OPERATING SYSTEM SIMULATOR DOCUMENTATION

OVERVIEW

Written in Java, an object-oriented programming language, Operating System Simulator emulates the process and resource management/prioritization aspects of a computer. The simulation includes hardware (CPU, memory, IO devices), BIOS (bootloader, kernel, operating system), and user interface (terminal), divided into packages. The .jar executable provided in the main directory will launch the GUI and source code.

PACKAGE: BIOS

The BIOS portion of the code contains a simulated handoff from the bootloader to the kernel once the CPU is started, then to the operating system upon launch.

PROCESS CONTROL BLOCK

The process control block constructs and permanently holds the definition of a process instance (process ID, child process pointer, program name, memory limit, priority, CPU start time, maximum CPU cycles, and program assembly list) upon creation. The child process pointer points directly to the process's child process, in a linked-list methodology, meaning infinite generations of children can stem from one initial parent. When no child process exists, the child process pointer points to null. The process control block also keeps track of the process state, program counter, instruction counter, CPU cycles used, last burst, and weighted priority, which are all dynamically updated depending on the current system state and the scheduler demand.

IO DEVICE IDENTIFIER

The IO device identifier class enumerates the available IO devices in the system that can be manually inputted to be dynamically broadcasted to the rest of the system, therefore, new IO devices can be named and indexed in this class without affecting the functionality of the kernel.

The default IO devices are correspondent to the default software included in the OS. The NULL device serves as a placeholder for processes that aren't in active need of IO devices:

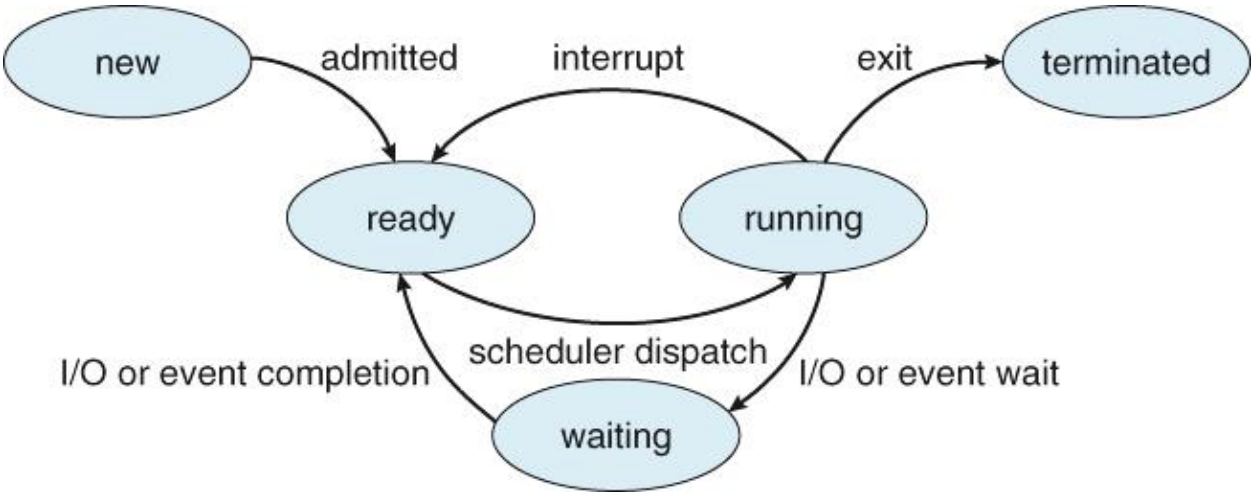
- NULL
- KEYBOARD
- NETWORK
- STORAGE

SYSTEM CALLS

The system calls class handles the communication of the physical system vs the logical system. Processes that are queued by the user in the GUI are interpreted by this component to be encoded in a logical manner. Child process creation dictated by keywords in the software are also read to construct new processes to be added to the scheduler. Reference [Software\Child Process]

SUB-PACKAGE: SCHEDULERS

The scheduling architecture is built around an inner-outer ideology. The active processes are contained in a dynamic queue and assigned states to identify the current status of a given process. Processes switching states after CPU cycles or IO requests are managed by the short-term scheduler, while the long-term scheduler queuing and handing over processes to the short-term scheduler based on availability of system resources.



LONG-TERM SCHEDULER (OUTER)

The scheduler utilizes a weighted-priority first algorithm to schedule tasks using multiple instances of the short scheduler. The main purpose of the long-term scheduler is memory management - to properly manage available resources and swap memory if necessary. All new processes are piped into this scheduler and are initially queued using a first-come algorithm. Any processes waiting for CPU time are placed in the waiting queue and swapped to VRAM. The processes in the waiting queue is weighted by the process comparator and ordered respectively. The processes which get committed by the system are placed in the short-term scheduler's ready queue.

SHORT-TERM SCHEDULER (INNER)

The short-term scheduler comes into play whenever the task requires state changes for CPU time and IO device handling. Once the CPU time is exhausted for a certain process, the task is then sent back to the long-term scheduler's waiting queue. For IO devices, if there are multiple sent in by the long-term scheduler, the task is placed in the IO queue and tasks that are then waiting for IO input are placed in the IO wait queue, with their respective process states.

SUB-PACKAGE: HANDLERS

Handlers are communicated by the CPU via processor interrupts. Based on the flag requested, a particular handler comes into play. Reference [Package: CPU\Processor Interrupts]

- Terminate Handler
- IO Handler
- IO Wait Handler
- Yield Handler
- Switch Handler

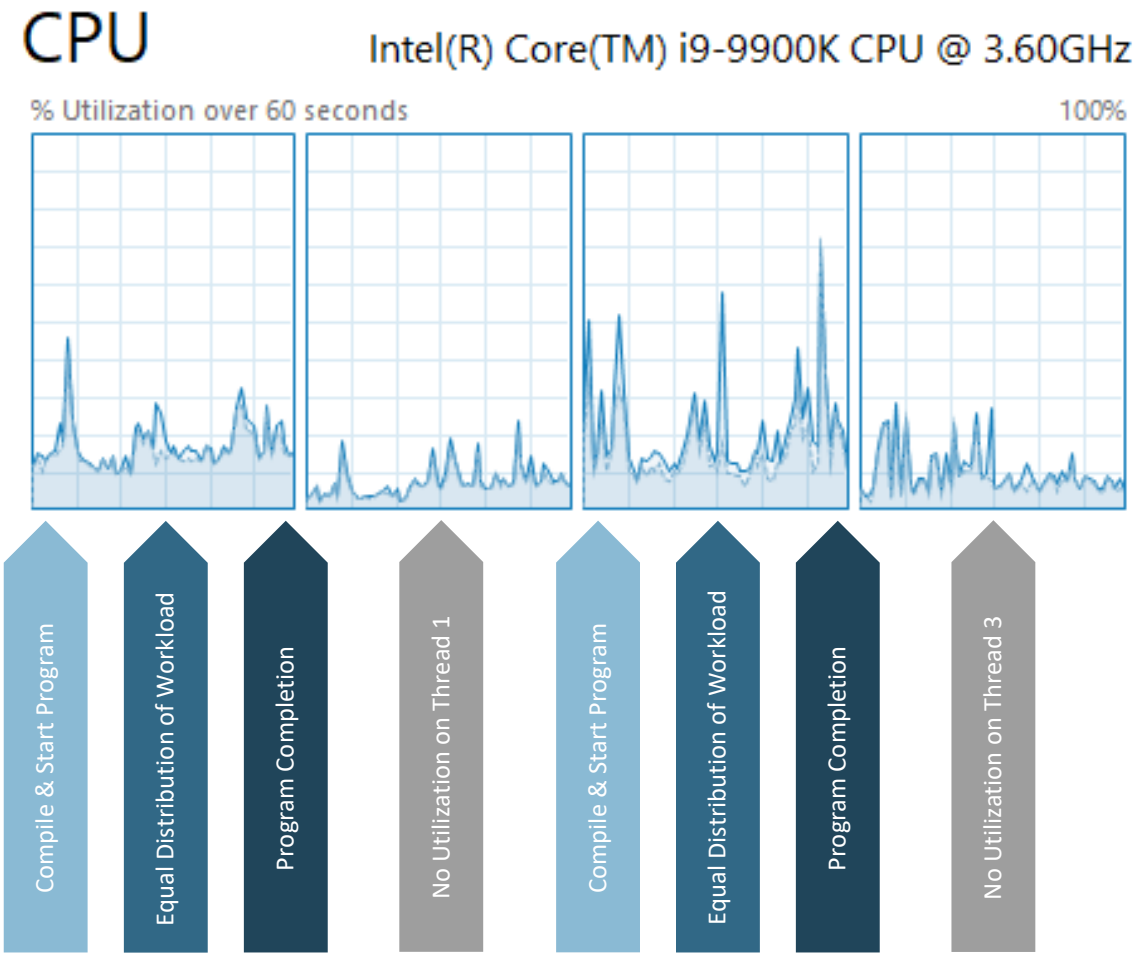
PACKAGE: CPU

The hardware portion of the code includes the classes that are most closely related to the CPU.

CPU AND MEMORY

The CPU class holds information relating to total memory (default 2MB), clock speed, and number of cycles. The CPU also requests the scheduler to index tasks and run tasks when they are bounced back. The CPU cycles as instructed by the system once initialized by the setup and started by the bootloader.

The CPU uses Java's multithreading procedure and can be dynamically adjusted to support more threads by altering the field variable to simulate a multi-core environment. Note that the GUI already allocates one dedicated thread as a requirement. The graphic below shows the Windows Task Manager displaying the synchronous use of threads 0 and 2 by the Java engine since two threads were set as the parameter. Reference [Package: User Interface\GUI\GUI Interaction Manual]



IO

The IO controller makes an interrupt system call to prioritize the SEM_IN and SEM_OUT instructions that are found in the assembly class and introduced by the assembler. The IO wait time is randomly generated with a range of 5 to 30 IO cycles to simulate the unpredictability of user input.

ASSEMBLER AND SEMAPHORES

The assembler interprets incoming tasks in a format that is readable for the kernel, essentially serving as an encoder for assembly language for this particular kernel. The assembly code is then placed in the process, holding information of instruction type and CPU cost. For the purpose of this simulator, the assembler generates a random value from the given seed in the software file for each CALCULATE instruction.

The assembler places the interpreted process text of the software templates by reading various commands in the following syntax:

- CALCULATE [integer]: CPU time using parameter as a random generator seed
- IO: IO instruction chain for SEM_IN, IO, SEM_OUT
- FORK: Create a child process
- MEMORY [integer]: Memory requirement
- PRIORITY [integer]: Initial/base priority

As shown, the assembler also bounds all IO processes with SEM_IN and SEM_OUT to simulate the critical sections with semaphore flags, preventing any other processes from accessing the IO controller while the current IO process is in the critical section. As for CPU bound processes, the semaphore protocol is handled by the waiting queue found in the long-term scheduler, meaning in the single-threaded nature of the simulation, another process cannot partake the same CPU resource as the current process.

PROCESSOR INTERRUPTS

The interrupt class handles incoming interrupt requests and hands them over to the respective handler in the “handlers” package.

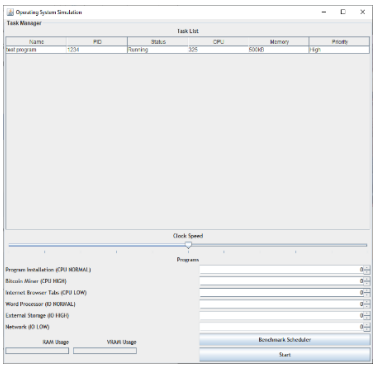
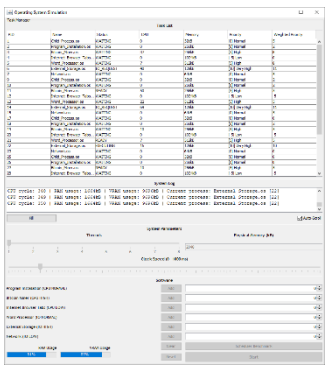
- [Flag 0] TERMINATE: Calls the Terminate Handler to terminate the process and its children
- [Flag 1] SEM_IN: Calls the IO Handler to request IO controller time
- [Flag 2] SEM_OUT: Calls the IO Handler to release the IO controller
- [Flag 3] IO_WAIT: Calls the IO Wait Handler to block the process until IO event is completed
- [Flag 4] IO_COMPLETE: Calls the IO Wait Handler to signal an IO event was completed
- [Flag 5] YIELD: Calls the Yield Handler to switch processes when the burst timer expires for the process

PACKAGE: USER INTERFACE

The UI is displayed in a similar manner to the process list found on Windows devices. The window is run on a .jar executable. The user interface package contains the shell and console, which is the virtualized underlying command line interface found on most modern operating systems.

GUI

The GUI portion is the code for the actual window that is interactable with the user. The GUI makes use of Java's JavaSwing API and contains information on current processes and the related information, the clock speed slider (incremented by 100ms ticks), and the program list (number of tasks to load determined by spinner).

Initial version (Left), Most recent version (Right)

GUI INTERACTION MANUAL

- Table: Displays each process instance by row (sorted by PID ascending) and is updated per CPU cycle. The columns can be moved left/right and resized freely. The following data points are itemized:
    - Process ID
    - Name
    - Status
    - CPU Usage
    - Memory Usage
    - Priority - Displays user-friendly indicator for process priority:
        - >= 10: Very High
        - >= 5: High
        - >= 0: Normal
        - >= -5: Low
        - < -5: Very Low
    - Weighted Priority - Calculated using a process age algorithm
- System Log: Displays activity monitor when interacting with the GUI and when certain crucial system calls are made
- Clock Speed: Slider adjusts the delay between each CPU cycle for a particular simulation run
- Physical Memory: Parameter to manually set installed physical memory
- Threads: Slider adjusts the number of system CPU threads to be utilized
- Software: The software input stream is an order dependent queue
    - How to queue processes:
        - Use the spinner to select how many instances of a particular software is desired
        - Press the "add" button
        - Repeat steps 1 and 2 in desired order of software queue
    - Start: Sends the created process queue to the system
    - Clear: Clears spinner fields but keeps process queue as is
    - Reset: Empties process queue
    - Scheduler Benchmark: Predetermined process queue to stress the scheduler

- RAM Usage: Percentage of physical memory in usage
- VRAM Usage: Quotient of swapped memory over committed page size represented by percentage
- Kill: Kills the current run of the simulation and returns the program to the initial launch state (multiple attempts may be required if the simulation is within a critical section).

The GUI will lock itself to prevent user intervention during a simulation run and toggle back to an editable state after a run has successfully completed or if the kill button is used. If the simulation returns an error code and hangs, the kill button can also be used as a forced toggle/reset switch to reset the simulation. Reference [Error Codes]

SOFTWARE

The Operating System Simulator has six built-in virtual tasks, that are accessible via the GUI, and cover every CPU/IO scenario. Custom programs can be added into the software folder in the main directory but must be run via console command. Memory, base priority, CPU usage seed parameters and line locations of the default software can be modified to test different scenarios.

PROGRAM INSTALLATION

The program installation task strains the CPU under a sustained load with normal priority flags. The program also generates a child process.

BITCOIN MINER

The Bitcoin miner task strains the CPU under a sustained load with high priority flags.

INTERNET BROWSER TABS

The internet browser tabs task strains the CPU under a sustained load with low priority flags.

WORD PROCESSOR

The word processor task sporadically uses the CPU in response to IO with normal priority flags.

EXTERNAL STORAGE

The external storage task sporadically uses the CPU in response to IO with high priority flags.

NETWORK

The network task sporadically uses the CPU in response to IO with low priority flags.

CHILD PROCESS

The child process task is a hidden process that serves as a template for the FORK instruction when creating a child process and can be used to recursively create more children.

ERROR CODES

The system reports error codes to the console log and halts the process, akin to BSODs in Windows operating systems, to simulate the kernel protecting the system from corruption/damage. These errors are expected and constantly checked for during the execution section in the simulation. Some are occasionally unavoidable due to the nature of Java and its thread/memory management.

Error Code	Issue	Type/Solution
CPU FAILED TO CYCLE	While the CPU may have initialized successfully, the CPU was unable to acquire the scheduled process.	System error: Reboot *Multithreading may also cause this error after finishing - ignore.*
INSUFFICIENT SYSTEM MEMORY	The system lacks the maximum physical memory for a particular process's memory requirement.	User error: Increase the physical memory parameter in the setup stage.
IO PROCESS NOT FOUND	The IO Handler did not find a waiting IO process despite being called.	System error: Reboot
IO BLOCKED	The IO Wait Handler determined that a requested IO device is in use and/or locked.	System notification: Usually ignore but could be the result of an erroneous IO device.
IO WAIT ERROR	The IO Wait Handler failed to promote a blocked IO process.	System error: Reboot
PROCESS LEAK ERROR	The system found a zombie process waiting after all processes were supposedly terminated.	System error: Reboot
SCHEDULER ERROR	The process state was failed to be recognized by the short-term scheduler after it was handed off.	System error: Reboot