**EE 468 Final Exam Fall 2014 Total = 100 pts NAME:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Instructions:** This is a take home exam. You may use your book, notes, homeworks and projects and their solutions. You may use Microsoft Office applications such as Word, Excel, etc or equivalent. You may also look up information on the Internet but only from wikipedia and web sites that discuss Linux or Unix system calls. You are not allowed to discuss this exam with anyone, use message boards, or any type of communication. You are to submit your final exam by into laulima by December 18, 2013 Thursday at 11:55PM.

Problems 6 and 7 have two programs to modify or upgrade: sched.c and vm.c. You can find a version of these two programs attached to this exam as a tarred and gzippeed directory. The directory is named “Final”. It has subdirectories Sched and VM that contain sched.c and vm.c, respectively. The subdirectories also carry data files. When submitting, put README files in the subdirectories with instructions to compile and run. Your README file should have your name. Rename the directory “Final” to “<YourLastName>”. To submit your directory of upgraded programs, tar and gzip it.

Submit your final exam solutions and programs by uploading into laulima.

**Problem 1** [10 pts]. The following program will generate 100 child processes. The parent process will wait until all processes are done before printing “All child processes are done”. The program is incorrect. Modify it so it works properly.

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

int main( )

{

pid\_t pid;

int i;

for (i = 0; i < 100; i++) {

pid = fork( );

if (pid < 0) {

fprintf(stderr, “Fork Failed”);

return 1;

}

else if (pid == 0) {

printf(“I am child # %d\n”, i);

return 0;

}

else { /\* parent process \*/

wait(NULL);

printf(“All child processes are done \n”);

}

}

return 0;

} /\* end of main \*/

**Problem 2**. Consider the following variation of the Dining Philosophers Problem called the Dining Martians Problem. There are five martians, numbered 0, 1, ..., 4, sitting around a circular table. Initially, there are ten chopsticks in the middle of the table. Each martian needs exactly three chopsticks to eat. Whenever a Martian is hungry, it starts picking up chopsticks one at a time, where the time between picking up chopsticks can be of varying delay. A hungry martian keeps picking up chopsticks (and doesn’t share) until it has three, then it starts eating. When it’s finished eating, it puts all its chopsticks in the middle of the table.

**a**. [5 pts] Give a scenario which puts the martians in a deadlock situation.

**b**. [16 pts] Show how each of the following four necessary conditions for deadlock hold in this setting.

i. Mutual exclusion

ii. Hold and wait

iii. No preemption

iv. Circular wait

**c**. [10 pts] Assume that the martians have distinct numbers for identification. Consider the following additional rule for picking up chopsticks: When there is only one chopstick left, the martian that has two chopsticks and has the lowest number gets this last chopstick. Prove or disprove the martians are deadlock free. To disprove, show a deadlock example. To prove, show which of the necessary conditions for deadlock do not hold.

**Problem 3**. [10 pts] Consider a file system that uses inodes to represent files (Figure 12.9 on page 560 of the textbook shows an inode). Disk blocks are 2KB in size, and a pointer to a disk block requires 4 bytes. This file system has 12 direct disk blocks, as well as single, double, and triple indirect disk blocks. What is the maximum size of a file that can be stored in this file system.

**Problem 4** [9 pts]. Consider the following segment table

|  |  |  |
| --- | --- | --- |
| Segment | Base | Length |
| 0 | 1000 | 256 |
| 1 | 2000 | 100 |
| 2 | 3000 | 500 |
| 3 | 4000 | 200 |
| 4 | 5000 | 200 |

For the following logical addresses, indicate if it is valid and if it is valid, indicate is its physical address?

a. 1, 500:

b. 3, 120:

c. 2, 128:

**Problem 5** (20 pts). Find program sched.c which simulates a demand paging system with three frames for three replacement algorithms: FIFO, least recently used (LRU), and optimal. It gets a reference string from a data file scheddata (note that the contents of this file can be changed by the user). The format of the reference string is explained in the comments in the program. The algorithms are implemented by the functions fifo( ), lru( ), and optimal( ). Currently, only optimal( ) works. Fix the other two functions so they work too. Successful implementation of fifo( ) and lru( ) is each worth 10 points

**Problem 6** [20 pts]. This problem is essentially the Programming Project in Chapter 9 of the textbook on pages 458 through 461. Read these pages as well as the following:

We will assume that both the physical and virtual memories have size 216 which is 65,536. Other specifics include the following:

* 256 entries in the page table
* Page and frame size of 256 bytes
* 16 entries in the TLB
* TLB uses the least recently used (LRU) replacement policy
* 256 frames

Initially, all the data is stored on disk (e.g., a backing store) and the physical memory has empty frames. Frames are filled starting from frame 0 and then filling consecutive unfilled frames. To keep track of which unfilled frame to fill next, we only need a pointer that is initialized to point to frame 0. Then as each frame is filled the pointer increments by one.

To simplify the scenario, we will assume that the page table is not part of physical memory. An example scenario is when the actual physical memory is larger than 256 frames, but the number of frames allocated to a process is 256. The page table is stored outside of the allocated 256 frames.

Note that the frames and pages are the same size, and the number of frames and pages are the same. So you do not need to be concerned about page replacements during a page fault.

A C program “vm.c” is supposed to simulate the virtual memory manager program. It reads a file named “address.txt”, which is a list of virtual addresses. The program should run as follows:

./a.out

The program reads the virtual addresses from “address.txt”. For each virtual address, it outputs on a line

* the virtual address
* the corresponding physical address
* whether it resulted in a page fault
  + in particular, it outputs “PAGE-MISS” or “PAGE-HIT” for a fault or no-fault, respectively
* whether it resulted in a TLB hit
  + in particular, it outputs “TLB-HIT” or “TLB-MISS”

After all this is done, the program will output on separate lines

* the number of virtual addresses in the file “address.txt”
* the number of page faults
* the number of TLB hits

In the directory “Final” and subdirectory “Vm”, there is a partially working “vm.c”. Instead of a TLB with 16 entries, it has a TLB with one entry. Also, it doesn’t keep track or display the number of virtual address in the file address.txt, number of page faults, or the number of TLB hits.

Modify “vm.c” so that it works correctly. Make sure vm.c works on wiliki or Ubuntu. If you can’t get it to work then explain what went wrong or why it doesn’t work. Hint: to implement the LRU policy for the TLB, you need to keep track of each entry’s age.

Other comments: Note that vm.c just simulates the address translations and keeps track of when page faults and TLB misses occur.

Example: Suppose address.txt has the contents

1

256

32768

32769

128

65534

33153

The following are the results

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Virtual addr** | **Page #** | **Frame #** | **Page Fault** | **TLB** | **Page offset** | **Phys addr** |
| **1** | **0** | **0** | **YES** | **MISS** | **1** | **1** |
| **256** | **1** | **1** | **YES** | **MISS** | **0** | **256** |
| **32768** | **128** | **2** | **YES** | **MISS** | **0** | **512** |
| **32769** | **128** | **2** | **NO** | **HIT** | **1** | **513** |
| **128** | **0** | **0** | **NO** | **HIT** | **128** | **128** |
| **65534** | **255** | **3** | **YES** | **MISS** | **254** | **1022** |
| **33153** | **129** | **4** | **YES** | **MISS** | **129** | **1153** |

So running

./a.out address.txt

will output

1 1 PAGE-MISS TLB-MISS

256 256 PAGE-MISS TLB-MISS

32768 512 PAGE-MISS TLB-MISS

32769 513 PAGE-HIT TLB-HIT

128 128 PAGE-HIT TLB-HIT

65534 1022 PAGE-MISS TLB-MISS

33153 1153 PAGE-MISS TLB-MIS

Number of virtual addresses = 7

Number of page faults = 5

Number of TLB misses = 5