**Logo, company name

Description automatically generated**

 

Operating Systems Concepts

Assignment 1 – Design Programming Project

CSIS 3810

Ernesto Louis

10/24/2022

**SharkOS**

**Overview**

This shark machine attempts and simulates that of an OS. It establishes single-tasking as well as multitasking. A process control block (PCB) or object holds information about each process put into the Request Queue. The machine also incorporates a round Robin algorithm (CPU scheduling) which gives the feel of multi-tasking and handles any interrupts that occur. Doing this, the files/programs’ instructions are read as they are populated in the Request Queue and, based on the CPU time allocated, are executed, or handled properly if they cannot finish the current task. The information state is then saved so when it gets executed again in the Request Queue it can finish executing and continue where it last left off.

**Tools / Resources**

To ensure full knowledge of the concepts of single-tasking and multi-tasking, various tools. Resources were used to ensure full understanding. The resources used were Operating System Concepts 10th edition by Silberschatz *et al*. The project is written in Java to further extended the project, a graphical interface would have been implemented to dynamically and visually show how each process is being loaded, waited, and executed within the OS and the request queue.

**Design**

The program contains 6 files (excluding the text files). First is the SharkMachine class; this provides an interface between the machine and the OS. It was created to simulate the user “starting” a machine and the machine boots and runs the system. This is the file to run the entire project. This can be done via an IDE or the command line at the current directory, that is, the “src” directory, by running the command, “java SharkMachine” (without quotation marks).

Importantly added is a PCB class that holds information about each process. It contains the properties id, the registers (to hold the instructions and addresses), the CPUTime (or burst time of a program), the process state (to set the status of the program), and much other important information.

To hold the PCB object, created is a RequestQueue class which serves the purpose of putting/enqueueing all the processes in the order that they arrive and dequeue a process if it is taking too long and a process needs to arrive.

To determine and keep track of the arrival times and the quantum time (QT or TQ), a Scheduler class is created to serve this purpose. Utilizing this class helps process the programs into the queue appropriately and helps in handling any interrupt(s). For example, when a process is put into the request queue, if the program will not be able to finish at the time allotted before the next program has to arrive in the queue, then the round Robin algorithm must make a decision, which is, to take the current process and save its current state, remaining time and then adds the arriving process(es) into the queue. This process essentially repeats until all programs are done. This then thus simulates multi-tasking by quickly switching between contexts and making certain decisions. To simulate the time (in seconds) a for loop is used which runs from the range 0 – max( arrival time ).

Next, a FileHandler class was created to be able to read a program and insert the specific instructions into the OS’ memory as well as write the content of the registers to an output file.

Lastly is the SharkOS class, which is where all the components or, rather, the classes (PCB, Scheduler, RequestQueue, FileHandler, as well as the memory) are instantiated to act as a component of the OS. The OS contains the Round Robin algorithm, one of the most important pieces of the project that displays the concept and understanding of multitasking. Called in the constructor is a function that iterates through the project folder and locates all of the .txt files within the “programs” folder. As each file is identified, a PCB object holding the properties (ID, REGISTERS, CPUTIME, PROCESS STATE, etc.) of the process is created and inserted into an array. Each process is then given a random arrival time that is used to map a process to its arrival time. This will be crucial later. The map of the processes to arrival times is implemented within the Scheduler class. Once the programs are identified, a function is called which utilizes the map from the Scheduler class aforementioned. Because the map contains the arrival times, a loop is run from 0-max(arrival time). If x = n where n is a valid arrival time contained in the map, it is loaded into the request queue and the job mapped to it dumps its instruction into the memory called by a specific function. While the queue is not empty, it runs all processes and evaluates the time quantum against the burst time. If the process is taking too long and another process is arriving, the current process is paused, and the state of the process and the remaining time is saved. The process is then put in the back of the request queue until it is time to run again and continue where it last left off. Also, as each instruction is read, the opcode and address are identified, and a corresponding function (ADD, SUB, LDI, LDA, etc.) is called that updates the properties (in this case the registers) in the process’ PCB. After each job is done, an output of all the registers and memory at each instruction is then output into a corresponding output file within the “outputs” folder.

**Project Structure**

All of the .java and .class files can be found within the parent folder in an inner folder named “src”. Within the same parent directory, an output folder is provided which will hold all the outputs of the registers for each iteration of an instruction of a program, as well as the current status of the memory.

**Example Output**

The very first line indicates that the Quantum Time given for all processes is 3 seconds. For simplicity and not to overfill the console, 3 programs were running (also, each process was given a fixed burst time, however in the actual project, burst times and arrival times are given randomly to truly simulate multi-tasking.). Process 1 arrives at 2 seconds with a burst time of 5 seconds; process 2 arrives at 1 second with a burst time of 5 seconds; process 3 arrives at 0 seconds with a burst time of 5 seconds.

The loop then iterates from the range of 0 – max( arrival time ), which in this case, the max arrival time is 2 seconds (process 1). When x = 0 process 3 arrives in the system queue and then the program is run, however, with a burst time of 5 seconds and process 2 arriving in 1 second, process 3 will not be able to finish executing as it will have (5-3) 2 seconds left. Thus, the state of process 1 is saved as well as the remaining time to finish and then it lets process 2 come into the system queue and moves to the back to wait its turn again.

As this loop occurs, process 2 will also have to save its current state (to “waiting”) and move to the end of the queue as it will not be able to finish and so, it will switch places with process 1. Process 1 remained 2 seconds to finish in the previous time it ran, and so, in this execution time process, 1 will be able to finish with a time of (2-3) -1 seconds. As no other process is in the queue or is arriving any time soon besides process 2, process 2 will then finish running. Lastly, process 1 will arrive at x = 2 seconds and will finish executing as it is the last process arriving.

After each process is finished, a .txt file for that specific process will be generated in the “outputs” folder which will display all the register values at each instruction of the program.