CSCE 5640: Operating System Design

Project 2, Fall 2024

There will be two projects for which you will be expected to work in small groups. This document contains the list of projects for Project 2. Your team needs to choose ***one*** from the list. The maximum group size is four (4) but can be less than four. **You should implement it in C/C++/Java and testable in CSE machines.**

First, please submit your project proposal (see Section: Project Proposal Format) which should include the selected project, team members, and tasks divisions by the due date (see Canvas). Proposal submission by one of the team members is sufficient. Then, for the project 2 submission, submit project report document (see Section: Project Report Format), and all the source code files, and a readme or any other file(s) that help to compile and test your project on canvas by the due date (see Canvas).

Projects:

1. **Contiguous Memory Allocation [1]**
    a. In Section 9.2, we learned different algorithms for contiguous memory allocation. This project will involve managing a contiguous region of memory of size ***MAX*** where addresses may range from **0 ... *MAX*–1**.
    b. Your program must respond to four different requests:
        i. Request for a contiguous block of memory
        ii. Release of a contiguous block of memory
        iii. Compact unused holes of memory into one single block
        iv. Report the regions of free and allocated memory
    c. Your program should take following inputs:
        i. Initial amount of memory.
        ii. One of the following commands:
            - **RQ** (request),
            - **RL** (release),
            - **C** (compact),
            - **STAT** (status report)
    d. You can include these inputs in the text file. The first line should be the initial amount of memory and the remaining lines are the commands supported by the program.
    e. Sample test file content:
        i. **1048576**
        ii. **RQ P0 40000 W**
        iii. **RL P0**
        iv. **C**
        v. **STAT**
    f. The above configuration initializes the program with 1 MB (1,048,576 bytes) of memory.
    g. Followed by the request for 40,000 bytes for process **P0**. The first parameter to the **RQ** command is the new process that requires the memory, followed by the amount of memory being requested, and finally the strategy. (In this situation, "**W**" refers to worst fit.)
    h. Then, releasing the memory that has been allocated to process **P0**.

i. After that, the command for compaction, **C**. This command will compact unused holes of memory into one region.

j. Finally, the **STAT** command for reporting the status of memory. It should output all the allocated memory ranges for the processes and unused memory ranges (external fragmentations):
Addresses [0:315000] Process P1
Addresses [315001: 512500] Process P3
Addresses [512501:625575] Unused
Addresses [625575:725100] Process P6
Addresses [725001] . . .

k. **Allocating Memory.** Your program will allocate memory using one of the three approaches highlighted in Section 9.2.2, depending on the flag that is passed to the RQ command. The flags are:
   i. **F**—first fit
   ii. **B**—best fit
   iii. **W**—worst fit

l. This will require that your program keeps track of the different holes representing available memory. When a request for memory arrives, it will allocate the memory from one of the available holes based on the allocation strategy. If there is insufficient memory to allocate to a request, it will output an error message and reject the request.

m. Your program will also need to keep track of which region of memory has been allocated to which process. This is necessary to support the **STAT**. The first parameter to the RQ command is the new process that requires the memory, followed by the amount of memory being requested, and finally the strategy. (In this situation, "**W**" refers to worst fit.) command and is also needed when memory is released via the RL command, as the process releasing memory is passed to this command. If a partition being released is adjacent to an existing hole, be sure to combine the two holes into a single hole.

n. **Compaction:** If the user enters the C command, your program will compact the set of holes into one larger hole. For example, if you have four separate holes of size 550 KB, 375 KB, 1,900 KB, and 4,500 KB, your program will combine these four holes into one large hole of size 7,325 KB. There are several strategies for implementing compaction, one of which is suggested in Section 9.2.3. Be sure to update the beginning address of any processes that have been affected by compaction.

o. Tests cases:
   i. Test the program with different system memories such as 1MB, 2MB, 4MB, 8MB, 16MB, etc.
   ii. For each system memory, test program with different sequences of different memory requests + allocation memory (**F, B,** and **W**), releasing processes, compaction.

p. Expected outcomes:
   i. Compute the external fragmentations for each allocation of memory types (**F, B,** and **W**).
   ii. Report the results in Tables or Graphs.
   iii. Analysis of the results.

q. See Chapter 9 (Programming Projects: P-48) for more.

2. **Page Replacement Algorithms:**
   a. Implement the following page replacement algorithms:
      i. FIFO Page Replacement
      ii. Optimal Page Replacement
      iii. Least Recently Used (LRU) Page Replacement. To implement LRU, you may use of the following techniques:
         1. Additional-Reference-Bits Algorithm
         2. Second Chance Page Replacement
         3. Enhanced Second Chance Page Replacement
      iv. Least Frequently Used (LFU) Page Replacement
      v. Most Frequently Used (MFU) Page Replacement
   b. Create test cases and compare the page replacement algorithms to find the best performing algorithms based on metrics such as page-fault rate or total number of page faults.
      i. To create test cases, you need to have some kind of string of memory references (called ***reference string***). Either one of the following:
         1. You can create text files that include list of address sequences such as "0100, 0432, 0101, 0612, 0102, 0103, 0104, 0101, 0611, 0102, 0103, 0104, 0101, 0610, 0102, 0103, 0104, 0101, 0609, 0102, 0105", or
         2. Create test cases files that just include the page numbers such as "1, 4, 1, 6, 1, 6, 1, 6, 1, 6, 1".
      ii. We can generate reference strings artificially (by using a random-number generator, for example), or we can trace a given system and record the address of each memory reference.
      iii. Whether you use sequence of addresses or page numbers, you need to create multiple test cases and take the average for each test category.
         1. For example, you create 5 test cases that include reference string assuming 10 frames, 5 test cases that include reference string assuming 20 frames and so on.
         2. You can create different test categories with different number of frames. See "Figure 10.11 Graph of page faults versus number of frames" for more.
   c. Expected outcomes:
      i. Tables or graphs showing number of page faults vs number of frames.
      ii. Take the average and standard deviation of each test categories and either plot it graph (like Figure 10.11) or list the data in table.
      iii. Analysis of the results.
   d. See section 10.4 (Page Replacement) for more.

Useful resources:

1. https://pages.cs.wisc.edu/~dusseau/Classes/CS537-F07/projects.html
   a. The Unix Shell
   b. Multi-threaded Web Server
   c. A "Better" Malloc

    d. A “Slower” File System

2. https://www.cs.princeton.edu/courses/archive/fall16/cos318/projects.html
    a. Bootloader
    b. Non-preemptive kernel
    c. Preemptive scheduler
    d. Inter-process communication and process management
    e. Virtual memory

Other resources:

http://www.cs.columbia.edu/~nieh/teaching/e6118_s00/projects/index.shtml

http://www.cs.columbia.edu/~nieh/teaching/e6118_s00/projects/miniproject.shtml

https://ocw.mit.edu/courses/6-828-operating-system-engineering-fall-2012/pages/projects/

https://cseweb.ucsd.edu/classes/sp00/cse221/projects.html#topics

https://www.cs.colostate.edu/~cs551/Project/ListOfTopics.html

https://www.freebsd.org/

https://www.minix3.org/

https://www.raspberrypi.com/

https://www.raspberrypi.com/software/

http://www.kernel.org

References:

1. Operating System Concepts, Tenth Edition. Avi Silberschatz, Peter Baer Galvin, Greg Gagne, John Wiley & Sons, Inc., ISBN 978-1-118-06333-0

Proposal Format:

In the project proposal, the following sections should be presented:

1. Overview and objective(s) of the project.
2. Team size and team members.
3. Project plan
    a. Task divisions for the team members.
    b. Due date for subtasks.

4. Experimental environment
    a. Programming language for implementation.
    b. Operating system to test the project.
    c. Test cases.

Project Report Format:

The project must be accompanied by a detailed project report describing the problem, the implementation, experiments, and results as well as their interpretation. The final report should contain at least the following sections:

1. Title
    a. Title of your project
    b. Member name(s)
2. Introduction
    a. Overview of the project
    b. What problem is being solved?
    c. Why is it important?
3. Background
    a. What does one need to know to understand the problem?
4. Implementation
    a. What are the solutions to your problem?
    b. How do you implement?
        i. Programming language for implementation.
        ii. Operating system to test the project.
    c. Test cases.
5. Experimental Results
    a. Any results or output in graphs or tables or figures that shows results of improvements/implementations.
    b. Interpretation of the results.
6. Conclusion
    a. What have been accomplished and what is still left to be done?
7. References