Shannon Toft

May 6, 2019

Operating Systems Sp’19

**Assignment 6 – Main Memory**

Main memory can be organized by different schemes, such as contiguous-memory allocation, pure segmentation, and pure paging. Each has their own techniques in storing memory and sharing throughout each process. They also have their own approaches when dealing with external and internal fragmentation.

For our first memory management scheme, memory is separated into two partitions that are set for the user processes and the operating system. Each partition is a block of memory that is a fixed size. In each partition, when free, lays a single process when selected from an input queue. It is placed into the free space also known as a hole. Each process has to have a record of memory needed to store, so that it can be adequate for the partition space. Once the process has completed its procedure, it will terminate, free the partition, so it may allow for the next process from the input queue to load. This method will continue until there is not enough adequate space to fill the holes of the memory block. When such a case happens, the operating system will skip through the next processes that can fulfill the leftover space, causing the other processes to wait.

In order to accommodate the other processes and to lessen wait times, there are approaches that can be made to confirm majority of the holes are being occupied. These strategies are known as first fit, best fit, and worst fit. The first fit, says it in the name, and occupies the first available hole that fits the memory needed. The second fit must search through the whole block to find the smallest available space that meets the requirements of the memory needed. And lastly, the worst fit also must search the whole block, placing the process into the largest available space, which is usually at the end of the block. Although the second fit and worst fit both run through the whole block, the worst fit does not hold the size of each hole available.

Between the first and best fit, free spaced holes are created after loading and removing memory causing external fragmentation within the memory space. The blocks will eventually contain broken scattered areas throughout. Ultimately, there will be enough space to satisfy a process, but because of the scattered areas, the process will not be able to load. This can be resolved through compaction, so that these scattered areas may be shuffled around to connect more free space together and allow a process to fit and be loaded. Internal fragmentation is not necessarily occurring in contiguous memory, unless if allocated to the same size pages. And since every process has its own page, sharing is not being practiced as well.

During pure segmentation, chunks of memory are now divided into variable-sized segments with ordering not a priority. Each segment records its name and size, so that the programmer may refer to each segment by name or offset. A segment table is created to link the user addresses to physical addresses. Segments are swapped back and forth between the main memory and the secondary storage which can create external fragmentation, similar to contiguous memory. Although similar to contiguous memory, pure segmentation is put in order to reduce the external fragmentation. Since the processes are different sizes, pure segmentation also does not have internal fragmentation. However, because the processes are dynamically linked, sharing throughout each process is achieved.

Lastly, pure paging is non-contiguous and works with pages and frames instead of segments. These pages and frames are broken into fixed sized blocks all of the same size. Processes are taken from a backing store when executed and loaded into an available memory space. A page table is created to hold the base address of each page in physical memory. In addition, a frame table is also created holding details of the physical memory. Since the processes can be relocated to any frame, external fragmentation is eliminated. But internal fragmentation is seen when processes exceed page boundaries. Overhead increases due to the escalation of context switching. However, sharing code is possible allowing for two or more processes to run at the same time.