Department of Computing and Information Systems
COMP30023: Computer Systems

Project 1

Due date: No later than 9:00am on Thursday, April 14.

Weight: 15%

Project Overview

The aim of this project is to increase your familiarity with issues in scheduling and memory management.

A number of simplifying assumptions related to scheduling and memory management have been made. We do this, so that the workload in the project is commensurate with the marks (and time commitment) available.

Your simulation program must be written in C. Submissions that do not compile and run on the School of Engineering student machines (eg. `dimefox` or `nutmeg`) may receive zero marks.

Part A - 12 marks

Assume there are 2 CPUs, the first one is dedicated to running the swapper and system code and can be ignored for the purposes of this simulation. The second CPU is used for running the (user) processes. The times required to do the swapping and scheduling are ignored in the simulation.

Scheduling algorithms

Two different queueing algorithms will be investigated as part of the simplified scheduling function in this project.

1. **First-come first-served queue**: The queue simply contains a list of all processes to be scheduled. The queue type corresponds to a non-preemptive algorithm. That is, once a process begins executing it continues executing until the total running time elapsed reaches the specified job-time.
2. **Multi-level feedback queue**: A simplified three-level round robin multi-level feedback queue contains a list of all process to be scheduled.
    - New processes are added to queue `Q1` with quantum = 2.
    - After exhausting their quantum, if additional CPU time is required the process is subsequently moved to queue `Q2` with quantum = 4.
    - If a process requires additional CPU time, it is moved to the final queue `Q3` with quantum = 8. The **round robin** mechanism is then used for scheduling until all processes have terminated.

Assume that a process continues executing until either its quantum expires or it terminates. A process terminates when the total running time has reached the specified job-time.

Memory

Assume that memory is partitioned into contiguous segments, where each segment is either occupied by a process or is a hole (a contiguous area of free memory). The free list is a list of all the holes. Holes in the free list are kept in *descending* order of memory address. Adjacent holes in the free list should be merged into a single hole.

When a process is moved from disk into memory, the **first fit algorithm** should be used. The first fit algorithm starts searching the free list from the beginning (highest address), and uses the first hole large enough to satisfy the request. If the hole is larger than necessary, it is split, with the process occupying the higher address range portion of the hole and the remainder being put on the free list.

Process data file

A process data file is a sequence of entries that describes a list of processes that are to be created in your simulation. The first entry refers to the first process that is created, and the last entry refers to the last process that is created. Each entry consists of a tuple (**time-created, process-id, memory-size, job-time**).

For example:

```
0 4 98 15
3 2 33 20
5 1 100 10
20 3 5 15
```

This models a list of created processes where process 4 is created at time 0, is 98 MB in size, and needs 15 seconds running time to finish; process 2 is created at time 3, is size 33 MB, and needs 20 seconds of time to get its job done; process 1 is created at time 5, is size 100 MB, and requires the CPU for 10 seconds; etc. Once created, processes begin their life on disk. (You do not have to worry about how processes are created in this simulation).

Points to note:

- The processes listed in the process data file are in ascending order based on the time that they were created.
- The first process is always created at time zero.
- Each process-id is a unique positive integer.
- A process-id also represents the priority of the process, where lower process-id indicates higher priority.
- Each process size is a positive integer $\leq m$ (the main memory size).

You may assume that if two (or more) processes are created at time `t`, they will appear in the process data file sorted by process-id. Thus, the issue of "priority" of a process will not cause any problems. (i.e. the process-data file will be sorted based on time-created, process-id).

You may also assume the input file being read in will always be in the correct format. You may also assume that the quantum values listed previously have the same units as the job-times listed in the process data file.

The simulation should behave as follows:

- Parse the process data file to obtain the lists of processes to be scheduled, then executed on the CPU, in your simulation.
- The schedule function (based on queue type) schedules which of the processes will execute next on the CPU. Only one process can execute on the CPU at any point in time.
- To use the CPU, the process must be in memory. Assume memory is initially empty.
- Use a timer/counter (starting from `time=0`) to control the overall simulation steps. At specific time steps, specific events will occur. For example, new processes arriving (based on the initial lists of processes read in from the process data file), processes moving from disk to memory, processes executing, processes terminating etc.
- At each time step, check the queue of processes to determine which process is to be run next. You should also check how long this process will run for (this will depend on the queue type used).
- If a process needs to be loaded into memory, use the first fit algorithm. If there is no hole large enough to fit the process, then processes should be swapped out, one by one, until there is a hole large enough to hold the process needing to be loaded.
- If a process needs to be swapped out, choose the one which has the largest size. If two processes have equal largest size, choose the one which has been in memory the longest (measured from the time it was most recently placed in memory).
- When a process has executed for the total running time specified by its job-time, its image is removed from memory and the process is terminated.
- The simulation should finish when all processes listed in the process data file have terminated.

Your program should print out a line of the following form, each time a process starts running:

`time 20, 15 running, numprocesses=3, numholes=2, memusage=77%`

where '`time`' refers to the time when the event happens, '`15`' refers to the *process-id* of the process that is now running, '`numprocesses`' refers to the number of processes currently in memory and '`numholes`' refers to the number of holes currently in memory. '`memusage`' is a (rounded up) integer referring to the percentage of memory currently occupied by processes.

Once the simulation ends, it should print a line of the following form:

`time 2000, simulation finished`.

Note: see the LMS for sample input process data files and the corresponding output files.

To clarify what we mean by "starts running" you should consider the following points:

- when a process moves from a given queue (based on the scheduling algorithm), it gains control of the CPU and "starts running"
- if a process does not terminate and its quantum expires, the process gives up control of the CPU and is moved back into the appropriate queue
- if/when it gets control of the CPU again, the process is considered to have "started running" again

Running your simulation program

Your program must be called `simulation`.

You should use a `Makefile` to create the executable `simulation`.

The name of the process data file should be specified at run time using a '`-f`' filename option. The scheduling algorithm to be used should be specified using the '`-a`' algorithm name option, where algorithm name is one of `fcfs, multi` corresponding to the first come first served and multi-level feedback queue respectively. The size of main memory should be specified using the '`-m`' memsize option, where memsize is an integer (in MB).

A sample command line:

```
./simulation -f in.txt -a fcfs -m 200 > out.txt
```

Part B – 3 marks

In this part of the project, you are required to investigate how the scheduling and memory management algorithms work in further detail. You should design and run small simulation experiments to illustrate the effects of model parameters. Your report, named `report.txt`, will be ≤ 500 words in length using plain text format. Ideally, your report will include a clear description of your simulation experiments; table(s) of results and a discussion of your findings.

Note: when we test your `simulation` program, will examine whether your first fit memory management implementation functions as specified for each of the queueing models for specific process data file and parameters. We strongly suggest that you develop your own test cases to be used as part of your simulation experiments. For example, you might consider investigating the effects of different quantum values in each of the queues for different process data files (consisting of varying number of processes with particular characteristics – run/burst time, memory requirements).

You must add the file `report.txt` to your SVN repository.

Submission details

Please include your *name* and *login_id* in a comment at the top of each file.

Our plan is to directly harvest your submissions on the due date from your SVN repository.

`https://svn.eng.unimelb.edu.au/comp30023-S1-16/username/project1`

NOTE: It is your responsibility to commit all necessary files to the `project1` directory of your SVN repository – this includes a Makefile, header files and source files. Do not add/commit object files or executables. Include a README file if there is any additional information that is relevant about your submission. Your should also add your `report.txt` from Part B to your SVN repository.

If you do not use your SVN repository for the project you will NOT have a submission and may be awarded zero marks. It should be possible to "checkout" your SVN repository, then type `make` to produce the executable `simulation`.

Late submissions will incur a deduction of 2 mark per day (or part thereof).

If you submit late, you MUST email the lecturer, Michael Kirley <`mkirley@unimelb.edu.au`>. Failure to do will result in our request to sysadmin for a copy of your repository to be denied.

Extension policy: If you believe you have a valid reason to require an extension you must contact the lecturer, Michael Kirley <`mkirley@unimelb.edu.au`> at the earliest opportunity, which in most instances should be well before the submission deadline.

Requests for extensions are not automatic and are considered on a case by case basis. You will be required to supply supporting evidence such as a medical certificate.

Plagiarism policy: You are reminded that all submitted project work in this subject is to be your own individual work. Automated similarity checking software will be used to compare submissions. It is University policy that cheating by students in any form is not permitted, and that work submitted for assessment purposes must be the independent work of the student concerned.

Using SVN is an important step in the verification of authorship.

Assessment

This project is worth 15% of your final mark for the subject. Your submission will be tested and marked with the following criteria:

Code:

- 10 marks – correct results generated for tests consisting of running your simulation program using varying command line arguments and process data files. (3 tests for the `fsfs` model and 7 tests for the `multi` model).

  Note: a `diff` command will be used to check program output against the expected output.

- 2 marks– for the coding style, design and appropriate documentation (clarity of code is important).

Report:

- 2 marks – simulation experiments listed, appropriate analysis of results included.
- 1 mark – conclusions valid, coherent arguments presented.

Questions about the project specification should be directed to the LMS discussion forum.