Algorithm Overview


algorithm begins by creating the processes to be executed, storing them within an array to facilitate easier management. After the processes have been created, the algorithm then creates the segment table that is used to manage the memory cells and processes. Finally, the timer begins (using timerVal to determine how many system cycles to complete), and the processes begin to be brought into memory and executed.

Within the timer, the program first displays the allocation status of all of the segments, and then determines whether a new process has arrived that needs to be allocated to a memory segment. Once allocation has completed and the updated segment table is displayed, the program begins to work on the processes within memory. A work buffer is created, which allows the program to keep track of which processes it has worked on thus far in a given system cycle (this prevents a process from being worked on multiple times if it occupies many memory cells). Finally, if a process has completed its execution, it is removed from main memory, and the segment table is printed once again.

Notes & Assumptions

memoryProject Class

The memory segments operate statically and are allocated with a first-fit style algorithm.

The allocateMemoryAndPrint() method is a helper method which is used to to allocate memory, print the status, and print the segment table.

segmentTable Class

The allocateMemory() method in the segmentTable class allocates memory for a given process. The method iterates through each memory segment in the array to find a suitable segment for the process. If it finds a suitable segment, it creates an allocated segment, updates the original segment's size and address, and returns the allocated segment. If the segment has insufficient space, it creates a new segment with available space. It then returns the allocated segment. If it does not find a suitable segment, it returns null.

The offset value is determined by taking the distance, measured in “bytes”, from the start of the current memory segment to the start of the next memory segment. Thus, if a process occupies segments A (at position 00) and B (at position 10), the offset will simply be 10 “bytes”. If a process occupies segments B (at position 10) and E (at position 40), however, then the offset will be 30 (as 40-10=30).

The printSegmentTable() method prints out the segment table. It does so by iterating over the memoryArray and printing out the details of each memory segment. It prints out the physical start address, remaining size, allocation status, process ID, and offset.

The checkEmptySegments() method prints out whether a given memory segment is allocated or not.

The nextEmpty() method in the segmentTable class determines the next empty cell available; this is used to help calculate the offset value.

The emptyMemory() method in the segmentTable class calls the memorySegment class's unassignProcess() method.

The condenseMemory() method in the segmentTable class checks to make sure all the segments are at the beginning of the array.

memorySegment Class

The size variable is often used to determine the remaining size of a given memory segment. The offset variable begins at -1, which signifies that there is no offset value.

The getProcessID() method in the memorySegment class ensures that, if a memory segment is found to not be holding a process, the memory segment is marked as deallocated as a form of safe redundancy. In this event, returns -1 to signify that no process is being held.

Process Class

The burstTime variable is how many system cycles must work on the process before the process is considered complete.