Programming Assignment 3: Memory Management
Due: April 21st 2015

Note: Start working on the assignment as soon as possible; do not put this off to the end!

ASSIGNMENT GUIDELINES

The goal of this lab is to write a simple memory management simulator based on the topics covered in class. You will write a memory manager that supports both segmentation and paged memory allocation.
For simplicity, assume that processes do not grow or shrink, no compaction is performed by the memory manager, and the paging scheme assumes that all of processs pages are resident in the main memory.

What should you submit?
Submit your assignment as a `.tgz` file on `aurora`. The submission instruction is `submit proj2 yourProj.tgz`. You can name 'yourProj.tgz' a name of your choice. The zipped file should contain:

(a) All your code files, and any other files that might be needed for executing your code.
(b) README.txt
(c) A PDF file (report) containing answers to the questions from Tasks $1-4$.

GETTING STARTED

Note that this assignment does not require you to use any special (e.g., synchronization) features of Java. Use the template 'memTemplate' on Moodle as a starting point.

I. TASK 1: SEGMENTATION

Write a segmentation based allocator which allocates three segments for each process: text, stack, and heap. The memory region within each segment must be contiguous, but the three segments do not need to be placed contiguously. Instead, you should use either a best fit, first fit, or worst fit memory allocation policy to find a free

region for each segment. The policy choice is yours, but you must explain why you picked that policy in your report. When a segment is allocated within a hole, if the remaining space is less than 16 bytes then the segment should be allocated the full hole. This will cause some internal fragmentation but prevents the memory allocator from having to track very small holes. (20)

Consider the following issues while designing your memory manager

A **Efficiency of your search algorithms** First-fit, worst-fit, and best-fit all require you to search for an appropriate hole to accommodate the new process. You should pay careful attention to your data structures and search algorithms. For instance, keeping the list of holes sorted by size and using binary search to search for a hole might improve efficiency of your best-fit algorithms. We do not require you to use a specific algorithm/data structure to implement the selected policy; you have the flexibility of using any search algorithm/data structure that is efficient. We'll give you 10 points for any design that is more efficient that a brute-force linear search through an unsorted list of holes. Similarly, use an efficient search algorithm when deallocating a process from the processList. Explain all design decisions, the data structures and search algorithms used clearly in the report.

B **Free block coalescing** When a process terminates, the memory allocated to that process is returned to the list of holes. You should take care to combine (coalesce) holes that are adjacent to each other and form a larger contiguous hole. This will reduce the degree of fragmentation incurred by your memory manager.

II. Task 2: Paging

Next write a paging based memory allocator. This should split the system memory into a set of fixed size 32 byte pages, and then allocate pages to each process based on the amount of memory it needs. Paging does not require free block coalescing, but you should explain in your report your choice of algorithm and data structure for tracking free pages. (20)

III. Task 3: Fragmentation

Your code should track the level of internal fragmentation for both the memory allocators. You should also track the number of process allocations which fail due to either external fragmentation or insufficient free memory regardless of fragmentation. You should run the same input file for each of your memory allocators and compare the level of fragmentation in each. (10)

IV. Task 4: Report

Answer the following questions in your report:

4.1 In your implementation of segmentation which of the best fit, first fit, or worst fit memory allocation policy do you use to find a free memory region for each segment? Why did you pick this policy? (5)

4.2 What data structures and search algorithm do you use for searching through the list of holes (in segmentation)? You will get 5 points for implementing a brute-force linear search. If you implement anything more efficient than this, you will get full 10 points. (10)

4.3 For segmentation, what data structure do you use to track the start location of each segment? (2)

4.4 For paging, explain your choice of algorithm and data structure for tracking free pages. (5)

4.5 In paging, what data structure do you use for tracking what physical page is mapped to each virtual page? (2)

4.6 How do the levels of internal and external fragmentation compare when you run the sample input in sample.txt with each of your allocators? Why is this the case? (5 + 5)

4.7 Write an input test case where the Segmentation allocator has little internal fragmentation. Explain why this test case produces the result you see. (3)

4.8 Write another test case where the Paging allocator sees lot of internal fragmentation. Explain why this test case produces the result you see. (3)

DATA STRUCTURES

Both of your memory managers should maintain a processList that lists all currently active processes, the process Id and size of each process. For the segmentation allocator, you should also track the start location of each segment. For the paging allocator you must track what physical page is mapped to each virtual page within the process, as well as the number of bytes used in each page. You are free to use any data structures (arrays, linked list, doubly linked list, etc) to implement these lists, but you should be aware that these decisions will also affect the use of search algorithms in the segmentation allocator.

INPUT FILE

You program should take input from a file and perform actions specified in the file, while printing out the result of each action.
The format of the input file is as follows:

```
//initialize memory to this size and use this policy:
memorySize policy

// specify memory allocated to this process:
//split into segments if using Segmentation
[A, size, pid, text, data, heap]

D pid // deallocate memory for this process

P // print current state of memory

An actual file may look as follows:
8192 1
A 234 1 80 100 54
A 458 2 300 98 60
A 30 3 15 8 7
D1
P
```

```
A 890 4 200 450 240 D3
P
A 70 5 55 5 10
D2
D5
D4
P
```

V. TASK 5: DOCUMENTATION

5.1 Prepare a "README.txt" file for your submission. The file should contain the following (10):

- **Names** of all the group members
- **Instructions** for compiling and executing your program(s). Include an example command line for the same.
- **Summary of your design choices**. Keep it short and to the point.
- If your implementation does not work, you should also **document the problems** in the "README.txt" preferably with your explanation of why it does not work and how you would solve it if you had more time.
- If you did not implement certain features, you should list them as well.

5.2 You should also comment your code well. The best way to go about it is to write comments while coding. (5)