementation of the `down()` and `up()` semaphore operations for a preemptive thread scheduler as follows:

```
1  semaphore sem = 0;
2
3  down(sem) {
4    disable interrupts;
5    if (sem->count <= 0) {
6      thread_sleep(sem);
7    }
8    sem->count--;
9    enable interrupts;
10 }
11
12 up(sem) {
13   disable interrupts;
14   sem->count++;
15   thread_wakeup(sem); // wake up one thread
16   enable interrupts;
17 }
```

In the table below, each row shows the execution of one or more threads. The notation `T1:5` means that Thread 1 executes Line 5. Only the lines (and all the lines) in which a thread accesses `sem->count` in the `down()` and `up()` operations are shown (Lines 5, 8, 14). For each row, indicate one of the three following options: 1) **Okay**: down() and up() work as expected, 2) **Problem**: down() and up() don't work as expected, and 3) **Not Possible**: this execution is not possible. Assume that the execution shown in each row is run independently of the execution shown in any other row. For each answer, provide a **short** explanation.

	Thread execution	Okay/Problem/Not Possible	Explanation
1	T1:5 T1:8	Not possible	sem starts at 0, T1 should sleep in Line 6 forever.
2	T1:5 T2:14 T1:8	Okay	T2 performs up, T1 is able to run down to completion.
3	T1:5 T2:14 T2:5 T1:8 T2:8	Not possible	When T2:5 runs, sem->count is 1, so T2 does not sleep, and so T1 should not be able to run since interrupts are still disabled
4	T1:5 T2:14 T2:5 T2:8 T1:8	Problem	T1 and T2 both finish down, so sem->count < 0 (this happens because Line 5 has "if" instead of "while")
5	T1:5 T2:14 T3:5 T3:8 T1:8	Problem	T1 and T3 both finish down, so sem->count < 0 (this happens because Line 5 has "if" instead of "while")
6	T1:5 T2:5 T3:14 T1:8 T2:8	Not possible	Only 1 wakeup (by T3), and two threads (T1 and T2) wake up from sleep.

Question 5. Scheduling [10 MARKS]

There are three threads, A, B and C in the system. Thread A is CPU-bound and always runnable. Threads B and C are IO-bound. After Thread B runs for 1 unit of time, it needs to block for 1 unit of time. After Thread C runs for 1 unit of time, it needs to block for 2 units of time.

You have been asked to implement a Unix-style feedback scheduler for your threads library. Your scheduler has a fixed time slice of 6 time units. At each time slice, the priority (p) of each thread is calculated as $p = p/2 + c$, where c is the CPU usage of the thread in the last time slice in terms of time units, and the scheduler runs a runnable thread with the smallest priority value.

At time unit 0, all threads are assigned a priority value of 0. When two threads have the same priority value, the scheduler prioritizes A over B and C, and B over C. Assumes that the IO-bound threads becomes runnable just before a time unit expires.

Part (a) [8 MARKS] Each box shown below is a unit of time, and each line shows a time slice. We have shown that Thread A is scheduled in the first time unit. Show how the threads are scheduled in the rest of the time units. On the right, show the priority values of each thread (PA, PB, PC) at the end of the time slice.

	1	2	3	4	5	6	PA	PB	PC
Time slice 1	A	A	A	A	A	A	6	0	0
Time slice 2	B	C	B	A	B	C	4	3	2
Time slice 3	B	A	C	B	A	C	4	3.5	3
Time slice 4	B	A	C	B	A	C	4	3.75	3.5

Part (b) [2 MARKS] If the three threads run for a long time, roughly what fraction of the CPU will each thread receive?

The threads will mostly run in the order (B A C B A C) each time slice, so each thread will consume roughly 1/3 of the CPU.
(the reason for the "mostly" is that the floating point operation for calculating the priority will eventually cause PB to become 4 (due to rounding in floating point arithmetic), at which point the order will change a bit, since A has higher priority than B).

[Use the space below for rough work.]