OS Lab 3 Report

Saturday, November 22, 2014
7:56 PM

Group Members:
Mina Khan
Jingjing Rong

1, Compare last print output:

Input file used : sample.txt

Segmentation:

Internal fragmentation = 0 byte
External fragmentation = 0

Paging:

Total Internal Fragmentation = 71 bytes
Failed allocations (No memory) = 3
Failed allocations (External Fragmentation) = 0
External Fragmentation = 0 //fixed size paging has no external fragmentation

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)

We chose best fit policy, which uses the smallest block that can satisfy request. Best fit maximizes available space compared to first fit, particularly when it comes to conserving space available for large allocations. This leads to lesser external fragmentation. Best policy generally means that we have to look through the entire array of memory holes to find the best match, but we optimized this process by keeping a sorted list of memory holes by hole sizes and then searching using binary search.

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)

We use ArrayList to store the memory holes and keep a sorted list using Collections.sort. A Hole is a Comparable object is compared according to its size and thus we sort holes according to their sizes and then do binary search to find the best fit hole. We use ArrayList to allow for quick random access using index in array.

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

We have a class called SegmentedProcess that keeps track of all the segments, their sizes and starting locations. We create a SegmentedProcess object corresponding to each new process and save the list of SegmentedProcess in a HasMap that maps from the pid to SegmentedProcess that has that pid.

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

For allocation, we first figure out how many pages needed, then check if there are that many free pages remaining. If so then for each page in need, we find the first page that is available and assigns to the process, then look for another page that is available. (a bit optimization done: for a process, if the first page is allocated, the second time it will look for unused page from the first page's next page, ie will not search what is alread searched and unavailable.)

For deletion, we first check if the process is in the active list, if not return with message. Then look up the process from the map, and free up the pages by setting related info to default. And removes the pages from the pageInUse list. And removes process from activeProcessesMap.

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

Physical memory simulation: array of Page objects, this is because we want to fix the physical address when it's allocated ie it does not shrink or grow or move places
Page Class: it tracks its physical address and this is permanent, when this page is assigned to other things or freed, physical location value stays. It also remembers which process is currently using it, how much of the 32 is used, and which virtual page of that process is this page in
Process Class: it tracks the pages instances assigned to it, keep record of process ID, size.
HashMap: It maps active process ID to the Process instance, for quick look up when trying to remove a certain ID. And for track which processes are active.

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)

For Segmentation, we some internal for process 4,but we do not have external fragmentation for sample.txt. External fragmentation is likely for segmentation, but since we are using best fit, we are minimizing the possibility of external fragmentation. Internal fragmentation does not usually happen for segmentation, but since we are using this policy “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”, segmentation happens for heap segment of process 4 because the best fit hole has size 84, which is only 14 larger than the heap size, i.e. 70.
Paging has no external fragmentation because pages are fixed size and can be allocated non-contiguously. Paging has a lot of internal fragmentation because of the fixed page sizes.

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)

```
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
```

This file has internal fragmentation=0 because process segments are never allotted in holes that are just 16 bytes or less bigger than the memory required by the process. Therefore, there is no internal fragmentation.

```
8192 1
A 234 1 80 100 54
A 458 2 300 98 60
A 30 3 15 8 7
D1
P
A 230 4 200 10 20
D2
D3
D4
P
```

This file has internal fragmentation=4 because heap segment for process 4 is allotted in hole of size 24 leftover when process 1 was removed, and the data and text segments of process 1 were already allocated in the hole created by deallocating process 1.

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)

When all process are of size 1, and the number of process equals to total page number. Then there are 31*totalPageNumber bytes of external fragmentation because each process take up 32 bytes even when it only uses 1 byte, thus wasting a lot of memories.