Github Link: FeevaDVA/CMSC312OSsimulator (github.com)

CMSC312OSsimulator

This is an Operating Simulator made for my class CMSC 312 at VCU

For phase 3 I wanted to finish up many of the planned things that I had like adding a priority scheduler as the second scheduler for comparison. I tried to make the comparison more standard as much as possible. I like the second level for queueing that I added for my first scheduler. I like the first come first serve for the really long process over 2000 CPU cycles that it will send up to the second level. The round robin scheduler I made is extremely efficient now I feel with the multithreading, and it acts like a 2-core system but with one performance core with 1 thread which is the second level of the queueing. It acts as a 1 core 7 threads and 1 core 1 thread system. Its like what intel is doing with their newer CPUs where they are using a performance core that is 1 thread to optimize performance. Another thing that I accomplished well is the GUI I worked on it from the start, and it has yet to have any real issues I feel or that I have experienced. I added multi parent child relationships my first implementation of child's allowed for this and worked very well it didn't need any tuning it just worked, and cascade terminates. My IPC methods are good and directly use the process resources that they have. The first IPC that I have m1 shares the processes resource with another process and the second method m2 sends a message to the process in the end of the list.

The latest commit should have everything in it to be graded.

Installation

Download the OS simulator.jar and the Process Templates folder and place them in the same place. The jar needs to be able to access the Process Templates folder to make the processes so make sure they are in the same folder and run the jar.

As for java version the jar was made with the Java 17 development kit so make sure that Java 17 is downloaded. JDK 17 can be found here https://www.oracle.com/java/technologies/downloads/

For compiling I used IntelliJ to make and compile the project as well as their GUI builder so whatever dependencies you need for that you will need for this and the java JDK I used to be 17 so JDK 17

Usage

There is a start/stop button at the button along with a add button with two text fields

The first of which is the number of processes to add to the os

The add button will generate the processes for both scheduler1 and 2

The second is the template you want to generate these processes from right now there is only 8 templates 1-11

1 - short running calculate heavy process

2 - calculate heavy process with critical section

3 - balanced process

4 - calculate heavy process

5 - long calculate heavy process with critical section

6 - i/o heavy process

7 - a copy of the template 1 but with a fork in the middle

8 - a copy of the template 2 but with a fork after the critical section

9- a copy of the template 8 but with a fork before and after the critical section for multilevel parents (only makes multilevel parents on scheduler 1)

10 - A copy of template 9 but with the first method m1 in it

11 – A calculate heavy process but with the second method m2 in it

The average turnaround time is shown at the bottom with the average wait time which is displayed in seconds for scheduler 1 and 2 respectively for comparing the efficiency of them

Next to this is memory for each scheduler they both have 1024 in memory each process has a randomly generated amount of memory that they take up

Each process will show its number/pid, state, remaining cycles, total memory, priority, and message if it has one until it is terminated where it will then show its turnaround time in seconds

The simulator can be started and stopped via the start/stop button.

The three section of list is split into ready list, waiting, and all the processes.

Documentation

Phase I -

Scheduler Algo: Round Robin

- I implemented it so that it will keep all the active processes in a list and go through them one at a time and decrement them by the quantum simulating running them it will run them for 2ms before moving to the next one.

Scheduler Algo: Priority

- The second scheduler algorithm that I choose to use for my comparison was Priority. Every process is assigned a random priority number between 0 – 5 with 0 being the highest and 5 being the lowest priority. It will sort them by priority then run through and complete the processes from highest to lowest.

PCB

- For this I used a class that holds all the information for the processes and the tasks it keeps track of everything that a process has.

Critical Sections

- I added a P and a V that denotes the critical section start and finish in the process templates

Making Processes

- I use multiple text docs that hold the rand of cycles and memory for each task then generates processes based on them. It goes through the text doc line by line. It generates processes based on the number of the amount the user enters and the given template number

UI/GUI

- I used the IntelliJ GUI builder, and it came out looking good I also have a plan for making it better for phase 2 and 3. I decided to go with a GUI instead of a UI because it felt easier for me to understand making it. So, there is no commands for it. It has a three-list system starting from the left we have the ready queue list, the waiting queue list, and all the processes list. Each process is show by PID, Status, Remaining Cycles, and then Turn around time if it has completed. The GUI also has the average turnaround time and waiting time at the bottom it also has text fields to enter the number of processes and template numbers to add. The start and stop button start and stop the thread running the scheduler. There are two tabs at the top for switching between the two schedulers one is round robin and the 2nd is shortest remaining time first

Phase II -

Threads

- For threads what I did was made 8 java threads that will be able to take in the process and when started it will run the process for the number of cycles that it needs in this case the number is the quantum. It will run through moving through the tasks until it reaches a process method such as FORK, P, or V then it will set it back to the ready state

Fork

- For forking I added it into the process template, and it will fork when it reaches the command. When it forks it will make a new process and be labeled child the Pid will be 1000 more than the parent process. The child will have the tasks that are left from when the parent called fork. The child also will not cost any memory as it shares the resources with the parent. The parent will be placed in wait until the child finishes.

Memory

- For memory each of the schedulers has 1024 memory and the processes have a randomly generated amount of memory based off the tasks in them. It will decrease the memory number by the process memory when the process is loaded into ready showing that it is loaded and ready to execute. At termination it will add the memory back. A process will not run if there is not enough memory.

IO Interrupts

- In the thread while a process is being ran there is a 1 in 10 chance at the beginning for the process to put into wait simulating a random io interrupt event. It is put into wait for 30 cycles before resuming normal operation

Phase 3 -

Process Resources

- For my process resources I made a resource class that every process has that has a random number to access and a string called message. Message is for passing messages to other processes or for a process to share the message.

GUI

- My GUI is very simple it has a list for each queue that updates in real time. You can switch between the two scheduling methods and compare them using the bottom bar. The bottom bar also lets you add any number of processes you want and choose from which template you want. The bottom bar also lets you monitor the memory.

Two IPC methods

- The first IPC method I made is called m1 and its in template 10. This method shares the current resource with another process ahead in the queue. The second IPC method I made is m2 and its in template 11 and it sends a message to the last process in queue to display.

Multi-Level Parent Child Relationship

- My multilevel parent child relationship will make all the parents wait until the last child of the tree finishes then they will cascade complete.

Two Schedulers and comparison

- The two schedulers I have are Round Robin and Priority and I try to sync them in terms of speed as much as I can but the Round Robin one is going to be faster since its multi-threaded.

Process Priorities

- Each process has a randomly assigned priority from 0 – 5 with 0 being the highest and 5 being the lowest. The scheduler Priority takes this information and sorts them from highest to least to complete.

Multi-level queue scheduling

- I added a second level on my round robin scheduler. The second level is a first come first serve scheduler that will be sent processes of over 2000 CPU cycles. It will then send these to a special thread for it thread8.

Multi-Threading and multicore

- For multi-threading I have 7 threads in 1 core if you look at the actual scheduler class as a CPU. My scheduler class essentially works as a sort of CPU and works through the processes with the threads that it has.
- The second level of my first scheduler can then be seen as a second core to get assigned tasks with a single thread.

- The second scheduler I have is not multi-threaded but can be seen as a single core CPU.

Paging

- Paging is not fully implemented but I added a frame allocation method and proportional page assignment I didn't have enough time to fully get proper paging implemented.

- Class: Scheduler
  - Variables:
    - private static final int quantum
      - the quantum for the round robin scheduler
    - private final ProcessList list
      - houses the PCB
    - private final mainGUI main
      - stores the GUI so that we can update lists
    - private int cycleCount
      - counts the cycles(not yet needed)
    - private int runningProc
      - returns the amount of currently running processes used to make sure that we can assign a process to a thread.
    - private int semaphore
      - the semaphore value I use it a bit different its 1 if there is a process in the critical section its 0 if not
    - private int procCrit
      - the process currently in the critical section
    - private volatile boolean exit
      - tells the simulator to stop or not
    -
  - Constructor: public Scheduler(ProcessList l, MainGUI m)
    - Desc: This creates the scheduler object which extends a thread so it acts as a runnable thread
  - Method: public void setExit(boolean e)
    - Desc: This will set the boolean exit to the given boolean e

    - Method: public void run()
        - Desc: This is the main loop of the class this will while exit is not true run a scheduler algorithm on the list of processes from the class processlist the scheduler algo that it is running as of now is round robin it will try to run at 20 CPU cycles per 2 milliseconds it will call the updateProcess method and pass the current process along to it
    - Method: public void updateProcesses(ProcessList.Process p, int i)
        - Desc: the method will update the current process depending on the state that it is in and it will assign the process to a thread if it is ready to be ran
    - Method: public void assignThread(ProcessList.Process p)
        - Desc: this method will assign the given process to a thread that is currently not running
    - Method: public void setSemaphore(int i)
        - Desc: sets the semaphore to i
    - Method: public void setRunningProc(int i)
        - Desc: sets the runningProc to i
    - Method: public int getSemaphore()
        - Desc: returns the semaphore
    - Method: public int getRunningProc()
        - Desc: returns the runningProc
    - Method: public ProcessList getPCB()
        - Desc: returns the ProcessList list
- Class: Scheduler2
    - Constructor: public Scheduler(ProcessList l, MainGUI m)
        - Desc: This creates the scheduler object which extends a thread so it acts as a runnable thread
    - Method: public void setExit(boolean e)
        - Desc: This will set the boolean exit to the given boolean e
    - Method: public void run()
        - Desc: This is the main loop of the class this will while exit is not true run a scheduler algorithm on the list of processes from the class processlist the scheduler algo that it is running Shortest remaining time and it will keep checking for the lowest remaining time process to run

- Class: ProcessList
    - Variables:
        - private int countProcess
            - Keeps count of the processes to assign them a pid
        - private List<Process> processlist
            - The list of all the processes that aren’t terminated
        - private List<Process> terminatedList
            - The list of all terminated processes
        - private int memory

            - Keeps track of and sets the memory
- Constructor: public ProcessList()
    - Desc: creates a list for processes, the count of processes, and a list for terminated processes
- Method: public void generateProcesses(int count, int temp)
    - Desc: generates a given number of processes of a given template number and adds them to the process list
- Method: public List<Process> getList()
    - Desc: returns the list of processes
- Method: public List<Process> getTerminatedList()
    - Desc: returns the list of terminated processes
- Method: public Process getProcess(int n)
    - Desc: returns the process at given location n from the process list
- Method: public void removeProcess(int n)
    - Desc: removes the process at location n in the process list
- Method: public setFrames(int f)
    - Set every process with an allocated amount of frames/pages it would have been proportional how many frames they got.

- Class: Task
    - Variables:
        - private String taskName
            - stores the name of the task
        - private int time
            - stores the time/cycles of the task
        - private int memory
            - stores the memory needed for the task
    - Constructor: public Task(String name, int t, int m)
    - Desc: the constructor for the task object that takes in a name, time, and memory of the task
    - Method: public int getTime()
        - Desc: Returns the time/cycles of the task
    - Method: public void setTime(int t)
        - Desc: sets the time/cycles of the task to the int t
    - Method: public int getMemory()]
        - Desc: returns the memory of the task
    - Method: public String getTaskName()
        - Desc: returns the name of the task

- Class: Process
    - Variables:
        - private int number, thread

        - Stores the number/pid of the process and the thread that it is assigned to
    - private double burstTime
        - stores the estimated burst time of the application
    - private long arrivalTime, completionTime
        - stores the arrival time and completion time of the process in milliseconds
    - private String state
        - stores the state of the process
    - private List<Task> taskList
        - stores a list of all the task for the process
    - private int currentTask
        - stores the location of what task the process is currently on
    - private boolean parent, child, running
        - stores if it is a parent process and child process and a running process
    - Process parentProc
        - Stores a reference to the parent process if it is a child process
- Constructor: public Process(int tempNum, int num)
    - Desc: Constructor for the process class that takes in the template number for the process and the number of the process also generates the tasks for the process
- Constructor: public Process(Process P, int pos)
    - Desc: constructor for making a child process. Copies the tasks from the parent p after the fork and inserts the child process after the parent. The pid is 1000 more than the ppid
- Method: private void generateTasks(int tempNum)
    - Desc: generates a task from the template number then uses the file for the template to make all the tasks from it
- Method: public void setArrivalTime(long n)
    - Desc: sets the arrival time of the process in milliseconds with the given long n
- Method:public void setCompletionTime(long n)
    - Desc: set the completion time of the process in milliseconds with the given long n
- Method:public long getTurnAroundTime()
    - Desc: returns the turnaround time in milliseconds of the process which is completion time - arrival time
- Method:public long getWaitTime()
    - Desc: returns the wait time of the process in milliseconds which is turnaround time - burst time
- Method:public int getTotalMem()
    - Desc: returns the remaining memory of the process
- Method:public int getTotalCycles()

            - Desc: returns the remaining cycles of the process
        - Method:public Task getCurrentTask()
            - Desc: returns the current task of the process
        - Method:public boolean nextTask()
            - Desc: return true if there is a next task and moves to it returns false if there is no next task
        - Method:public void updateState(String newState)
            - Desc: updates the state of the process to the string newState
        - Method:public String getState()
            - Desc: returns the state of the process
        - Method:public int getNumber()
            - Desc: returns the number of the process

- Class: Threads
    - Constructor: public Threads()
        - Desc: This is the default constructor it sets all variables to null or 0
    - Constructor: public Threads(ProcessList l, ProcessList.Process p, int c, Scheduler s)
        - Desc: this sets the list to l, proc to p, cycles to c, and scheduler to s
    - Method: public void run()
        - Desc: this is the main run method for the thread it will take the given process and to calculate task until it reaches the end of the given cycles it also has a 1 in 10 chance to cause an io interrupt

- Class FirstComeThread
    - Constructor: public FirstComeThread(ProcessList l, mainGUI m)
        - Desc: this creates the new thread
    - Method: public void setExit(boolean e)
        - Desc: sets the Boolean exit to e
    - Method: public void run()
        - Desc: runs the priority algorithm solving the first process in the first come list for it will update all the processes in it list and run their Io and wait.

- Class: MainGUI
    - Constructor: public mainGUI()
        - Desc: makes a new ProcessList called pcb and starts the thread scheduler initializes all the text and button fields also the scrollable lists
    - Method: public JButton getStartButton()
        - Desc: returns the JButton startButton
    - Method: public static void main(String[] args)
        - Desc: starts the MainGUI by making a new object

    - Method: private int getTempNumber()
        - Desc: returns the template number entered the text field of a0TextField1
    - Method: private int getNumberProcess()
        - Desc: returns the number of processes to be added from the text field of a0TextField
    - Method: public void updateList()]
        - Desc: updates the three scrollable lists with all the processes from the process list of the PCB and organizes them into their columns it will also update the average turnaround time and wait time
    - Method: public void updateList2()
        - Like updateList() but for scheduler2 and is in another tab
- Class: Resource
    - Constructor: public Resource()
        - Creates the resource with a random number and a empty message
        - Has getters and setters for these