COMSC 322
Operating Systems
Programming Assignment 3: Memory Management
Due: November 22nd, 9:00 PM

Note: Please do not put this off to the end!

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 process's pages are resident in the main memory***.

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

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)

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)

Answer the following questions in your report:

A. 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)
B. 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)
C. For segmentation, what data structure do you use to track the start location of each segment? (2)
D. For paging, explain your choice of algorithm and data structure for tracking free pages. (5)
E. In paging, what data structure do you use for tracking what physical page is mapped to each virtual page? (2)
F. 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)
G. Write an input test case where the Segmentation allocator has little internal fragmentation. Explain why this test case produces the result you see. (3)
H. 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:

memorySize policy //initialize memory to this size and use this policy
A size pid text data heap // allocate so much memory to this process (split into segments if using Segmentation)
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
D 1
P
A 890 4 200 450 240
D 3
P
A 70 5 55 5 10
D 2
D 5
D 4
P

4. Prepare a "README.txt" file for your submission. The file should contain the following: (5)

a) Names of all the group members.
b) Instructions for compiling and executing your program(s). Include an example command line.
c) If your implementation does not work, you should also document the problems in the README file, preferably with your explanation of why it does not work and how you would solve it if you had more time.
d) If you did not implement certain features, you should list them as well.

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

Getting Started

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

```
class MemoryManager
{

public MemoryManager(int bytes, int policy)
{  // intialize memory with these many bytes.
   // Use segmentation if policy==0, paging if policy==1

}

public int allocate(int bytes, int pid, int text_size, int data_size, int heap_size)
{ // allocate this many bytes to the process with this id
  //   assume that each pid is unique to a process
  // if using the Segmentation allocator: text_size, data_size, and heap_size
  //   are the size of each segment. Verify that:
  //     text_size + data_size + heap_size = bytes
  // if using the paging allocator, simply ignore the segment size variables
  // return 1 if successful
  // return -1 if unsuccessful; print an error indicating
  //   whether there wasn't sufficient memory or whether
  //   you ran into external fragmentation

}

public int deallocate(int pid)
{ //deallocate memory allocated to this process
  // return 1 if successful, -1 otherwise with an error message

}


public void printMemoryState()
{ // print out current state of memory
  // the output will depend on the memory allocator being used.

  // SEGMENTATION Example:
  // Memory size = 1024 bytes, allocated bytes = 179, free = 845
```

```
 // There are currently 10 holes and 3 active process
 // Hole list:
 // hole 1: start location = 0, size = 202
 // ...
 // Process list:
 // process  id=34, size=95 allocation=95
 //    text start=202, size=25
 //    data start=356, size=16
 //    heap start=587, size=54
 // process id=39, size=55 allocation=65
 // ...
 // Total Internal Fragmentation = 10 bytes
 // Failed allocations (No memory) = 2
 // Failed allocations (External Fragmentation) = 7
 //

 // PAGING Example:
 // Memory size = 1024 bytes, total pages = 32
 // allocated pages = 6, free pages = 26
 // There are currently 3 active process
 // Free Page list:
 //   2, 6, 7, 8, 9, 10, 11, 12...
 // Process list:
 //  Process id=34, size=95 bytes, number of pages=3
 //    Virt Page 0 -> Phys Page 0  used: 32 bytes
 //    Virt Page 1 -> Phys Page 3  used: 32 bytes
 //    Virt Page 2 -> Phys Page 4  used: 31 bytes
 //  Process id=39, size=55 bytes, number of pages=2
 //    Virt Page 0 -> Phys Page 1  used: 32 bytes
 //    Virt Page 1 -> Phys Page 13  used: 23 bytes
 //  Process id=46, size=29 bytes, number of pages=1
 //    Virt Page 0 -> Phys Page 5  used: 29 bytes
 //
 // Total Internal Fragmentation = 13 bytes
 // Failed allocations (No memory) = 2
 // Failed allocations (External Fragmentation) = 0
 //
}

}
```