Department of Computer Science
University of Toronto

Duration: 50 minutes
Examiners: Angela Demke Brown
Kuei (Jack) Sun

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

Student Number: 1 0 0 8 0 1 2 1 2 4

Last Name: Liu

First Name: Zexi (Jay)

Instructions

This is a **Type A** "close book" examination. No aids are permitted except for a non-programmable calculator (**Type 3**).

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

Please fill out the identification section above. Once you receive signal to start, make sure that your copy is complete. You may not remove any sheets from this test book. If you use any space (e.g. back of the page) for rough work, indicate clearly what you want marked.

This exam consists of 6 questions on 11 pages (including this page). The value of each part of each question is indicated. The total value of all questions is 85 marks.

For the written answers, be concise and write legibly. Answers that include both correct and incorrect or irrelevant statements will not receive full marks.

Work independently.

MARKING GUIDE

P1: 10 (18) AB

P2: 10 (12)

P3: 2 (9) A

P4: 8 (8) OT

P5: 2 (22) JZ

P6: 16. LH (16)

Total: 49 (85)

10

Part 1. True or False [18 marks]

Circle **T** if the statement is true, otherwise circle **F** if the statement is false. [2 marks each]

1. It is not possible to run more than one copy of a program that uses compile-time binding. ✓ **T** **F**

2. Limited direct execution refers to the limited amount of time for a user program to execute before the operating system performs a context switch. ✗ **T** **F**

3. A process can trap into the kernel by generating a hardware interrupt. ✓ **T** **F**

4. The bootloader is a small program that is stored in the BIOS. ✗ **T** **F**

5. Priority inheritance occurs when a low priority process using a lock is given the priority of a high priority process waiting for the same lock. ✓ **T** **F**

6. The wait operation for semaphores can suffer from the lost wakeup problem. ✓ **T** **F**

7. Once the requested data from disk is available, the requesting process will change state from blocked to running. ✓ **T** **F**

8. Without knowing anything about a new process, a MLFQ scheduler should place the new process in the highest priority queue. ✓ **T** **F**

9. Shortest job first scheduling suffers from the convoy effect. ✗ **T** **F**

Part 2. Multiple Answers [12 marks] 10

Circle all correct statements or answers. You lose 2 marks per wrong choice, down to 0 marks.

a) Which of the following are system calls, i.e., not C library functions: [4 marks]

i. `printf()`

ii. `fork()`

iii. `malloc()`

iv. `write()`

v. `exec()`

vi. `strlen()`

4

b) Which of the following statements about user-level threads are correct? [4 marks]

i. User-level threads can be implemented to perform context switches without system calls.

ii. User-level threads allow for custom scheduling algorithms specific to an application.

iii. When one user-level thread blocks, other user-level threads cannot run either. 2

iv. User-level threads outperform kernel-level threads in highly parallel programs.

v. User-level threads have lower memory and performance overhead than kernel-level threads.

c) Which of the following scheduling algorithms are not free from starvation? Assume all processes can end in a finite amount of time. [4 marks]

i. Shortest job first

ii. First-come first-served

iii. Round robin

iv. Priority scheduling

v. Lottery scheduling

4

Part 3. Short Answer [9 marks]

a) For a multi-threaded application, explain why disabling interrupt on multicore systems does not ensure atomicity and mutual exclusion. [3 marks]

Because multiple threads are running at once, so even if interrupts are disabled it's still possible for multiple already running threads to access the same critical ~~se~~ section.

X multicore ?

b) There are still other issues with disabling interrupt, even on a single core machine. Describe two of these issues. [6 marks]

Disabling interrupt will disable preemptive context switches, so poorly coded or malicious programs ~~can~~ may disable interrupt to hog up resources and starve other threads.

✓ 3

Part 4. Dynamic Partitioning [8 marks]

8/8

Given a dynamic partitioning scheme without compaction where memory is allocated in units of 1KB, the diagram below shows the current state of memory, where each allocation is given to a process (larger number represents more recent allocation, i.e., the last allocation was 4KB for process #5).

1	1					5	5	5	5						3	3	3		

Suppose process #6 requires 2KB and process #7 requires 3KB, fill in the allocated blocks for each process using the following algorithms:

1. First-fit algorithm. [2 marks]

1	1	6	6			5	5	5	5	7	7	7			3	3	3		

2. Next-fit algorithm. [2 marks]

1	1					5	5	5	5	6	6	7	7	7	3	3	3		

3. Best-fit algorithm. [2 marks]

1	1	7	7	7		5	5	5	5						3	3	3	6	6

4. Worst-fit algorithm. [2 marks]

1	1	7	7	7		5	5	5	5	6	6				3	3	3		

Part 5. Memory System [22 marks]

2^{10} size

Consider a 32-bit computer system that uses a two-level page table where the page size is 2KB. Answer the following questions and show detailed work for each question to receive full marks.

a) Suppose the page table entry size is 4 bytes and the page directory fits in one frame without unused bytes, how many page table entries will fit in a page directory? Answer in **decimal**. [2 marks]

32 bits

4 bytes

128 entries

2048 2^7 =

2048 entries

0

b) Based on the same assumptions as part (a) for the second-level page tables, fill out each of the boxes below. Answer each part in **bits**. [6 marks]

Virtual Address Layout:

Page Directory Index	Page Table Index	Page Offset
7	7	10

Size of virtual address space: $\boxed{2^{24}}$

0

Note: getting this question correct is essential to completing section (d) and (e)!

c) How much memory does a page table of this scheme take up when it is completely full? Answer this question in **kilobytes (KB)**. [4 marks]

~~256 · 256~~ 128·128

$2^7 \cdot 2^7 = 2^{14} = 2^4$ kBs = 16KB

0

Now, suppose the format of both the page table entry (PTE) and the page directory entry (PDE) for the two-level page table looks like this:

1 bit	10 bits	21 bits
Valid	Unknown	Frame number

Hex to Binary Table

0: 0000	1: 0001	2: 0010	3: 0011	4: 0100	5: 0101	6: 0110	7: 0111
8: 1000	9: 1001	a: 1010	b: 1011	c: 1100	d: 1101	e: 1110	f: 1111

With the partial memory dump shown at the bottom of this page, translate the virtual address: **0xd152fe**.

d) What's the frame number of the secondary page table, **in decimal**? [5 marks]

0x d 152 fe $\xrightarrow{bin}$ 1101 0001 0101 0010 1111 1110

~~64+32+8~~
104

e) What's the physical address for the virtual address 0xd152fe, **in hexadecimal**? [5 marks]

0x fe

Add row and column numbers to find PTE/PDE at an index. For example: `PD[9] = PD[8 + 1] → 0xf78b0fec`.

PAGE DIRECTORY:

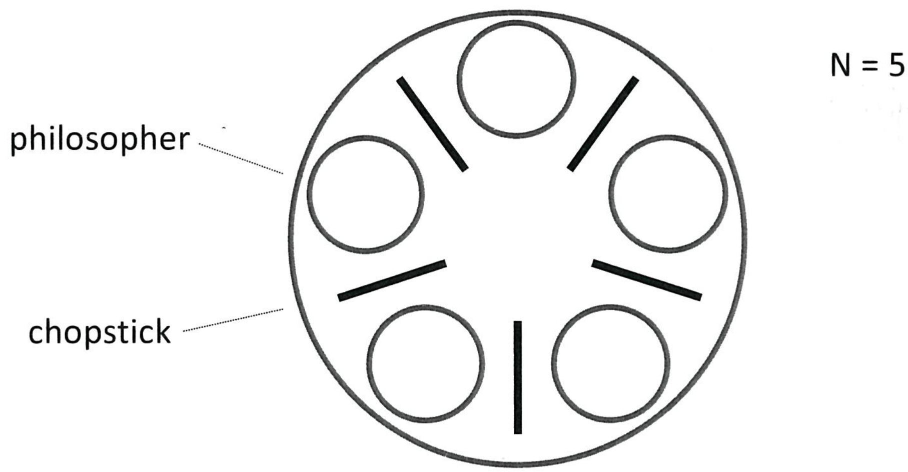
	0	1	2	3	4	5	6	7
0	b596d762	dd493d25	9e4174c9	90779654	240f6dba	af01faf4	a78ab1f4	4f5ec3d9
8	7e04cd56	f78b0fec	27eb359b	ee53dc39	de8dca76	8620081d	d782cce9	768f982a
16	5378ced3	ff43ff48	8ea8e1ff	b71c1cce	8779cba8	7d9d4d3c	5c53b031	7b32a8bd
24	0505d739	c0aaa149	09270c00	92641834	78108e18	5050da21	e6362fab	2c077168
32	88f66f24	0fbd686b	442ab1ec	ab3747e5	d5dc0ef7	9ce8bb83	e2f16feb	44b5193d
40	1d9f97ac	e002006f	db7e6dce	bf9e215e	b670d1ae	3ab7a82d	c87d4b64	817b25e3
48	9a37bd34	67d946fc	b01f6c7e	d74414e2	e6700951	cbb66c53	a88cce2c	3d837315

SECONDARY PAGE TABLE:

	0	1	2	3	4	5	6	7
0	a4910b23	a32b012e	b38d5705	2156a31d	c25f78f8	1a73962e	f40cb67b	27d6fb52
8	85fea745	221d89e6	8e4a13f6	12a36538	30475a41	b83d7db5	1bcafcbe	5b7b5510
16	62e09195	6d08ae36	de2829dc	59fe8ef1	acb7d7a6	dd98b2f4	82863048	9dfc5259
24	4906a1de	b7e5ad62	49a3860d	93438cb7	d7e696fe	5fe04498	6cee5cdd	966778c7
32	f99a77f6	74e59c66	fe6d8e1b	903874f0	4219d50a	95dcc1c3	f4658ef3	1a52bee4
40	70240f40	265188f6	bf8175b9	72b9a83c	ca3a54f0	141d24b0	0c926ebb	431fab5d
48	30fd0061	07f538b2	c8969b87	20cc27df	762ae217	8c96b81e	a02339e1	26933b9c

Part 6. Dining Philosophers [16 marks]

Dining philosophers is a classical synchronization and concurrency problem where there are *N* philosophers and *N* chopsticks on a round table. Each philosopher has access to one chopstick on the left-hand side, and another on the right-hand side. The philosophers alternate between two states: thinking and eating. In order to eat, a philosopher must obtain the two chopsticks, one on each side.



We model each philosopher as a thread, and each chopstick as a mutex object, like this:

```
struct lock * chopsticks[N];    // assume initialized in main()

void philosopher(int pid)       // pid will be between 0 and N-1, inclusive
{
    int left = pid;
    int right = (pid + 1) % N;

    while(1) {
        think();

        lock_acquire(chopsticks[left]);
        lock_acquire(chopsticks[right]);

        eat();

        lock_release(chopsticks[right]);
        lock_release(chopsticks[left]);
    }
}
```

a) There is currently the possibility of a deadlock in the above code. Suppose $N = 3$, list the sequence of *one* possible interleaving that will trigger deadlock. You must clearly state the order of every synchronization operation and show when context switches occur. [8 marks]

thread 0:

calls philosopher [0]

left = 0
right = 1
while(1){
think();
lock_acquire(chopsticks[0])

$\xrightarrow{\text{context switch}}$

thread 1:

calls philosopher(1)

left = 1
right = 2
while(1){
think();
lock_acquire(chopsticks[1])

$\xleftarrow{\text{context switch}}$

thread 2:

calls philosopher(2)

left = 2
right = 0
while (1){
think();
lock_acquire(chopsticks[2])

gets stuck on lock_acquire(chopsticks[0])

deadlock because all 3 locks are acquired and no threads can proceed.

8

b) Fix the deadlock in the original code in the spaces below. You will not lose marks for *minor* syntactical mistakes in your solution, as long as it does not affect semantics. However, you will lose **4 marks** if your solution fails to maintain the parallelism in the original code. [8 marks]

```
struct lock * chopsticks[N];     // assume initialized in main()

void philosopher(int pid) {      // pid will be between 0 and N-1, inclusive
    int left = pid;
    int right = (pid + 1) % N;
```

```
if (left > right){
    left = right;
    right = pid;
}
```

```
    while(1) {
        think();
```

```
lock_acquire(chopsticks [left]);
lock_acquire (chopsticks [right]);
```

8

```
        eat();
```

```
lock_release(chopsticks[right]);
lock_release ( chopsticks [left]);
```

```
    }
}
```

[*Use the space below for rough work*]

END OF EXAMINATION