CMSC 312 Project Part 1 Description 2022

Operating system simulator

Process management

"Program files" (also referred to as "templates") are text or .xml files that are used to create processes that will be then managed and run by our OS simulator. Therefore, "program file" must be treated as a template that can be used to create numerous processes with diverse parameters (e.g., by randomization of the values in the template). Each template (and thus each process created from it) will consist of a "mock" code written in a form of keyword operations, each representing the usage of a different resource managed by OS. For simplicity, only three types of operations are required:

- CALCULATE – When this line is read in, the simulator will run the process in the run state for the number of cycles specified as a parameter (i.e., occupy CPU for a given number of cycles, simulating the usage of CPU resource).
- I/O – This will put the process in the waiting state for a specified number of cycles.
- FORK – this will create a child process according to a selected parent-child management scheme.

Each "program file" must be created using different combinations of these commands (they can appear multiple times). Each command must have assigned number of cycles necessary for completion, as well as memory necessary for storing this command (in case of a paging model being implemented).

Example of a program file / template (this needs to be enriched with memory management in a later step):

Operation list:	min cycles	max cycles
CALCULATE	5	100
CALCULATE	25	50
I/O	10	20
CALCULATE	5	20
I/O	15	25

This template must be used as an input (seed) for a generator that will spawn a user-defined number of processes with randomized cycle length for each operation. Example of two processes created from the above template are:

Process #1		Process #2	
CALCULATE	46	CALCULATE	82
CALCULATE	33	CALCULATE	48
I/O	17	I/O	11
CALCULATE	11	CALCULATE	19
I/O	22	I/O	16

Each program must contain at least one critical section and at least a single critical section resolving scheme must be implemented. Critical sections can be implemented as tags encoding a number of operations that are considered as a critical section. Critical sections can be hard coded into a template or placed randomly in each created process. All information about each process must be stored in Process Control Block (PCB).

Process lifecycle

The processes created from templates must be managed by the OS simulator following a process lifecycle that is realized as switching among the possible states in which each individual process will be in a given moment:

- NEW – The program or process is being created or loaded (but not yet in memory).
- READY – The program is loaded into memory and is waiting to run on the CPU.
- RUN – Instructions are being executed (or simulated).
- WAIT – The program is waiting for some event to occur (such as an I/O completion).
- EXIT – The program has finished execution on the CPU (all instructions and I/O complete), releases resources and leaves memory.

Process scheduling

Student must choose and implement at least two different schedulers must and compare them with regard to their performance.

This program/simulator will incorporate a user interface so that she/he can control the flow of the operating system and observe the “running” of it for testing purposes. It must be possible to load a program or job file automatically/manually into the simulator thus to conduct the allocation of the program’s PCB and memory space. The user can also specify the number of cycles to run before pausing. Statistics of the simulator should be available upon request, such as number of processes in each state, journal log of the simulator, and what resources are currently being used.

General rules for the project

- This is an **open-ended project** and the design skills, capability of adapting to specifications where necessary, as well as imagination and creativity of solving the required tasks play a crucial role.
- Project must be fully functional, and the code of project must compile and run.
- Project must be a compound system of modules working together, not a collection of non-integrated parts.
- Project must be done individually, not code sharing or copying / modifying is allowed.
- Students are allowed to discuss and consult theoretical solutions and approaches among themselves.
- Source code must be delivered for grading, as well as an executable version of the project.
- Student is obliged to deliver a .pdf documentation of the project, discussing the implemented solutions, software, and hardware requirements for the project to run, as well as guidelines on how to run the project.