Lab 4

Process Scheduler

Design Document

Memory Allocation

In the tasked project, the memory was allocated using first-fit algorithm. This algorithm will scan the list and if it finds the first big enough space to keep a process, it will stop scanning. In addition, it will load that process into that memory block. The first-fit works with partitions, producing two of them in the algorithm. Over both of these partitions, one will be just an empty space, meanwhile the other will store the process. There are a bunch of algorithms that I could have used. Can search them online. There is the one used First-Fit, there is Next-fit, Best-Fit, Worst-Fit, Quick-Fit. The first fit simply is said to be the best because it takes smaller amounts of time compared to the rest, and it makes bigger holes needed to load more processes later. And it is also easiest to implement. Also, was the most used when researching how to make a host dispatcher. If I were to use another one would probably be Next-fit, as it is similar to the one, I just implemented, and it uses an assumption backed up by good probability. Which states that there will be a larger hole in the next spaces remaining.

Data Structures

The data structures implemented includes Linked List, to link the process nodes one after another. Queue to operate how the linked list will work. The project required that there be 4 queues in total. One for real-time, and 3 Priority Queues, from Priority 1 down to priority 3. Within the queues, the process needed to be linked in some way that it was easy to navigate, and the processes needed to be encapsulated in nodes, therefore using a normal array would not help us. This is why we need to make a process node, which contained that process’s information. This information is crucial for the priority of the process, and the time left in the process.

The structure used by the dispatcher for queuing is a linked list integrated within the queue itself, this is needed due to the amount of information needed for one memory block, which an array simply cannot handle, or is not an appropriate use in this situation. Memory in this dispatcher also using linked list to navigate through the memory blocks, it will look at the size and offset to determine if the memory block can be allocated, else if cannot find the met criteria, there will be allocation failure. The dispatcher itself when dispatching will unload the pending process from the input queue, check to see if it is a user job to add it to user job queue, otherwise it will add it to the real-time process queue, checks memory size and remaining memory to check if should admit the memory or no. The dispatching checks where the processes should go depending on their requirements and what the dispatcher can accept.

Functions

The Functions for this project within process management c file includes create\_pcb(), that brings back a pointer to an empty process block. areEmptyQueues(), outputs true if all dispatcher queues are empty. ReqRes\_check(), ReqRes\_alloc(), start\_process(),restart\_process(), suspend\_process, kill\_process, init\_process(),placeInQueue(). These are all functions needed to check and allocate the requested resources, start the process, restart it, suspend it, kill it, initialize it, and place it in the queues. The functions for the memory management c file includes getMemoryLeft() this retrieves the memory left in the storage provided by the dispatcher software. There is memoryCheck, AllocMemory, FreeMemory, SplitMemory, MergeMemory. All functions needed to allocate memory, check on it, free it, split the memory, and merge the memory, all needed for the first-fit algorithm. Which brings us too the queue c file. This file includes functions like init\_queue, dequeue, isEmptyQueue, enqueue. All functions related to initializing the queue, popping an item out of a queue, pushing an item into the queue, and seeing if the queue is empty or else there is no point in using the init\_queue. Last file is util.c, which is just a file used to read the input files, extract the information and feed it into the 4 queue host dispatcher. This file includes functions such as arraySize(), readInfo(), read\_file(). The first function returns the size, followed by the second function which reads the input file and produces an array the size of 8 with the needed int values. And last is the read\_file just reads the content of a file, putting them in a string.

Overall Structure

The dispatcher host executes main, main read process inputs, passes it to the process management file, with the function calls Init\_Dispatcher and Start\_Dispatcher. Init dispatcher will initialize a dispatcher that will run and do the said task for a dispatcher. The initialization sets up the queue lists for Inputs, Jobs, real-time, level 1-3 priority queues. In initialization there will also be extraction from the read input file and placed within the Input Queue, that will apply the FIFO rule when the dispatcher starts. When dispatcher starts it will constantly check if the process is a user job or not, place it either real-time queue, or put into the user job queue, and from the user job queue to the priority queues. The important thing here is that the process is being checked constantly throughout the lifetime of the dispatcher, it will ask if there is a process available then check to identify for real-time or user job, and if there is room in the queues for the said process. The process is then placed within a managed memory block of the queue list, that is done using the check and allocate memory functions, using partitioning with the split and merge memory functions. Overall utilizing the First-fit algorithm to successfully allocate the process in the queues, until the processes have finished with checking the time the processes have left.