

THE UNIVERSITY OF HONG KONG

FACULTY OF ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION SYSTEMS

CSIS0230A Principles of Operating Systems

Date: December 27, 2002

Time: 2:30 pm–5:30 pm

Candidates may use any calculator which fulfils the following criteria: (a) it should be self-contained, silent, battery-operated and pocket-sized; (b) it should have numeral-display facilities only and should be used only for the purpose of calculation; (c) it should not have any printing device, alphanumeric keyboard, or graphic display; and (d) it should not contain any recorded data or program. It is the candidate's responsibility to ensure that the calculator operates satisfactorily and the candidate must record the name and type of the calculator on the front page of the examination scripts. Lists of permitted/prohibited calculators will not be made available to candidates for reference, and the onus will be on the candidate to ensure that the calculator used will not be in violation of the criteria listed above.

This is an “open-book” examination. Be reminded that, in your answer, you may refer to materials of the course including (1) the textbook and reference book of the course, (2) sample solutions of any assignment and quiz, and (3) reading materials, worksheet and solutions of tutorials. If you do so, please clearly list the source of your reference together with the section, page and line number. You don't need to copy materials verbatim to the answer book.

Answer all questions in the answer book provided. Each of the 7 questions is worth 10%, and the marks of the 3 questions you score the most will be doubled.

1. (*Program execution*)

- a. (5%) Write a function `run_command_with_io` for Unix which takes 3 arguments: **char** * `name`, **int** * `in`, and **int** * `out`. The function should create two new file descriptors and put them into the space provided by `in` and `out`. Then it should run the program `name` with no additional command line argument, in such a way that the caller can write data to the new program through the `in` file descriptor and read data from it through the `out` file descriptor. Remember to clean up unneeded file descriptors.
- b. (5%) It is clear that sometimes we want to create threads rather than processes so that memory is shared and thus efficiency is improved. Explain 3 major reasons why sometimes we want the reverse, to create processes rather than threads.

2. (*Process synchronization*) Noticing that the reader-writer problem with writer priority requires a “no_writer” semaphore which can be modelled easily using a “condition” instead, a programmer decides to solve the problem using mutexes and conditions. He writes the following as the read function, which is just a translation of the original solution to use mutexes and conditions:

```
int readcount = 0, writecount = 0;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t wrt = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t no_writer = PTHREAD_COND_INITIALIZER;
void read() {
    pthread_mutex_lock(&mutex);
```

```
    if (writecount == 0) {
        ++readcount;
        if (readcount == 1)
            pthread_mutex_lock(&wrt);
        pthread_mutex_unlock(&mutex);
    } else {
        pthread_wait_cond(&no_writer, &mutex);
        pthread_mutex_unlock(&mutex);
    }
    /* perform the read here */
    pthread_mutex_lock(&mutex);
    if (--readcount == 0)
        pthread_mutex_unlock(&wrt);
    pthread_mutex_unlock(&mutex);
}
```

The write function is written similarly.

- a. (4%) By focusing on the differences between PThread mutex and a semaphore with an initial value of 1, explain why this doesn't work.
 - b. (6%) Write a correct implementation of reader-writer problem with writer priority using only the PThread mutex and condition constructs, without semaphores. You may write pseudo-code rather than actual programs. (Hint: the solution probably **won't** resemble the program that uses semaphores.)
3. (*Scheduling*) In an interactive system, we have the following CPU bursts to schedule:

Name	Start time	Amount of CPU time required
P	0	10
Q	1	10
R	2	5
S	3	4
T	6	5
U	8	10

- a. (6%) Show the schedule produced by a FIFO scheduler, an RR scheduler with time quantum of 2, and an SRPT scheduler.
 - b. (4%) In Linux implementation of the RR scheduler for non-realtime tasks (i.e., the conventional scheduler), the concept of **epoch** is used to keep track of the amount of CPU time usable by each task. What property of the scheduler makes this necessary? Explain.
4. (*Deadlocks*)
- a. (4%) To deal with occasional deadlock conditions, a program implements the banker's algorithm to keep track of resources usage. The program uses three types of resources, A, B and C. Each of them have multiple copies, in particular we have 5 A's, 4 B's and 4 C's. 5 processes P, Q, R, S and T are using the resources. At some time, a process

requests for some resources, which would result in the following:

Process	Resources allocated	Maximum amount needed
P	2B	4A, 2B, 1C
Q	1A, 1C	3A, 1B, 2C
R	2A, 1B, 1C	2A, 3B, 3C
S	None	2A, 4B
T	1A, 1C	1A, 2C

Explain why this is not a safe situation. What should happen after the system determines that the situation is not safe?

- b. (6%) The new Linux multiprocessor scheduler has a number of run-queues, one for each CPU. Spin-locks are used to ensure mutual exclusive access to the queues. At a number of different situations, multiple queues need to be locked. One such situation occurs as follows: one of the CPU finds that it has no process to execute, while there is a process in the queue of another CPU that is currently busy running another process. So it tries to “steal” the process for execution. To do this, it requires locking both the queue of the idling CPU, and the queue of the CPU with the process to steal. The pseudo-code for the locking is something like this:

```
/* we already have the lock for queue of process i */
/* and want to lock the queue of process j */
if (i < j)
    spin_lock(queue[j]);
else {
    spin_unlock(queue[i]);
    spin_lock(queue[j]);
    spin_lock(queue[i]);
}
/* now we have both locks */
if (queue[i] is not empty) {
    spin_unlock(queue[i]);
    spin_unlock(queue[j]);
    return; /* without stealing a process */
}
```

Explain why the procedure to acquire the additional lock depends on whether i is smaller than j . Also, explain why $queue[i]$ might not be empty after both locks are acquired.

5. (Memory management)

- a. (5%) Consider a memory system with 5 frames, initially empty. Pages are numbered between 1 to 10. Suppose the second-chance algorithm is used to control frame allocation. Show the events that occur when pages are requested in the following order:

1 2 1 3 4 5 6 2 3 7 8 7 2 9 10

- b. (5%) Consider the `init` program running in the i386 system, with the following map.

How much physical memory is used for page tables? Explain.

```
08048000-0804f000 r-xp 00000000 03:05 454802 /sbin/init
0804f000-08050000 rw-p 00006000 03:05 454802 /sbin/init
08050000-08054000 rwxp 00000000 00:00 0
40000000-40012000 r-xp 00000000 03:05 162043 /lib/ld-2.2.5.so
40012000-40013000 rw-p 00011000 03:05 162043 /lib/ld-2.2.5.so
40013000-40014000 rw-p 00000000 00:00 0
4001a000-4012b000 r-xp 00000000 03:05 162451 /lib/libc-2.2.5.so
4012b000-40131000 rw-p 00111000 03:05 162451 /lib/libc-2.2.5.so
40131000-40135000 rw-p 00000000 00:00 0
bffff000-c0000000 rwxp 00000000 00:00 0
```

6. (*Filesystems*)

- a. (5%) For a filesystem of 10GiB with block size of 1KiB and average file size of 4KiB using a simple filesystem like FAT32, roughly estimate the amount of filesystem storage space that is wasted due to fragmentation when the filesystem becomes full. Show your reasonings. Is the fragmentation internal or external? How the other type of fragmentation affects system performance?
- b. (5%) Given the block size of an ext2 filesystem, what other parameters of the filesystem determine each of the following characteristics of the filesystem? Explain your answers.
 - Number of blocks within each block group (except the last).
 - Size of block group in bytes.
 - Size of inode table.
 - Total number of block groups.

7. (*Protection and security*) Suppose, when implementing a system call similar to *sigaction* which allows a kernel data structure to be retrieved and modified, the kernel fails to check a user-supplied structure using *copy_from_user* before reading the content.

- a. (4%) This results in information leakage. Give detailed instructions on how to perform an exploit when we want to obtain certain memory content of a particular process.
- b. (3%) Can this defect (without any other flaw in the system) result in denial of service? How about unauthorized modification or deletion of information, and root access?
- c. (3%) Can these be exploited by a remote, unauthenticated intruder? Why?

END OF PAPER