Program 2: Memory Simulator

CSC 458

Josiah Huntington

My Memory Management simulator two structs for its primary functions. One struct is designed to contain the process's details (ID, arrival time, lifetime, etc), and the other contains the basic blocks for memory (start/end address, and content ID).

My program begins by prompting for and storing the user input, then parsing the provided file to load processes. It then sets up memory according to the given policy, designing either a page table for PAG or one solid block of memory for VSP and SEG. It then runs a loop until the system times out.

My program runs off of three main sections. It begins the loop by checking all the processes' arrival times against the system clock and adds them to the input queue if they are equal. Then it checks all the processes' finish times against the system clock and removes them from memory and cleans up if they match. In the final primary section it will check the memory management policy and attempt to store processes from the queue into memory utilizing the provided policy and parameters.

The simulator handles paging (PAG) by separating the memory into a paging table using the provided memory size and page size. When a process is queued up for placing into memory, a check is made to see how much total memory is available and as long as there is enough the process will be stored in as many pages as needed. Once a process is finished, the ID's in all of the processes pages will be updated to 0 to show their availability.

Variable-size partitioning (VSP) is handled by looking a vector of memory blocks containing the entirety of the system memory. Utilizing whichever fit algorithm was provided by the user the memory will be searched to find a block of continuous free space that the process can be placed into. If there is excess space, a new memory block will be emplaced into the vector and the free space's value range will be updated. If it is a perfect match for space, the ID in that memory block will simply be changed to that of the process. To clean up after VSP, the ID of the memory block a process was using will be reset to 0. Additionally, if it was next to additional free space, one of the memory blocks will update its range to cover both, and the now useless one will be erased.

Segmentation (SEG) is dealt with very similarly to VSP. It uses the same concepts, but rather than being constrained to one continuous block, the processes are able to split up their segments as needed. My simulator tries to place each segment into memory using the given algorithm, but if one is unable to fit it will clean up any already placed and then continue to wait in the queue.

The algorithms for VSP and SEG are fairly straightforward. First-fit simply looks through the vector of memory blocks to find the first available and large enough block. The best-fit algorithm compares the memory blocks against the needed space and places the process/segment into the closest size match, placing and exiting immediately if it finds a perfect match. Finally, the worst-fit algorithm checks the available space and places the process/segment into the largest open space it can find.

The final aspect of my simulator is the system clock. The system clock looks at the processes to find the next closest event (arrival or completion) and then sets itself to that event. Each process has a next_event value that is set to its arrival time at creation, and then updated to its end time once it is stored in memory.