Carleton University
Department of Systems and Computer Engineering
SYSC 4001 Operating Systems Fall 2016

Assignment 3 – IPC, Threads, Virtual Memory, Files

POSTED: Tuesday, November 22nd 2016.

DUE DATE: Friday, December 9th (9:00 PM) 2016. No late assignments are accepted.

THIS ASSIGNMENT MUST BE DONE BY TEAMS OF TWO STUDENTS. Only ONE student must submit the assignment, clearly stating who the members of the team are. This programming part of the Assignment should be done using the Pair Programming technique. SUBMIT ONLY ONE ASSIGNMENT AND WRITE THE NAME OF THE TWO STUDENTS.

Part I – Concepts [60 marks]

Answer the following questions. **Justify your answers. Show all your work.**

0. [10 marks] Solve Part I. f) of Assignment 2 if you did not submit it yet.

1. Memory management [10 marks]

Consider a multiprogrammed system that uses multiple partitions (of variable size) for memory management. A linked list of holes called the *free list* is maintained by the operating system to keep track of the available memory in the system. At a given point in time, the free list consists of holes with sizes:

102K, 205K, 43K, 180K, 70K, 125K, 91K, and 150K

The free list is also ordered in the sequence given above: the first hole in the list is of size 102K words, which is followed by a hole of size 43K words and so on. There are a number of Jobs arriving to the system with different memory requirements; they arrive in the following order:

Job No.	Arrival Time	Memory Requirement (words)
1	t1	122K
2	t2	105K
3	t3	203K
4	t4	90K

[Given t1 < t2 < t3 < t4]

Determine which free partition will be allocated to each process for the following algorithms:
(a) First Fit (b) Best Fit (c) Worst Fit

Show all you work.

2. **[12 marks]** Consider the following page reference string in an Operating System with a Demand Paging memory management strategy:

201,302,203,404,302,201,205,206,302,201,302,203,207,206,203,302,201,302,203,206

(i) How many page faults will occur for the following page replacement algorithms when 3 frames are allocated to the program? Use
(a) LRU
(b) FIFO

(c) Optimal page replacement strategy.

(ii) Repeat (i) for 5 frames allocated to the program.

3**. [6 marks]** Consider a system with memory mapping done on a page basis and using a single-level page table. Assume that the necessary page table is always in memory and a memory reference takes 250ns.

a. How long does a paged memory reference take? **Explain** your answer.
b. If we add TLB that imposes an overhead of 30ns on a hit or a miss. If we assume that 80% of all memory references hit in the TLB, what is the effective memory access time? **Explain** your answer.
c. Why adding another layer, TLB, can improve performance? Are there situations where the performance may be worse with TLB than without TLB? **Explain.**

4**. [6 marks]** Consider a paged logical address space (composed of 32 pages of 2 Kbytes each) mapped into a 1-Mbyte physical memory space.

a. What is the format of the processor's logical address?
b. What is the length and width of the page table (disregarding all the control bits)?
c. What is the effect on the page table if the physical memory space is reduced by half? Assume that the number of page entries and page size stay the same.

5. **[4 marks]**

a. Explain what a Race condition is. Discuss a concrete example of a race condition.
b. Why disabling interrupts is not a good mechanism to prevent race conditions? Explain and justify.

6. **[2 marks]** Briefly explain how Semaphores work.

7. **[4 marks]** Explain, in detail, what an *open* operation must do. Consider the case of a file opened for APPEND, in a file system using a hierarchical directory structure.

8.

a) **[4 marks]** (from Silberschatz) Consider a file system that uses inodes to represent files. Disk blocks are 8Kb 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?

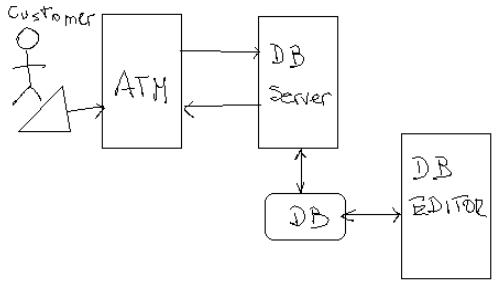
b) **[2 marks]** Explain what you can do in case (a) if you need to store a file that is larger than the maximum size computed. Give an example showing how you can define a larger file, and what the size of that file would be.

Part II – Programming [40 marks]

Problem description:

We will build a simple prototype of a program that mimics the work of an ATM system using concurrent processes and IPC services (message queues).

The system is organized in three concurrent processes, as shown in the following figure:



The Data Base (DB) contains information about the customers using the following format:

Account No.	Encoded PIN	Funds available
5 char	3 char	Float

The initial database for the system is the following:

00001,107,3443.22
00011,323,10089.97
00117,259,112.00

The ATM component does the following:

1. Requests an account number
2. Requests a PIN
3. Transmits these two values to the DB Server using a PIN message. If the DB server returns "NOT OK", restart from 1.
4. After the 3rd trial, "account is blocked" is displayed on screen.
5. If the DB server returns "OK", request the operation:
    a. display current funds: send a "Request Funds" message, waits for the funds available, and displays the number on screen
    b. funds to be withdrawn: sends a Withdraw message, together with the funds requested. If the funds available are not enough, the DB server returns a "Not Enough" message. In that case, "not enough funds" is displayed on screen. If there are enough funds, the DB Server responds with and "Enough" message. In that case, "Enough funds" is displayed on screen.

The ATM component then ends, waiting for the next customer (we will use it to terminate the program: if you use the string "X", the program ends.

The DB server does the following:

1. Wait messages forever
2. If the message is a PIN message, it gets the account number and the PIN. It searches the account number in the DB. It then subtracts 1 to the PIN number, and compares with the one stored in the DB. If the account number is found and the PIN is correct, return an "OK' message, and saves the information on the account in a local variable. If there is a failure, returns a "NOT OK" message. After the third attempt, the account is blocked. To do so, the first digit in the account number is converted to an "X" character.
3. If the message is a "Request Funds" message, get the Funds field, and return it.
4. If the message is a "Withdraw" message, get the amount requested, and check the funds available. If there is enough money into the account, return "Enough funds", decrement the funds available, and update the file. If there is not enough money, return "Not enough funds"
5. If an "Update DB" message is received, the updated information is obtained and saved to the file (adding 1 to each position of the PIN).

The DB Editor will:

1. Request an account number
2. Request a PIN.
3. Request a value representing the funds available
4. Send this information with an "Update DB" message to the DB.

Start by defining your system's architecture, showing the messages that you are going to use, the processes, how they are organized, and any other consideration (semaphores, shared memory, shared variables, etc.). This information will be useful to organize your work, and will be evaluated (together with comments in your source code and any other form of documentation provided).

Part A) [20 marks – OPTIONAL. You can submit Part B only, and obtain the 40 marks. Part B includes Part A. Submit Part A) only if you did not finish Part B successfully for partial marks]

Write a partial solution to the problem written in C/C++ running under Linux (only the DB server and the DB Editor). Create the initial DB discussed earlier using the DB editor.

Your solution must have two processes communicating using IPC. Other Linux system calls may be used for process creation, termination, creation and deletion of shared memory etc.

HINT: work incrementally.

Once this is working, add the ATM component executing concurrently. Your solution must have now three processes being synchronized using IPC services. You must guarantee mutual exclusion when accessing the database. To test this, open two windows (one for the ATM, and the second for the DB Editor) and check that the DB is properly updated while the ATM interface works properly.

Part B) [20 marks]

Convert the Program you wrote in Part A to a multi-threaded program using the PThreads library, where each process is now converted into a thread.

Marking Scheme:

We will evaluate

Correctness (including error checking): 70%
Documentation and input/output: 20%
Program structure: 5%
Style and readability: 5%

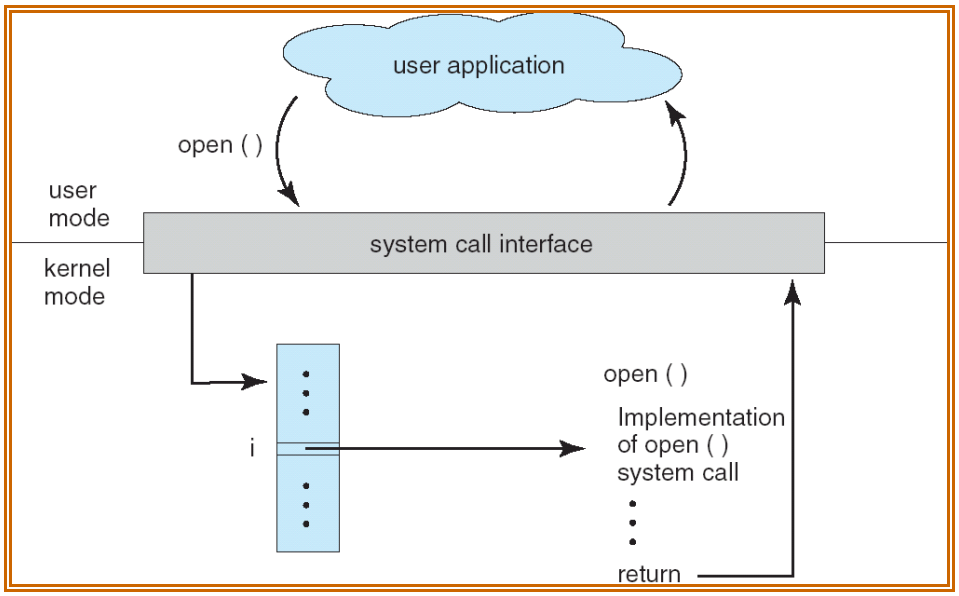
Programs not compiling or those with rude comments/text will receive a mark of 0 (zero).

Part III – Second part of the Term Project Take Home Exam (20 bonus marks)

IMPORTANT – FOR MARKING ORGANIZATION SEND AN EMAIL TO PROF WAINER IF YOU ARE PLANNING TO COMPLETE THIS OPTIONAL PART.

Combining the System Call Interface and the Scheduler simulators

The objective is to combine the two components of Assignment 2. Remember the following figure of the textbook.



This defines the system call interface to the OS. In Assignment 2, we used a simulation tool that provides you with predefined functions that we will use to model and simulate these components of the system. You built:

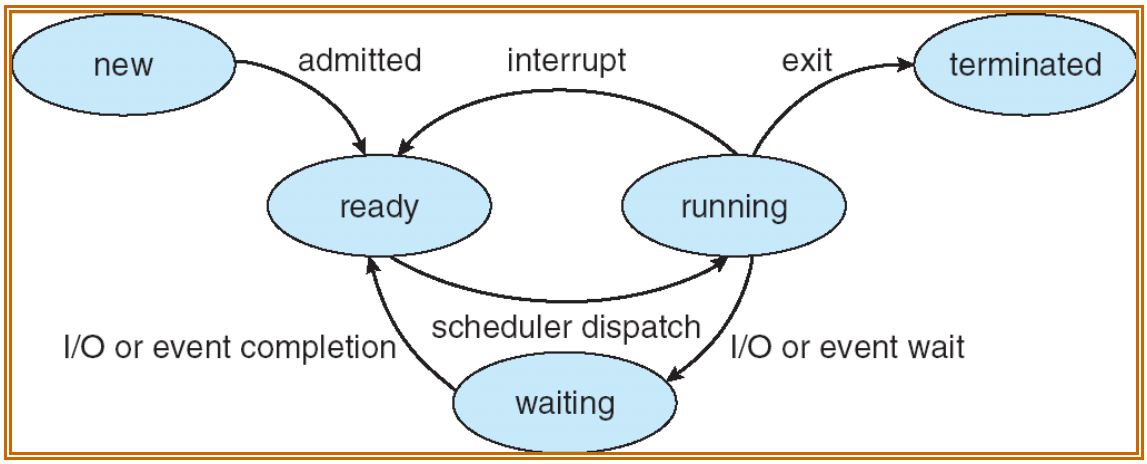
. A simulated "program". This program:
- "Execute" (random wait in the internal transition function)
- Issues a system call Read (in the output function)
- Waits for an answer (in the external transition function)
- NEW: issues a system call Write (in the output function).
- Repeat 5 times this cycle

. A simulated "system call interface". It:
- Receives 2 possible requests: Read or Write (in the external transition function)
- Transfers it to the lower layer (using the output function)
- Passivates, awaiting a response from the lower layer (another input in the external transition)
- Passes the results to the upper layer (using the output function)
- Internal states are changed by the internal transition. Timing will be programmed in the time advance function.

. Two simulated "system calls" components (based on the Receiver model in the ABP). They:
- Are activated (external transition)
- Simulate the activity ("read" or "write")
- Return (output function)

You also programmed a simulated kernel state diagram with a scheduling algorithm. You defined a data structure in memory similar to a PCB (only including the information needed: PID, CPU and I/O information, remaining CPU time – needed to represent preemption –).

The simulation reproduced the behavior of this state diagram (from Silberschatz et al.):



Note: the New and Terminated states are optional (you can simply load the process information in the Ready queue when they start.

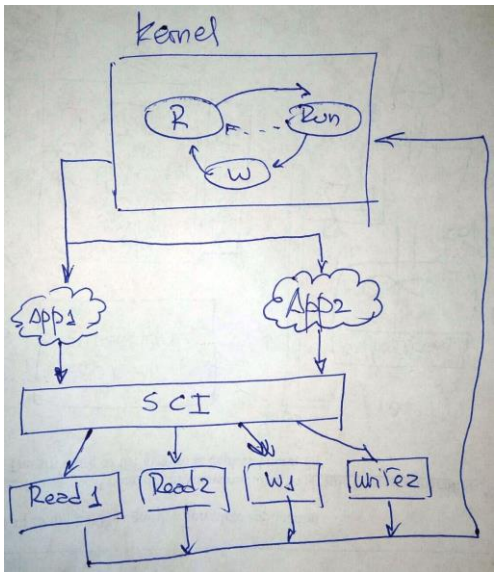
You completed a simulator of this kernel including a scheduler.

INTEGRATION OF THE TWO COMPONENTS:

ASSUME THAT THE TRANSITIONS ABOVE TAKE A VERY SMALL TIME (that is, do not worry for the time taken by the ISRs, scheduler, context switch, etc. When you program the time advance function, any small number will do).

ASSUME THAT YOU ALWAYS HAVE AN I/O DEVICE AVAILABLE (that is, do not worry about waiting queues at the I/O devices: whenever you request an I/O, the I/O starts immediately).

The parts above should be integrated as in the following figure:



You will build an atomic model called "Kernel" which will reproduce the behavior of the kernel above. For now, the "New" and "Terminated" states are not needed. Your *Kernel* atomic model will now have a table including a small PCB, including the following fields:

. PID
. State (Running, Ready, Waiting)
. Information about the process (in this case, it is a pointer to the App_i model)

At this stage, your PCB table will have only two entries. You should complete the two entries with information about two running applications (static information for each of the 2 processes). Initially, App_1 will use PID 1, and App_2 will use PID 2. Both will be in the state "Ready".

The two app models must be initialized to the *passive* state (passivate), and they will be activated by the Kernel model. The Kernel model will:

. Call the scheduler
. Change the state of the app to "Running"
. The app picked by the scheduler will be sent a message, which, in turn will activate its external transition. Then, the model must "run" (i.e., simulate the time executed) the "Execute" part of the model (described above).
. When the time is consumed and the app finishes simulating its execution (described above), it will issue a system call. That will produce a call to the SCI model, which, in turn will call the Read or Write submodels. The Read/Write simulated system calls will schedule an internal transition after a time (1 second of simulated time, for instance). Before this, the Read or Write models will inform the kernel that they have been called. The Kernel will receive this external transition and will change the state of the simulated process from Running to Waiting. Call the Scheduler.
. When the Read/Write models consume their simulation time (i.e., when the time advance has been consumed), they issue an output and an internal transition. The output function must inform the kernel that the simulated I/O has finished. The kernel will receive this information and will put this process in the ready queue. The Read/Write models will then passivate waiting for the next I/O request.

To make the model simpler, you can use as many Read/Write submodels as you want (assume that each of these models is directly connected to a specific I/O device, and that all of them are different).

Your implementation should use a FCFS scheduler (without preemption).

Run the simulation (remember that the two simulated apps run only 5 cycles, so, after 5 cycles each, the simulation should finish).

Analyze the simulation results. Show how the processes interact, change states, activate the I/O functions, and change from one state to the next one.

You should use the following link to download and install the software (you will need an installation of Linux to run the simulation software):

http://www.sce.carleton.ca/courses/sysc-5104/lib/exe/fetch.php?media=cdboost_example_installation_templates.zip

You must submit an executable, the source code, all files needed to compile, test scenarios, and scripts to run them. Include, at least, two scripts named "test1.bat" and "test2.bat" that will be used to run two different tests automatically. The TAs will use these two basic tests, and then will modify your input files.

Discuss the results you obtained based on your simulations. Write a report and attach it to the submission.

PAIR PROGRAMMING

We will be applying the industry's pair programming model. Assignments can be done individually or by a pair of students **following the pair programming practice.** Note that pair programming is **NOT**:

- Decompose the program and each do one part;
- One person does all of one program.

You must decide which option to choose. The decision cannot be changed for the whole term unless your partner drops the course or has special medical reasons.

Specific rules for pair programming for this course (adopted from University of Virginia, CS101 at originally at http://www.cs.virginia.edu/~cs101/s03/info/pairprog-info1.html):

You should plan to work with your partner spending most of your time in front of a computer together creating the program. While working together, you should switch back and forth between two roles, **driver** and **navigator**.

- The driver is the one typing code or writing down code or a design.
- The navigator is doing a number of things while looking carefully at what the driver is creating. S/he looks out for syntax errors, typos, using the wrong variable or function, or other logic errors. S/he also thinks about strategic problems, such as using the wrong approach, making the wrong assumptions, etc.

The driver and navigator should brainstorm about problems they encounter. They should communicate frequently. The navigator should ask the driver to explain what they are doing, or justify an approach, or say where they are headed with the solution they are creating. Good pairs say something to each other at least once a minute. Part of their discussion should be inspection and review of code that has been written to try to identify errors or other problems that one pair of eyes might not find.

Again, members of a pair alternate between roles. Swap off. Let the other person type, even if you are cruising down the path that the driver has devised.

Therefore, here are the "rules" if you want to work in pairs:

1. You must buy into the idea of pair programming, trying to carry out the work in the spirit of this method.
2. You must do most of your work together as a pair, swapping off between the driver and navigator roles. At least 60% of the total time you work on the program should be done as pairs and the main parts need to be designed and programmed together.
3. You are allowed to do some work alone, but each of you should spend no more than 20% of the total time spent on the program working alone.
4. The partner must review any work done by a solitary programmer.

DOCUMENTATION

You should use System V Message Queues for synchronization. If the information in the book, the tutorial and the manual pages are not enough, you can check on:

https://users.cs.cf.ac.uk/Dave.Marshall/C/node25.html

You can find a tutorial on PThreads here:

https://computing.llnl.gov/tutorials/pthreads/

Keep in mind that you ALWAYS have to double-check your manual pages, in case that there are slightly different values in your installed software.