Module 4: Critical Thinking

Comparing Best-Fit to First-Fit Placement Algorithms

Nolan Byrnes

CSC507 – Foundations of Operating Systems

Colorado State University – Global Campus

Professor Joseph Issa

December 10th, 2022

Module 4: Critical Thinking

GeeksforGeeks explains that with dynamic partitioning, main memory is initially empty and “partitions are made during the run-time according to process’s need instead of partitioning during system configure” (GeeksforGeeks, 2020, para. 5). When a new process is introduced, it finds a partition that is available that will accommodate the space that it requires and assigns it to the new process. A situation can arise where there is enough space in main memory to load the next process, but with it being too fragmented, to where the available memory is not together in one block. To resolve this issue, compaction can be used which is defined as “a technique to collect all the free memory present in form of fragments into one large chunk of free memory, which can be used to run other processes” but is also time consuming (GeeksforGeeks, 2021, para. 1). To avoid memory compaction, memory allocation algorithms such as best-fit and first-fit can be used to reduce the frequency of compaction from occurring. This paper discusses and compares the best-fit and first-fit memory placement algorithms, and why the first-fit algorithm can perform better than best-fit.

The first-fit allocation algorithm “works by allocating the first free partition or hole large enough which can accommodate the process. It finishes after finding the first suitable free partition” (Tutorialspoint, n.d., para. 2). Tutorialspoint claims that it is the “Fastest algorithm because it searches as little as possible” (Tutorialspoint, n.d., para. 3). The best-fit allocation algorithm works by “allocating the smallest free partition which meets the requirement of the requesting process” (Tutorialspoint, n.d., para. 5). This algorithm takes longer to complete, as it must check and compare each of the free memory blocks each time that it is allocating memory space for a process. This ensures that after each memory allocation, the leftover fragment size is the smallest possible. Stallings explains that “although each memory request always wastes the smallest amount of memory, the result is that main memory is quickly littered by blocks too small to satisfy memory allocation requests. Thus, memory compaction must be done more frequently.” (Stallings, 2019, pg. 322).

I created a scenario where five initial processes were loaded, three completing, and then six more processes being loaded in. I modeled the memory allocations of the six extra processes being loaded in for both the first-fit and best-fit algorithms. As I was modeling the steps for the best-fit algorithm, I noticed that the execution time would take longer to allocate the memory, as it had to compare each of the empty blocks, whereas with the first-fit, it only had to identify the first available block that contained enough space. The best-fit algorithm had to perform a memory compaction when loading in Process 11 due to the 11th process requiring 100 MB, and the only spots available contained 50 and 91 MB. Because the best fit algorithm was choosing to have the smaller sized blocks, when a process with a larger memory requirement came in, although it technically had enough space, it was fragmented, so a memory compaction had to occur. The first fit algorithm always chose the first memory spot that had enough space, so when Process 7 that required 100 MB, the first fit chose a partition with 150 MB, resulting with more of the free blocks to contain more available space. This helped the first fit algorithm proceed when process 11 came through with a 100 MB request, whereas the best-fit algorithm had its main memory too fragmented to continue.

A situation where the best-fit allocation algorithm may perform better than first fit is when by using the best fit algorithm, each allocation would leave no fragmentation. Kellett explains that “fragmentation is more likely to be a problem when your application makes a series of allocations and deallocations such that each time an allocation is made it cannot re-use space that was left by a previous deallocation of a similar (or larger) size block” (Kellett, 2021, para 38). If each process allocation is leaving zero fragmentation after each allocation, and the processes are finishing at a very fast rate, theoretically, the processes could be repeated forever without having to perform a compaction. This would require a low degree of variability in terms of process size, and for each process completing at the same rate of speed, at a very fast rate. A likely scenario would be the same few processes being loaded into memory multiple times in a loop. I feel that this scenario described is very controlled, and an unlikely situation to occur because as soon as a process comes in that leaves any fragmentation, the best fit algorithm will soon leave main memory too fragmented to continue unless they perform a compaction. Due to the unpredictability and variability of the size of processes that are coming in and completion time of each process in a typical situation, I would choose the first-fit algorithm over the best fit algorithm every time.

**REFERENCES**

GeeksforGeeks. (2020, January 16). Variable (or dynamic) partitioning in operating system. <https://www.geeksforgeeks.org/variable-or-dynamic-partitioning-in-operating-system/>

GeeksforGeeks. (2021, December 17). Compaction in operating system. <https://www.geeksforgeeks.org/compaction-in-operating-system/>

Kellett, S. (2022, September 7). Memory fragmentation, your worst nightmare. Software Verify. <https://www.softwareverify.com/blog/memory-fragmentation-your-worst-nightmare/>

Stallings, W. (2018). Operating systems: Internals and design principles. Pearson.

Tutorialspoint. (n.d.). OS memory allocation Q & A #2. Online Tutorials Library. <https://www.tutorialspoint.com/operating_system/os_memory_allocation_qa2.htm>