PLEASE HAND IN

PLEASE HAND IN

**UNIVERSITY OF TORONTO**
**Computer Science Department**

**St. George Campus**

**July 2017 EXAMINATIONS**

**CSC 369H**
**Instructor — Sina Meraji**
**Duration — 2 hours**

**Student Number:** |_|_|_|_|_|_|_|_|_|_|

**Last (Family) Name(s):** _____

**First (Given) Name(s):** _____

---

*Do **not** turn this page until you have received the signal to start.*
(In the meantime, please fill out the identification section above, and read the instructions below *carefully*.)

---

This Term Test examination consists of **??** question on **??** page (including this one). *When you receive the signal to start, please make sure that your copy of the examination is complete.*

If you need more space for one of your solutions, use the last pages of the exam and indicate clearly the part of your work that should be marked.

In your written answers, be as specific as possible and explain your reasoning. Clear, concise answers will be given higher marks than vague, wordy answers. **Marks will be deducted for incorrect statements in an answer.** Please make your handwriting legible!

Marking Guide

| | |
|---|---|
| **??** | # 0: _____/**??** |
| | TOTAL: _____/**??** |

## Question 1. Short Questions [20 MARKS]

**Part (a)** [4 MARKS] what is a process and what is PCB?

process is a running program and PCB is a data structure associated with the process that ahs all data related to the process

**Part (b)** [4 MARKS] Write the Peterson' solution for critical section problem

check slide 45 from the second week



This exam continues on the following page

**Part (c)** [4 MARKS] What is the difference between Monitors and Semaphores?

with monitor we can only let 1 process to enter monitor and run in critical section. even if the entire code is not critical section other processes will be blocked. with semaphores we will have fine grain synchronization where we protect critical section before entrance and after entrance.

**Part (d)** [4 MARKS] briefly explain what does $REQUEST\_START\_MONITORING$ do in assignment1

you should know the answer :)

**Part (e)** [4 MARKS] Briefly Explain Second chance(clock) algorithm for memory page replacement.

put frames on clock, have a clock handle to point to oldest. check what is the oldest frame that has not been referenced and evict that page



This exam continues on the following page

## Question 2. Synchronization Problem [12 MARKS]

Some monkeys are trying to cross a ravine. A single rope traverses the ravine, and monkeys can cross hand-over-hand. Up to five monkeys can hang on the rope at any one time. If there are more than five, then the rope will break and they will all fall to their end. Also, if eastwardmoving monkeys encounter westward-moving monkeys, all will fall to their end.

Assume that monkeys are processes.

**Part (a)** [10 MARKS] Write a monitor with two methods $WaitUntilSafeToCross(Destionationdst)$ and $DoneWithCrossing(Destinationdst)$. Where Destination is an enumerator with value EAST=0 or WEST=1.



This exam continues on the following page

```
int crossing[2] = {0, 0}, waiting[2]= {0, 0};
Condition wantToCross[2];
WaitUntilSafeToCross(Destination dst)
{
if(crossing[!dst] > 0 || waiting[!dst] > 0 || crossing[dst] == 5)
{
++waiting[dst];
wantToCross[dst].wait();
--waiting[dst];
}
crossing[dst]++;
}
DoneWithCrossing(Destination dest)
{
--crossing[dst];
if (crossing[dst] == 0)
wantToCross[!dst].signal();
else if (waiting[dst] > 0 && waiting[!dst] == 0)
wantToCross[dst].signal();
}

Alternative implementation
int wcrossing=0, ecrossing=0, wwaiting=0, ewaiting=0;
Condition wantToCrossWest, wantToCrossEast;
WaitUntilSafeToCross(Destination dest)
{
if(dest == EAST)
WantToGoEast();
else
WantToGoWest();
}
DoneWithCrossing(Destination dest)
{
if(dest == EAST)
DoneGoingEast();
else
DoneGoingWest();
}
WantToGoEast()
{
if(wcrossing > 0 || wwaiting > 0 ||
ecrossing == 5)
{
++ewaiting;
wantToCrossEast.wait();
--ewaiting;
}
```



This exam continues on the following page

```
ecrossing++;
}
WantToGoWest()
{
if(ecrossing > 0 || ewaiting > 0 ||
wcrossing == 5)
{
++wwaiting;
wantToCrossWest.wait();
--wwaiting;
}
wcrossing++;
}
DoneGoingEast()
{
ecrossing--;
if (ecrossing == 0)
wantToCrossWest.signal();
else if (ewaiting>0 && wwaiting==0)
wantToCrossEast.signal();
}
DoneGoingWest()
{
wcrossing--;
if (wcrossing == 0)
wantToCrossEast.signal();
else if (wwaiting > 0 && ewaiting==0)
wantToCrossWest.signal();
}
```

**Part (b)** [2 MARKS] Does your solution suffer from starvation? If so, briefly explain (e.g. give a sequence). Otherwise, simply state starvation-free. In either, case state your assumptions, if any.



This exam continues on the following page

## Question 3. Scheduling [12 MARKS]

For normal processes, Linux uses a credit-based, preemptive, prioritized scheduling algorithm. Each process is associated with a counter of credits. The credits are initialized to the process priority when it enters the system, and the credits are reduced by 1 each time the quantum expires while the process is running. The process with the most credits left is chosen to run. (If the scheduler must choose between processes with the same number of credits, the one waiting the longest is run first; if more than one has been waiting the same amount of time, it chooses the one with the lowest process ID.) When no runnable process has any credits left (i.e. all processes with non-zero credits are blocked, say on I/O), the credits of all processes are increased by the formula $credits = (credits/2) + priority$ (truncating the fraction).

**Part (a)** [3 MARKS] How does this scheme tend to favor I/O-bound processes over CPU-bound ones? Why is this a good idea for Linux?

Because processes with the most credits are scheduled first, and because I/O-bound processes are more likely to gain credits over time, I/O bound processes become preferred scheduling choices. This is good for Linux, which is used both interactively via the keyboard and via the web as a server (either way, end users are I/O-bound).

**Part (b)** [3 MARKS] How does this scheme compare with the multi-level prioritized scheduling that we discussed in class? Are the goals the same? How are the mechanisms different?

The goals are the same, but the method of prioritization was different: queue level as opposed to priority number. The main difference is the criteria used to move to different priorities; the Linux approach is a bit more fine-grained.



This exam continues on the following page

**Part (c)** [6 MARKS] Linux also supports real-time scheduling, using a round-robin scheduler. If any real-time process is ready to run, the one with the highest priority is scheduled (a real-time process always is favored over a normal process). If multiple processes could run, the one that has waited the longest is chosen, else the one with the lowest process ID. Consider the following workload:

| Process | Priority | Burst Time |
|---|---|---|
| p1 | 2 | 6,6,4 |
| p2 | 1 | 2,2,3,2 |
| p3 | 1 | 16 |

explain how these processes would be scheduled using prioritized round-robin algorithm. You do not need to include context-switching time in your picture. Assume that all process arrive at the same time(time 0), the quantum is set to 5 time units, and that the time spent doing I/O between bursts is 1 time unit. For each algorithm, calculate the waiting time for each process.

Linux real-time round robin scheduling: P1(5), P1(1), P2(2), P1(5), P1(1), P3(5), P1(4), P2(2), P3(5), P2(3), P3(5), P2(2), P3(1).



This exam continues on the following page

## Question 4. Virtual Memory and Paging [15 MARKS]

**Part (a)** [7 MARKS] Suppose that we have a two-level page translation scheme with 4K-byte pages and 4-byte page table entries (includes a valid bit, a couple permission bits, and a pointer to another page/table entry). What is the format of a 32-bit virtual address? Sketch the paging architecture required to translate a 32-bit virtual address (ignore the TLB).



This exam continues on the following page

Check Slides 18-20 from week6

**Part (b)** [8 MARKS] Assume that a process has referenced a memory address not resident in physical memory (perhaps due to demand paging), walk us through the steps that the operating system will perform in order to handle the page fault. Note: We are only interested in operations that the OS performs and data structures modified by the OS. explain all the details

1. Trap to the OS 2. Save the process state (program counter, stack pointer, registers, etc) 3. Determine that the interrupt was a page fault 4. Check that the page reference was legal and determine the location of the page on disk In particular, if the reference was invalid, terminate the process. If it was valid, but have not yet brought in that page, page it in. 5. Find a free frame (by taking one from the free-frame list, for example) a. If there is a free frame, use it. b. If there is no free frame, use a page-replacement algorithm to select a victim frame. c. Write the victim frame to disk (if it was dirty)

6. Issue a read from the disk to the free frame: a. Wait in a queue for this device until the read request is serviced b. Wait for the device seek and/or latency time c. Begin the transfer of the page to the free frame 7. While waiting, allocate the CPU to some other process 8. Receive an interrupt from the disk I/O subsystem (I/O completed) 9. Save the registers and process state for the other process (if step 7 is executed) 9. Determine that the interrupt was from the disk 10. Correct the page table and other tables to show that the desired page is now in memory 11. Add the process to the ready queue 12. When process selected to run, restore user registers, process tate, and new page table, and then resume the interrupted instruction.



This exam continues on the following page

**Question 5.** Assignment1 Question [11 MARKS]

Assuming that we have following data structures for Assignment1.

```
#ifndef _COMMON_H
#define _COMMON_H

#define REQUEST_SYSCALL_INTERCEPT       1
#define REQUEST_SYSCALL_RELEASE         2
#define REQUEST_START_MONITORING        3
#define REQUEST_STOP_MONITORING         4

#define MY_CUSTOM_SYSCALL               0

#ifdef __KERNEL__

asmlinkage long my_syscall(int cmd, int syscall, int pid);

#define log_message(pid, syscall, arg1, arg2, arg3, arg4, arg5, arg6) \
        printk(KERN_DEBUG "[%x]%lx(%lx,%lx,%lx,%lx,%lx,%lx)\n", pid, \
                syscall, \
                arg1, arg2, arg3, arg4, arg5, arg6 \
        );
#endif


#endif /* _COMMON_H */


//----- System Call Table ------------------------------------
/* Symbol that allows access to the kernel system call table */
extern void* sys_call_table[];


//----- Data structures and bookkeeping ----------------------
/**
 * This block contains the data structures needed for keeping track of
 * intercepted system calls (including their original calls), pid monitoring
 * synchronization on shared data, etc.
 * It's highly unlikely that you will need any globals other than these.
 */

/* List structure - each intercepted syscall may have a list of monitored pids */
struct pid_list {
        pid_t pid;
        struct list_head list;
};
```



This exam continues on the following page

```
/* Store info about intercepted/replaced system calls */
typedef struct {

        /* Original system call */
        asmlinkage long (*f)(struct pt_regs);

        /* Status: 1=intercepted, 0=not intercepted */
        int intercepted;

        /* Are any PIDs being monitored for this syscall? */
        int monitored;
        /* List of monitored PIDs */
        int listcount;
        struct list_head my_list;
}mytable;

/* An entry for each system call */
mytable table[NR_syscalls+1];

/* Access to the table and pid lists must be synchronized */
spinlock_t pidlist_lock = SPIN_LOCK_UNLOCKED;
spinlock_t calltable_lock = SPIN_LOCK_UNLOCKED;
```

**Part (a)** [5 MARKS] Write a function to remove a pid from a system call's list of monitored pids. The function should returns -EINVAL if no such pid was found in the list.

```
static int del_pid_sysc(pid_t pid, int sysc) {
```



This exam continues on the following page

**Part (b)** [6 MARKS] Write a function to remove a pid from all the lists of monitored pids (for all intercepted syscalls). The function should return -1 if this process is not being monitored in any list

```
static int del_pid(pid_t pid) {
```



This exam continues on the following page