CIS 345/545: Project 2
(Due: Oct 25)

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

1. Segmentation: Write a segmentation based allocator which allocates three segments for each process: text, data, 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 README. 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. You should consider the following issues while designing your memory manager:
   - 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 README file.
   - 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 README your choice of algorithm and data structure for tracking free pages.

Getting Started:

You may choose to use either C/C++/Java to finish this assignment. In particular, the Java template looks as follows. Note that this template is to demonstrate how your program should react to each input line. You don't have to use this as a skeleton even if you're writing in Java.

```
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
  // ...
  // Failed allocations (No memory) = 2
  //

  // 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
   //
   // Failed allocations (No memory) = 2
   //
}

}
```

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 explain why you pick the data structure you choose. This decision 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:

```
### input_frag.txt for segmentation
1024 0
A 256 1 80 122 54
A 32  2 15 10 7
A 512 3 300 98 114
A 256 4 200 50 6
D 1
A 257 5 200 40 17
D 2
A 257 6 200 40 17
P

### input_page.txt for paging
1024 1
A 259 1 65 97 97
A 64  2 16 16 32
```

```
A 512 3 300 98 114
A 32 4 16 16 0
D 2
A 257 5 200 40 17
D 3
A 257 6 200 40 17
P
```

Turning it in

Each group (at most two students) has to submit your program electronically by using the following command on grail:

turnin -c cis345y -p proj2 USERID

Please create a new directory called USERID and copy your source files and a README file (must be in txt format) to this directory. Note: No need to hand in a hard-copy document but your README file should include the instructions that explain how to compile and run your program as well as the description of your code, experiences in debugging and testing, etc. Again, start on time and good luck. If you have any questions, send e-mail to c.yu91@csuohio.edu or the course TA, Jingyuan Liang, at j.liang18@vikes.csuohio.edu.