**Design Document for SharkOS**

**Overview**

SharkOS is a simple operating system simulator implemented in Java. It provides a platform for executing a set of programs in the Shark Machine Code. The system uses a basic round-robin scheduling algorithm and a time quantum set to 4 to allocate CPU time to each job and context switch when necessary. Below is an in-depth explanation of the components and functionality of SharkOS.

**Components**

1. SharkOS Class

This class serves as the entry point for the operating system. It contains the main loading and control logic and orchestrates the execution of jobs. Basically, it’s where everything happens.

Attributes:

- ACC, SDR, PSIAR, CSIAR, TMPR, SAR, IR, MIR: Registers used for instruction execution and memory operations.

- currentJob: Keeps track of the currently executing job.

- clock: An instance of the SystemClock class to manage time.

- jobQueue: A queue to hold jobs that are ready for execution.

- MEMORY: An instance of the Memory class representing the system's memory.

- TIMER\_QUANTUM: The time slice (in clock ticks) allocated to each job for round-robin scheduling. This will always be set to 4 unless not set to 4.

-yield: A Boolean marked as true only when YLD/80 is passed as an instruction, thereby invoking a context switch.

Methods:

- SharkOS(): Constructor method that initializes the system.

- INIT\_SYSTEM(): Initializes the registers and memory.

- LOAD\_SHARKOS\_PROGRAMS(): Calls the functions that determine the order of jobs and the storing of their instructions into memory

* Files are loaded based on their file name in relation to the src directory (main folder where program is stored)

- createJobOrder(): Reads the file listing the programs and their order and creates a list to determine the order of each job.

- loadInstructions(List<Job>): Takes in the sorted list of jobs and loops through the instructions in each job, storing them to memory to any space that can be allocated.

* Memory is allocated dynamically using three functions—isContiguousMemory(), allocate(), and allocateRegion()—that check if the program size (number of instructions) can be allocated within the next program size addresses and if it can, then it sets the Boolean allocated to true for those spaces in memory.

- opcodeConversion(String opcode): Maps assembly opcodes to numerical codes (from 0 to 80).

- RUN\_SHARKOS(): Implements the round-robin scheduling, context switching, and job execution logic.

* It begins with a while loop that continues execution if there are jobs in the jobQueue waiting to be processed.
* Then it dequeues the next job from the jobQueue, making it the currentJob.
* Then, the state of the currentJob is set to 1, indicating that the job is in the "running" state. It signifies that the job is currently executing.
* The, loadState is used to load the saved state of the job into the registers (e.g., ACC, PSIAR, etc.). This step is important when switching between jobs to ensure that the execution continues from where it left off.
* The inner loop while (currentJob.getState() != 0) keeps looping until the current job is terminated. Its state is only set to 0 once the HALT instruction is passed.
* fetchInstruction(), executeInstruction() then perform fetch decode execute functions that consist of obtaining the instructions and executing the operation on the operand.
  + A clock.tick() occurs after each instance of an instruction being executed until it reaches TIME\_QUANTUM, where a context switch is automatically executed
  + This is also done if the yield Boolean is set to true.

- fetchInstruction(): Fetches the instruction pointed to by PSIAR from memory.

- executeInstruction(): Executes the instruction based on CSIAR.

-contextSwitch(): Executes a context switch where a job is saved and pushed to the back of the queue allowing another job to execute.

-loadState(Job): Loads the state of a particular job (sets the registers in SharkOS to that Job’s registers).

-saveState(Job): Saves the state of a particular (sets the registers in that job’s class to the new values from executing the instructions).

- Instruction methods (ADD(), SUB(), STR(), LDI(), LDA(), BRH(), CBR(), YLD(), END()): Perform various operations based on the instructions passed in

* Since the instructions aren’t loaded into a specific location and are dynamically placed, BRH() and CBR() were changed to accommodate this by having PSIAR += SDR so when passing instructions, locations could be relative to the PSIAR rather than direct (PSIAR = SDR)

2. Memory Class

This class simulates the system's memory and provides methods for memory operations. The memory is written as a 1024 x 2 array where the first array holds all the addresses and the second holds the converted opcode and operand.

Attributes:

- memory: The 2D array which will be representing memory cells, storing the opcodes and operands.

- allocated: An array (of Booleans) indicating whether each memory cell is allocated.

Methods:

- Memory(int size): Constructor method that initializes memory.

- allocate(int size): Allocates a contiguous region of memory for a program.

- isContiguousMemory(int start, int size): Checks if a region of memory is contiguous and unallocated.

- allocateRegion(int start, int size): Marks a region of memory as allocated.

- read(int address): Reads an instruction from memory.

- getOperand(int address): Gets the operand from a memory cell.

- write(int add, int opcode, int operand): Writes an opcode and operand to a memory cell.

- writeToMem(int address, int data): Writes data to a memory cell.

- clear(): Clears the contents of memory.

- displayMemoryContents(): Displays the full contents of memory.

3. SystemClock Class

This class manages the system's clock to keep track of time.

Attributes:

- currentTime: Stores the current time in clock ticks.

Methods:

- SystemClock(): Constructor method that initializes the clock.

- getCurrentTime(): Retrieves the current time.

- tick(): Advances the clock by one tick.

4. Job Class

This class represents a job or program that can be executed by the operating system.

Attributes:

- programName: The name of the program.

- arrivalTime: The time at which the job arrives in the system.

- jobID: A unique identifier for the job.

- state: The current state of the job (0 for finished, 1 for ready to execute).

-acc, sdr, psiar, csiar, tmpr, sar, ir, mir: Each job has their own registers to be saved and loaded while running the instructions in RUN\_SHARKOS()

Methods:

- Job(String programName, int arrivalTime): Constructor method that creates a new job.