Senior Project Proposal

Martin Nestorov

2019-01-10

Idea

Process scheduling - comparison and contrast

Description

There are several process scheduling algorithms that are used in batch, interactive, and real-time systems. These include, but are not limited to, First Come First Serve (**FCFS**), Shortest Job First (**SJF**), Priority Scheduling, Round-Robbin Scheduling, Guaranteed Scheduling, Lottery Scheduling, and Multilevel Queue Scheduling. All of them have their advantages and weaknesses. Some are simpler and work for small sequential systems, while others are more complex, but distribute the workload better.

The purpose of this project is to analyze and compare these different algorithms, to show their strengths and weaknesses.

Another thing to consider is the type of the system that lies under the processes. In general, we can either consider a *real-time* system, or an *interactive* one. *Real-time* systems are such that take into consideration time as an essential role. Typically, one or more devices can stimulate the system and it has to react accordingly in a certain amount of time. *Interactive* systems, much like the 'real-time' ones, can and are stimulated by other programs, but don't have such a strict time constraint.

Motivation

The motivation of this topic is due to the overwhelming interest in scheduling algorithms. It's a deep and complex topic that can be explored and serve as a great academic exercise.

Although there are a lot of algorithms to cover and time is scarce, at least the more complex and interesting ones can be tackled and implemented. There might be even some solutions to *synchronization* and *inversion* problems!

Implementation

The envisioned application would look at and compare the different process/task scheduling algorithms. It will demonstrate graphically, in real time, how these algorithms go against each other. For this purpose, different types of processes will be emulated, on some abstract level (in the form of objects, labeled as *light*, *medium*, *heavy* processes), that will pass through the different algorithm implementations. The processes themselves won't be doing any "real work", but will take some time to be considered as "completed". The algorithms themselves will evaluate these "empty" processes and schedule them appropriately. The underlying system that would host said algorithms and processes will be either a (most probably) *interactive* and *batch* system, or a *real-time* one.

Languages and frameworks

The application can be written in either `Python` or `C++` with the `curses` library (used for graphics in the terminal).