CS2506 Operating Systems II Lab 1 30/01/2024
Process management

Laboratory Goals

Students will analyse the concept of process in computing systems, its life cycle and scheduling strategies. The questions addressed by this lab are "what is a process?", "what are the states of its life cycle?", "what events trigger switches of processes between states?", "how are processes scheduled for execution and managed until completion?".
The expectation is that by the end of the lab, the student report will give answers to the above questions and will demonstrate an understanding of the process concept and how it is managed by the operating system.

1. The process entity

Computing systems are very diverse in terms of architecture, configuration and range of applications. While general purpose systems run all applications with acceptable performances, the embedded systems run dedicated code with performances required by their applications. Consequently, the process attributes and life cycle are adapted to the kind of targeted systems. In embedded systems, all processes should complete in less than a time slice (quantum). They need process ID but no information about open files or current directory. Therefore, their context is much simpler.

2. The process life cycle

The kernel creates the process whose first state is READY. As such, the process competes with other processes to get control of the CPU for its execution. When the current time slice (quantum) is finished the scheduler decides which process will switch to RUNNING state, etc. (see the lecture notes for the details).
It is important to note that in any computing system, a process will go through several states that make its life cycle.

3. The scheduler

The scheduler is the kernel service that monitors processes, hands over control of the CPU based on certain criteria, e.g., priority, and fixes contention situations (e.g., by priority inversion).
Multilevel feedback queues are an architectural solution for implementing dynamic priorities of processes. Each queue has a priority level and a given size of the CPU time slice. If user processes don't complete the execution while controlling the CPU for the time slice allocated to them, they will be returned in a queue with a lower priority and a bigger time slice. If they reach the lowest allowed priority queue, they will be scheduled there round-robin until completion. However, if a user process is blocked for the execution of I/O operations, when it will be returned to the ready state, its priority will be increased; therefore, it will go into a different queue of higher priority. The occurrence of an interrupt will suspend the process for the time the interrupt is handled after which the process resumes execution.

- The CPU time slice corresponding to any queue is a multiple of the quantum (the time slice of the highest priority queue), according to the queue level – see lecture notes.

- A process that is created and is ready to run will be inserted on its arrival time (when its state becomes “ready”) in a queue that corresponds to its priority.
- Each process needs a determined number of CPU time quanta to complete (a positive integer, 1, 2, 3, ...).
- Some processes execute I/O operations. When such an operation is started, the process is pre-empted and switched to the blocked state. The next ready process with highest priority takes control of the CPU. The event of I/O completion triggers the switch of the process from the blocked state to the ready state – the process is inserted into a queue of a higher priority than the one the process had when blocked.
- If the process completes its execution during a time slice, it will be terminated (exits the system), and the next process of the highest priority takes control of the CPU.
- When the system load (measured in terms of number of processes) decreases below a certain value (e.g., two processes on the lowest priority queue), the {frequency, voltage} configuration is switched from the standard one to a lower one.
- If there is no other process ready, the idle process will take control of the CPU.

The idle process waits in the lowest priority queue and if there is no process of higher priority ready, it will be given the control of the CPU. Its execution will correspond to the energy saving policy.

The kernel power manager strategy for the CPU

The kernel power manager works in two stages. During the first stage which corresponds to the CPU running processes, the configuration, {voltage, frequency}, is adapted to the load. If the load is high, the standard configuration in terms of voltage and frequency values is used. If the load is decreasing below a threshold, the configuration is changed to a lower one (lower frequency and voltage).
When there are no processes in the queues, the CPU is running the idle process that will switch the system to a sleep state. If the idle state extends in time, the CPU will switch into a deeper sleep state. The process continues down to the deepest sleep state.
When there are new processes arriving in the CPU queues, the CPU is switched back to the running state.

Lab Work

The lab work will be carried out over three weeks. It will consist of applying the lectures content and making decisions regarding process management and execution. The starting point is to define the process, its context and the set of states and events that trigger state switches (a table is recommended – first column, current state, second column, event, third column, new state, fourth column, comment). Then, design and write the scheduler algorithm in pseudo-code and explain how it works. Finally, wrap-up the report and submit it.
The report should comprise of an introduction about the general concept of a process, sections corresponding to each of the tasks discussed below and conclusions.
The report should be submitted as a pdf file on Canvas by the deadline.

Tasks to do

Task 1 - at the end of the first week, you should have completed the work associated with the process definition.

- Decide the kind of system the process will be running on, general purpose or embedded.
- Define the process context.
- Choose the range of integers that will be allocated as IDs to processes and write in pseudo-code (or code, if you want) the process ID allocation function. What happens if all integers are in use?
- Consider and present a data structure used by the kernel to store context info for all user processes in the system.

Task 2 – at the end of the second week, you should have the set of states and the set of events that make the life cycle of a process.

- Create a table with all states and the events associated with these states. Add a column where you explain the role of each state and event.
- Discuss the mechanism by which an event triggers the change of the state of a process. Consider as examples the switch to the Blocked state, and from Running to Ready. If in the life cycle you defined there isn't the Blocked state, consider another example.
- Define the rule by which the priority of a blocked process is increased when I/O is completed and the process becomes ready again.
- Describe the work (algorithm) carried out by the idle process.

Task 3 – at the end of the third week, all work should be completed and presented in a report. During that week, design the scheduler and write it in pseudo-code. Explain how it works and what data structures it is using – e.g., ready queue, or blocked queue.

- Discuss the role of the scheduler in your system and its principles (e.g., using priority)
- Present the data structures the scheduler is using.
- Write the scheduler in pseudo-code and explain how it works.

Although not required, if you program any function, insert comments in the code and include screenshots of the execution in your report.

In addition to the three tasks, you can include in your report a section of conclusions, where you can explain what you learned, challenges and the general experience of this lab.

Submission

Return your report of Tasks 1 - 3 above, including the pseudo-code in a report (pdf file) on Canvas, by the deadline – 20/02.

Tasks 1 and 2 have 3 marks allocated each; task 3 is worth 4 marks. The total for this lab is 10 marks.