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: _____

Last Name: _____

First 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 4 questions on 7 pages (including this page). The value of each part of each question is indicated. The total value of all questions is 40.

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: _____ (10)

Q2: _____ (12)

Q3: _____ (10)

Q4: _____ (8)

TOTAL: _____ (40)

Question 1. Synchronization [10 MARKS]

Fairness is an important aspect of mutual exclusion and synchronization problems. One way to implement fairness is to use first-come, first served queuing. Below, we have provided the outline for two functions, `queue` and `dequeue`. You need to implement these functions. To simplify your work, we have already added locks and condition variable synchronization in the function. Assume that they have been initialized correctly. We have also provided a variable, `wc`, to indicate the count of waiters in the queue. Use this variable to implement the queuing functions. Make sure to write your code clearly in just the boxes that are provided.

```
// shared variables
int wc = 0; struct lock *waiters_lock; struct cv *waiters_cv;

void
queue()
{
        1:
        lock_acquire(waiters_lock);
        wc++;
        if (wc > 1) {

        cv_wait(waiters_cv, waiters_lock);
        }

        lock_release(waiters_lock);
        9:
}

void
dequeue()
{
        1:
        lock_acquire(waiters_lock);
        wc--;
        if (wc > 0) {   // this condition is not essential

        cv_signal(waiters_cv, waiters_lock);
        }

        lock_release(waiters_lock);
        9:
}
```

Question 2. System Calls [12 MARKS]

For each program shown below, we have listed five possible outputs. Circle all the outputs (zero, one or more) that may be produced by the program. For each wrong answer, 1/2 mark will be deducted. These programs are called `whatA`, `whatB`, `whatC` and `whatD`. Assume that all system calls execute successfully.

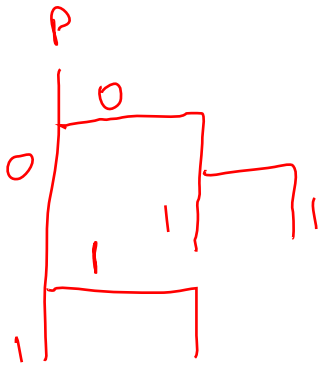
Part (a) [3 MARKS] whatA

```
int main() {
        int i = 0;
        char *argv[2] = { "./whatA", NULL };

        printf("%d ", i);
        i = i + 1;
        if (i >= 2)
                exit(0);
        execv(argv[0], argv);
}
```

A: 0 1 2
B: 0 0 0
C: 0 0 0 0 0 0 ... (forever)
D: 0 1 2 0 1 2 ... (forever)
E: none of the above

Part (b) [3 MARKS] whatB

```
int main() {
        int i = 0;

        while (i < 2) {
                fork();
                printf("%d ", i);
                i = i + 1;
        }
}
```



A: 0 1 0 1
B: 0 0 1 1 1 1
C: 0 1 0 1 1 1
D: 0 1 1 0 1 1
E: 0 1 1 1 0 1

Part (c) [3 MARKS] whatC

```
int main() {
        int i = 0;

        while (i < 2) {
                int pid = fork();
                if (pid)
                        waitpid(pid, NULL, 0);
                printf("%d ", i);
                i = i + 1;
        }
}
```

A: 0 1 0 1
B: 0 0 1 1 1 1
C: 0 1 0 1 1 1
(D): 0 1 1 0 1 1
E: 0 1 1 1 0 1

Part (d) [3 MARKS] whatD

```
int i = 0;

void sig(int num) {
        printf("-1 ");
        exit(0);
}

int main() {
        int pid;

        signal(SIGUSR1, sig);
        pid = fork();
        if (pid) {
                i = i + 1;
                kill(pid, SIGUSR1);
                waitpid(pid, NULL, 0);
        }
        printf("%d ", i);
}
```

(A): -1 1
(B): 0 1
C: 1 0
D: -1 0 1
(E): 0 -1 1

Question 3. Scheduling [10 MARKS]

There are four processes in the system. Two processes, A and B, are CPU-bound processes that are always runnable. The other two processes, X and Y, are IO-bound processes that run for 0.1 seconds and then block for 0.5 seconds. After blocking, they become runnable again.

Part (a) [5 MARKS] Show how the four processes are scheduled by a round-robin scheduler. The time slice of the scheduler is 0.1 seconds. Start by scheduling processes X, Y, A and B, as shown below. Show the process that is running in each time slice for the first 20 time slices. Assume that a process becomes runnable just before the arrival of a timer interrupt.

1	2	3	4	5	6	7	8	9	10
X	Y	A	B	A	B	A	X	B	Y

11	12	13	14	15	16	17	18	19	20
A	B	A	B	X	A	B	Y	A	B

Part (b) [5 MARKS] Show how the four processes are scheduled by a Unix-style feedback scheduler. Assume that timer interrupts arrive every 0.1 seconds and the time slice of the scheduler is 1 second. Start by scheduling processes X and Y as shown below. Assume that all processes have the same nice value.

Assuming X, Y, A, B all start at priority value = 0.

1	2	3	4	5	6	7	8	9	10
X	Y	A	A	A	A	A	A	A	A

11	12	13	14	15	16	17	18	19	20
B	B	B	B	B	B	B	B	B	B

Assuming A and B have been running for a while, their priority values will always be higher the priority values of X and Y.

X	Y	A	A	A	A	X	Y	A	A
B	B	X	Y	B	B	B	B	X	Y

Question 4. Hardware and OS Design [8 MARKS]

You plan to start a company that will design a simple, power-efficient processor for next generation smartphones. Your design must allow the mobile operating system to isolate processes from each other and protect itself from malicious processes. To do so, you are convinced that the simplest processor design requires only one privileged instruction. Convince your funding partner that your design will work.

Part (a) [2 MARKS] Describe exactly what operation the privileged instruction will perform in your processor.

The privileged instruction should be able to setup the MMU registers so that the OS can ensure memory protection.

There should be no IO instructions and all IO and interrupt access should be memory mapped.

Part (b) [4 MARKS] Explain how the OS will prevent the following:

1. A process directly modifies kernel data:

The OS uses the MMU to ensure that kernel memory is not mapped to user space, so the process cannot access kernel memory.

2. A process directly executes arbitrary kernel code:

The OS uses the MMU to ensure that kernel memory is not mapped to user space, so the process cannot access kernel memory.

3. A process writes directly to a network device:

IO access is memory mapped. This memory range lies in kernel memory.

4. A process disables interrupts:

Interrupt access is memory mapped. This memory range lies in kernel memory.

Part (c) [2 MARKS] Describe one way by which a malicious process may still be able to compromise the OS.

OS has bugs or vulnerabilities. A process can issue a system call with bad/crafted arguments, that cause the OS to crash, reveal information to user space, etc.

[Use the space below for rough work.]