Caleb Pudvar

Andrew Thomas

Memory Management

Analysis

Our solution of this memory management situation is done using the first fit and worst fit algorithms. First fit places a process or a segment of a process in the first free space that would be able to accept it. In this way, for the segmented approach different segments of the same process are allowed to be separated between other processes. Worst fit places the next process or process segment in the largest available fragment. If none exist that will accept the selection, it will proceed to be added after the last process as long as memory permits it. In either case, the segmented portion is made to allow the separate segments of a process to separate one another. In theory, this separation should greatly reduce fragmentation between processes.

We created a number of input files that will analyze the various properties of our choices. Each file has been designed to sufficiently demonstrate that memory is being managed in the desired fashion. As follows will be an exploration of each file's strength. Since each file is designed for specific demonstration, they don't demonstrate the other modes of management in a helpful way and thus will not be included.

Contiguous – First Fit

This method is fairly simple and doesn't involve anything special. Memory coming in as a single chunk will enter the first available free space in memory that will accept it. If none exist, it will follow the last process currently in memory. File used is "con_first.txt". Issues with first fit lie in the fact that they tend to have fragmentation towards the bottom of the process list. Since it doesn't check for a better option, perfect matches can be overlooked.

• **First, memory is set up to have multiple spaces that could hold the next process** • **When the next process (P4) comes into memory, it takes up the first space it can. In this example, no fragmentation remains.**	```---------- ID: FREE Memory: 500 ID: P2 Memory: 2000 ID: FREE Memory: 1000 Total Used Free 256000 2000 254000 ----------```	```---------- ID: P4 Memory: 500 ID: P2 Memory: 2000 ID: FREE Memory: 1000 Total Used Free 256000 2500 253500 ----------```

Contiguous – Worst Fit

In this section, memory as a contiguous block will search for free space upon the start of a process. After scanning all available positions, it will enter the largest free option. Again, if none exist it will follow the last process in memory. File used is "con_worst.txt". The issue with worst fit is fairly clear – small processes can decrease the size of a large free area, thus not providing an option for a large process that would otherwise fit.

• **First, memory is formed to have two free spaces of 400 and 600 KB respectively.** • **P6, with a size of 250, could enter both free spaces** • **It scans to find the largest free space in memory, and properly takes up the larger available space.** • **Free space is modified to account for the entry.**	```---------- ID: FREE Memory: 400 ID: P2 Memory: 300 ID: P3 Memory: 700 ID: FREE Memory: 600 ID: P5 Memory: 1000 Total Used Free 256000 2000 254000 ----------```	```---------- ID: FREE Memory: 400 ID: P2 Memory: 300 ID: P3 Memory: 700 ID: P6 Memory: 250 ID: FREE Memory: 350 ID: P5 Memory: 1000 Total Used Free 256000 2250 253750 ----------```

Segmented – First Fit

Segmented memory is more complex, but helps to remove fragmentation by using smaller selections of free memory since processes are allowed to be broken up. In this, each segment of a process independently searches for the first available free space it will allow. Segmented first fit still has the issue of containing more fragmentation towards the bottom of a process list, because it doesn't look for better options.

• **Memory setup so that the free space is smaller than the total size of P4, but a single segment of it will still fit** • **When P4 enters memory, one segment enters the free space** • **The other segment has no other option but to follow at the end** • **Minimized fragmentation**	``` -------------------------- ID: P1 Memory: 100 ID: FREE Memory: 500 ID: P3 Memory: 50 ID: P3 Memory: 50 Total Used Free 256000 200 255800 -------------------------- ```	``` -------------------------- ID: P1 Memory: 100 ID: P4 Memory: 450 ID: FREE Memory: 50 ID: P3 Memory: 50 ID: P3 Memory: 50 ID: P4 Memory: 300 Total Used Free 256000 950 255050 -------------------------- ```

Segmented – Worst Fit

This method is arguably pretty good, because the segments force the fragmentation that will inevitably exist to be minimized if enough processes with reasonably sized segments enter memory. Also, this method keeps the various fragmentation sections large as possible; helpful because it provides space for a process with large segments if ever necessary. The only

issue could arise when these large segments are filled enough just before a process with large segments arrives, forcing it to come after the last process in memory.

• **NOT SHOWN: Memory spaces of 1850 and 2000 forced respectively.** • **First, P6 enters memory and the first segment enters the 2000KB space, making the free 1750KB, making it the smaller option** • **The second segment of P6 now enters the first free space which has become the larger option.** • **In the next column, P7 enters with the same situation** • **In this manner, all free spaces would fill at roughly the same rate over time** • **This would gradually reduce fragmentation evenly**	---------- ID: P1 Memory: 250 ID: P1 Memory: 100 ID: P1 Memory: 100 ID: P1 Memory: 50 ID: P6 Memory: 150 ID: FREE Memory: 1700 ID: P3 Memory: 100 ID: P3 Memory: 200 ID: P3 Memory: 300 ID: P3 Memory: 400 ID: P6 Memory: 250 ID: FREE Memory: 1750 ID: P5 Memory: 100 ID: P5 Memory: 100 Total Used Free 256000 2100 253900 ----------	---------- ID: P1 Memory: 250 ID: P1 Memory: 100 ID: P1 Memory: 100 ID: P1 Memory: 50 ID: P6 Memory: 150 ID: P7 Memory: 250 ID: FREE Memory: 1450 ID: P3 Memory: 100 ID: P3 Memory: 200 ID: P3 Memory: 300 ID: P3 Memory: 400 ID: P6 Memory: 250 ID: P7 Memory: 250 ID: FREE Memory: 1500 ID: P5 Memory: 100 ID: P5 Memory: 100 Total Used Free 256000 2600 253400 ----------