

SageOS Documentation

Class Information

Process.....	Page 2
PCB.....	Page 3
OS.....	Page 4
GUI.....	Page 5
CPU.....	Page 6
Extractor.....	Page 7
ProcessState.....	Page 7
ProcessScheduler.....	Page 8
ThreadListener.....	Page 9
Dispatcher.....	Page 10
RowColorRenderer.....	Page 11
Memory.....	Page 12
Cache.....	Page 13
IODevice.....	Page 13
Keyboard.....	Page 14
Mouse.....	Page 14
Monitor.....	Page 14
AnalyticsWindow.....	Page 14

How To Guide

How to Use the Program.....	Page 15-17
Troubleshooting.....	Page 17
Small bug.....	Page 17

Classes

Process

Description: Process is a class which simulates a process. As of now, it is a very simple class with few methods and instance variables but it will be expanded upon in later project parts

Instance Variables

PCB pcb – a PCB which is created to hold process information for each process

ArrayList<String> textSection – the text section of the process which holds program instructions

Constructors

Process() – creates a new process with a PCB and empty textSection

Methods

int getTotalCylceCount() – returns int value of total cycles from instructions in the text section

void setTextSection(ArrayList<String> textSection) – takes an ArrayList<String[]> as a parameter and sets it as the value of textSection

PCB

Description: PCB is a class which simulates the Process Control Board of a Process. It holds important information of a process such as process id, process state, etc.

Instance Variables

ProcessState state – current state of a process

int processID – process ID of a process

int programCounter – address of the next instruction to be executed

int parentPID – PID of parent process, -1 if process does not have a parent

ArrayList<String> registers – registers this process occupies

ArrayList<String> IOStatus – list of I/O devices this process is using

int currentTaskCyclesRemaining – number of cycles remaining in current task

int currentInstruction – index of current instruction in textSection

boolean inCriticalSection – indicates if process is in critical section or not

Constructors

PCB() – creates a new PCB which sets processID to lastPID +1, state to ProcessState.NEW, and programCounter to 0

Methods

ProcessState getState() – returns state

void setState(ProcessState state) – sets instance variable state to parameter state

void incrementPC() – increments value of programCounter

void setParentPID(int parentPID) – set parent PID

void setTextSection(ArrayList<String[]> textSection) – set textSection of PCB

void getTotalCycleCount ()– returns total cycle count of all instructions

void incrementCurrentInstruction() – increments current instruction

String[] getCurrentInstruction() – returns current instruction

int getPID() – returns processID

OS

Description: executable class for SageOS. Runner scans for program files and creates an instance of GUI

Instance Variables

static int lastPID – keeps track of last assigned process ID so that all process IDs are unique

static ProcessScheduler scheduler – instance of process scheduler for the OS

static Dispatcher dispatcher – instance of dispatcher of OS

static CPU cpu – instance of cpu of OS

static ArrayList<Process> processes – list of created processes

Constructors

N/A

Methods

static void getPrograms(File file, ArrayList<File> allPrograms) – takes parameter file as name of the folder to search, collects all files within the folder, and adds them to allPrograms

static void createProcess(int parentPID, File program, Boolean isChild) – creates a process from a given program file. If the process is a child, omit all FORK operations from text section

static void main(String[] args) – main method for the program. Runs getPrograms method then creates an instance of GUI, CPU, Dispatcher, and ProcessSchedulers

GUI

Description: GUI creates a JFrame window and allows for visualization of SageOS. A JTable on the right half of the window displays all processes and information about the processes in real-time as they execute

Instance Variables

JFrame frame – a JFrame which contains all GUI components

JPanel panel – a JPanel which contains all action buttons and a JTextField

JTextField numProcessesText – a JTextField which allows users to enter number of processes to run

File program – current program selected

int numProcesses – number of processes to be executed

JButton executeProgram – JButton which creates number of processes from selected program and executes them, if input is valid and a program is selected

static JTable table – Jtable which shows process information about all current processes

RowColorRenderer cellColorRenderer – allows to set custom colors of rows on JTable

Final Color darkestDark, primaryDark, secondaryDark, tertiaryDark, blue, orange – colors used for components in the JFrame

JButton selectedProgramButton, selectedAlgorithmButton – current selected program/algorithm buttons

int selectedAlgorithm – int value of selected algorithm

Constructors

GUI() – creates a JFrame window and fills it with all necessary components

Methods

initTable() – initializes JTable with header

updateTable(int pid) – updates a process on the JTable to current process values

addChildProcessToTable(int parentPID) – adds a child process to JTable

void execute() – add necessary number of rows to the JTable. Creates processes and adds them to jobQueue of ProcessScheduler for cpu to execute

void formatButton(Button b) – takes parameter b and formats it to specified look

void updateSelectedProgramButton – updates selectedProgram and appearance of selected button

void createProgramButtons(ArrayList<File> programNames) – creates button for each program file and adds them to panel

CPU

Description: CPU class simulates a CPU. It is in control of the cycle count and has an execute method which executes processes

Instance Variables

int cycleCount – cycle count of the cpu

final int coreCount – number of cores of cpu

final int threadsPerCore – number of threads per core

int[] threadsAvailableInCore – number of threads in each core not being utilized

Boolean processToExecute – Boolean value that stops cpu clock when false

Static int sleepDuration – duration of sleep after each clock cycle

Static ScheduledFuture<?> clock – called advanceCycleCount every sleepDuration milliseconds

Final Semaphore criticalSectionLock – semaphore for process entering critical section

Constructors

CPU() – creates an instance of CPU and sets its cycleCount to 0

Methods

void Execute(Process pr, int cycles) – this method receives a process to execute from Dispatcher, runs the process on a thread, and updates GUI accordingly. **Note:** currently the processes are executed sequentially. I understand that we are supposed to implement multi-threading and intend to do this in a later iteration of the project

startClock() – creates ScheduledExecuterService that calls advanceCycleCount() every sleepDuration milliseconds to simulate a CPU clock

stopClock() – stops the CPU clock

advanceCycleCount() – increments cycleCount and notifies dispatcher of each cycle to update waiting queue

Extractor

Description: Extracts instructions from a given program and assigns random number of cycles to each instruction. This class is used to generate the textSection for a process

Instance Variables

N/A

Constructors

N/A

Methods

static ArrayList<String[]> getInstructionsWithRandomCycleCounts(File program, Boolean isChild) – takes parameter program, collects all instructions, then assigns a random number of cycles to each instruction. If param isChild is true, removes all Fork instructions to prevent infinite loop

ProcessState

Description: Enum class for all process states (NEW, READY, RUNNING, WAITING, TERMINATED)

Note: this class has no instance variables, constructors, or methods because it is an Enum

ProcessScheduler

Description: class that simulates a Process Scheduler. This class will be in charge of deciding when and what order processes in the system will run.

Instance Variables

List<Process> jobsQueue – Queue of all processes in the system that are not in ready or waiting queue

List<Process> readyQueue – Queue of all processes ready for execution

List<Process> deviceQueue – List of I/O devices

final static int SJF_ALGORITHM – constant value for shortest job first algorithm

final static int RR_ALGORITHM – constant value for round robin algorithm

int algorithm – current scheduling algorithm

static int Quantum – Quantum value for Round Robin

Constructors

ProcessScheduler() – creates an instance of ProcessScheduler and initializes all queues

Methods

void setSchedulingAlgorithm(int algorithm) – set scheduling algorithm

int getSchedulingAlgorithm – returns scheduling algorithm

void scheduleNewProcess(Process p) – adds parameter p to jobsQueue if it is not contained in jobsQueue

void processReady(Process p) – adds parameter p to readyQueue and removes it from jobsQueue

void newIORequestFrom(Process p, String device) – adds process p to correct device from deviceQueue and sets its state to WAITING

void removeFromIOQueue(Process p, String device) – removes process p from correct device in deviceQueue

static Process getNextReadyProcess() – pops Process from readyQueue

void shortestJobFirst(Process pr) – takes parameter Process and uses shortestJobFirst algorithm to insert it into the queue where it belongs (currently only by comparing next burst of processes)

void roundRobin(Process pr) – adds new process to end of queue

ThreadListener

Description: interface implemented by Dispatcher to know of certain events pertaining to threads

Constructors – N/A

Instance Variables – N/A

Methods

void burstFinished(Process pr) – notifies Dispatcher that a process has finished a burst

void clockCycled() – notifies Dispatcher that the CPU clock has cycled. Used to update remaining waiting times of processes in deviceQueue (waiting queue)

Dispatcher

Description: Dispatcher is in charge of removing process from the ready Queue and offering them to the CPU for execution

Constructors

Dispatcher() – basic constructor for creating an instance of Dispatcher

Instance Variables

N/A

Methods

void dispatchProcesses() – while readyQueue has processes cache is available, and CPU has available threads, this method passes a process to CPU to execute for numCycles. If ProcessScheduler is using round robin, it passes Quantum as number of cycles. If ProcessScheduler is using shortest job first, it passes currentInstruction cycles

synchronized void burstFinished(Process pr) – gets current instruction from parameter pr and updates its status accordingly. Either sets state to TERMINATED, adds it to deviceQueue, or adds it to readyQueue. Also responsible for creating child processes from Fork instruction, cascading termination of child processes from a parent, and updating critical section status. Finally, it calls dispatchProcesses()

synchronized void clockCycled() – calls cycleProcesses() for each device in deviceQueue

RowColorRenderer

Description: Overrides rendering of cell components of the JTable so that rows are color coordinated according to the process state and whether it is a child process or not. Much better for visualization purposes

Constructors

RowColorRenderer(Color childProcessColor) – basic constructor

Instance Variables

Color childProcessColor – color of child process

ArrayList<Integer> childRows – indices of child process rows

Methods

addChildIndex – adds a child index to childRows

clear() – removes all child indexes from childRows

Component getTableCellRendererComponent(JTable table, Object value, Boolean isSelected, Boolean hasFocus, int row, int column) – returns a customized component cell

Memory

Description: Memory is a class that simulated main memory of a computer. This class is responsible for creation and management of memory in the simulation.

Constructors

Public Memory(int memorySize) – basic constructor which takes int memorySize as a parameter

Instance Variables

Private int availableMemory – memory left in mainMemory

Private int totalMemory – total memory in mainMemory

Private final int baseRegister – last allocated position of OS memory

Private int lastAllocatedNum – last position of allocated user process memory

Private ArrayList<int[]> memoryBlocks – list of all blocks the memory has been divided into.

Each int[] object contains index of the block, beginning location of memory block, and ending position of memory block

Private ArrayList<int[]> freeMemoryBlocks – list of memory blocks that have been made and are no longer allocated to a process

Methods

Public void allocateMemoryBlock(Process pr) – Takes a process as a parameter and allocates memory to that process. If free blocks exist, this method will find firstFit for the process, else it creates a new memory block. If there's no room to create a new block and there's not a freeBlock that is large enough, the process is not allocated into memory

Public void deallocateMemoryBlock(Process pr) – Deallocates the memory block held by the given process

Public int[] createMemoryBlock(int processSize) – creates and returns a memory block from lastAllocatedNum + 1 to lastAllocatedNum + processSize to allocate to a process

Cache

Description: simulation of cache

Constructors

Public Cache(int cacheSize) – basic constructor. Takes cache size as parameter

Instance Variables

Public int availableCache – remaining cache available to be allocated to a process

Public int totalCache – size of total cache

Methods

Public void allocateCache(Process p) – decrements available cache based on size of process

Public void deallocateCache(Process p) – deallocated cache that a process was previously using

IODevice

Description: Abstract class that is extended by all IO devices (keyboard, mouse, monitor)

Constructors

N/A

Instance Variables

Public volatile List<Process> queue – queue of processes

Methods

addProcess(Process p) – add a process to the queue

removeProcess(Process p) – remove a process from the queue

public void cycleProcesses() – for each process in the queue, decrements its currentCyclesRemaining and increments its programCounter. If currentTaskCyclesRemaning = 0, it increments the instruction and calls burstFinished(pr)

Keyboard, Mouse, Monitor

Description: Classes that simulate an IO device

[see IODevice for description and details]

AnalyticsWindow

Description: a second window which display information about the CPU and main memory during execution

Constructors

Public AnalyticsWindow() – default constructor. Inits the window and its children components

Instance Variables

Private JFrame frame – frame of the analyticsWindow

Private JPanel panel – panel which contains children views

Private JLabel memoryView1-3, cpuView1-2: labels containing information

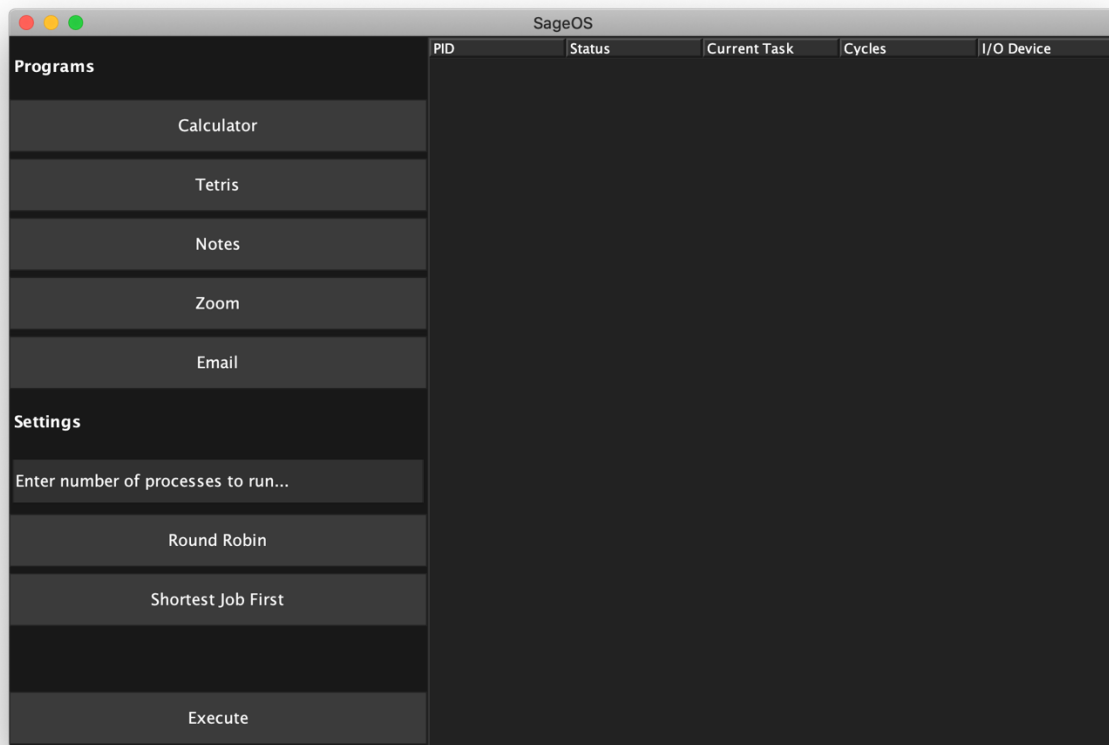
Private final Color darkestDark, primaryDark – colors used in the window

Methods

updateStats() – updates information in all children views each CPU clock cycle

Running the Program

SageOS is written in Java. To open the program, simply import the project into Eclipse or another Java IDE, open the Runner class, and execute it

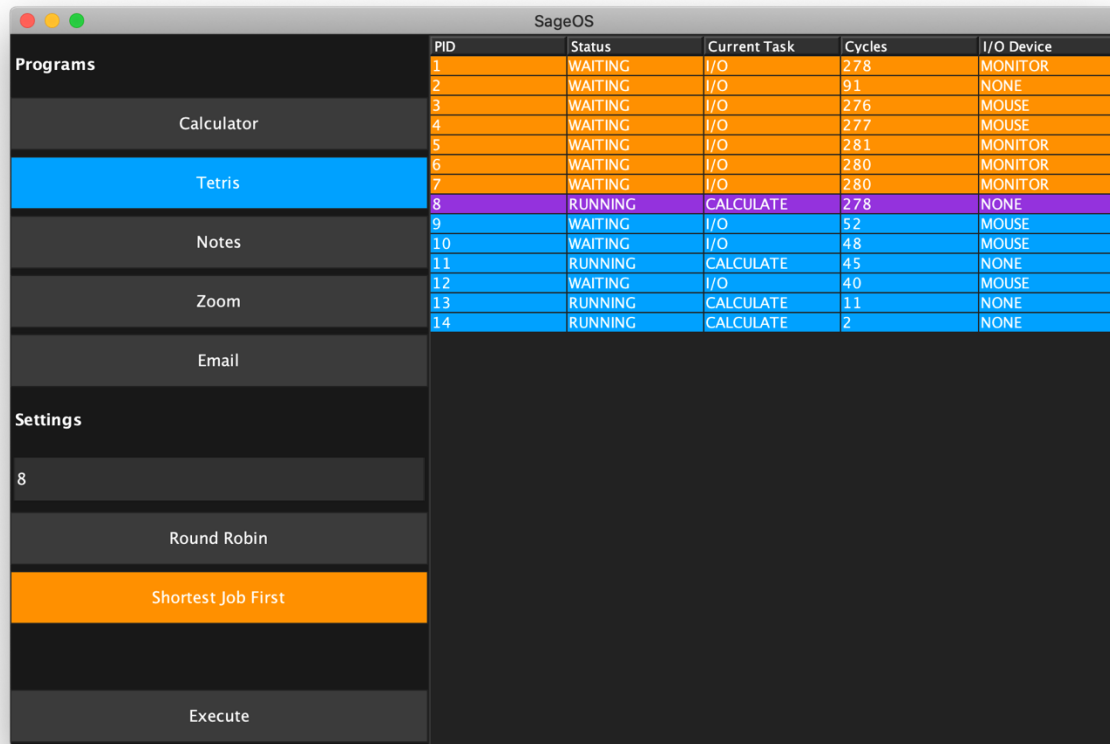


Upon executing the Runner class, a new window like this should display on your screen. If not, see troubleshooting tips

Instructions:

1. Select a program by clicking one of the program buttons (the button should turn blue when selected)
2. Select the text box which says "Enter number of processes to run..." and enter the number of processes you would like to create from the selected program
3. Optional: select a scheduling algorithm (SJF used by default if nothing selected)
4. Press the "Execute" button
5. That's it! The processes should then show up on the right side of the screen

After following the above instruction, the table should populate with processes and the CPU class will execute each process sequentially (for now, but multi-threading will come in future iterations) and update as process instructions are executed. During execution, it will look like this:



The screenshot shows a window titled "SageOS" with a sidebar menu on the left and a process table on the right. The sidebar menu includes sections for "Programs", "Settings", and "Execute". The "Programs" section lists Calculator, Tetris, Notes, Zoom, Email, and Settings. The "Settings" section includes a value of 8, Round Robin, Shortest Job First (highlighted in orange), and Execute. The process table on the right has columns for PID, Status, Current Task, Cycles, and I/O Device. The table contains 14 rows of data, with the 8th row (PID 8) highlighted in purple, indicating it is currently running.

PID	Status	Current Task	Cycles	I/O Device
1	WAITING	I/O	278	MONITOR
2	WAITING	I/O	91	NONE
3	WAITING	I/O	276	MOUSE
4	WAITING	I/O	277	MOUSE
5	WAITING	I/O	281	MONITOR
6	WAITING	I/O	280	MONITOR
7	WAITING	I/O	280	MONITOR
8	RUNNING	CALCULATE	278	NONE
9	WAITING	I/O	52	MOUSE
10	WAITING	I/O	48	MOUSE
11	RUNNING	CALCULATE	45	NONE
12	WAITING	I/O	40	MOUSE
13	RUNNING	CALCULATE	11	NONE
14	RUNNING	CALCULATE	2	NONE

After all processes have terminated, feel free to create new processes by following the same steps mentioned above.

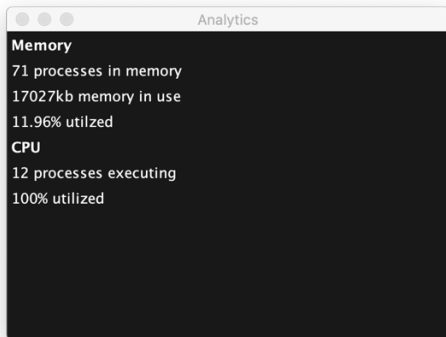
Note: Please let all processes enter the “terminated” state before trying to create more new processes. I have not implemented a feature to end current processes if the user starts more processes while others are executing, so the program will not act as expected

Process Table Color Guide:

- Red – terminated
- Purple – currently executing (running)
- Orange – in waiting queue (waiting)
- Blue – child process (turns red when terminated)

Analytics Window

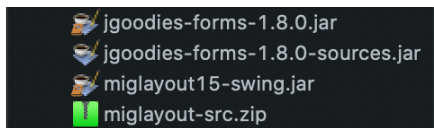
In addition to the main window, a smaller window will appear when the program is run. This window simply displays live information about the CPU and memory as processes execute. The window should look like this:



Troubleshooting

There should be no problem with running this program, however I have thought of a few issues that could possibly arise if the project is not imported/exported correctly

- If an error occurs while attempting to run the program, make sure that the following jar files are located in the project:



- If no program buttons show up, ensure that there is a folder labeled "program files" and that it contains a few .txt program templates, as the buttons in the UI are created dynamically based off of what files are located in the "program files" folder

Bug

I fixed the last bug; all programs work fine now. However, there is a new bug where there is always at least 1 process that gets stuck during execution, either in WAITING or RUNNING state. However, this only started when I implemented memory so I'm sure that is the source of the issue and I will look over my code to hopefully resolve the issue