Lecture 6: CPU Scheduling Continued

Adam Hawley

January 31, 2019

Contents

1 More algorithms...

1.1 Shortest-Remaining-Time-First

Preemptive version of SJF.

1.2 Earliest Deadline First (EDF)

Each process must declare a deadline, the algorithm then runs the most urgent process first. Implemented as a variable priority scheduling algorithm.

2 Multilevel Queue

In most operating systems, more than one algorithm is used at one time. To enable this, **multilevel queues** are used. Often they are separated into **foreground**(interactive) and **background** (batch). For example:

- Foreground: RR
- Background: FCFS

This example means that the user will see the snappiest responses to the foreground services.

Scheduling has to be done between the queues. Fixed priority scheduling could be used (i.e serve all from the foreground then from the background). Or *time slice*, where each queue gets a certain amount of CPU time, e.g. 80% to forground and 20% to background.

3 Multilevel Feedback Queue

A more complicated version of the multilevel queue. In a **multilevel feedback queue** a process can move between the various queues. This can be used to control the *aging* of processes. Multilevel feedback queue schedulers are defined by the following parameters:

- Number of queues
- Scheduling algorithms for each queue
- Method used to determine when to upgrade a process
- Method used to determine when to demote a process
- Method used to determine which queue a process will enter when that process needs service

4 Thread Scheduling

4.1 Kernel Threads

OS knows about all processes and threads, so makes scheduling decision across all processes and threads.

4.2 User Threads

OS decides which process to run. OS doesn't know whether a process contains threads (if that process contains a user threads implementatin, decisions regarding scheduling of the user threads are taken there). Essentially hierarchical scheduling: when a process is run, the thread scheduler within the process (managing the user threads) willl decide which thread to run.

5 Multiprocessor Scheduling

CPU scheduling is more complex when there are multiple processors (cores) available).

5.1 Homogenous processors

- **Asymmetric multiprocessing**: only one processor accesses the system data structures, alleviating the need for data sharing.
- **Symmetric multiprocessing (SMP)**: each processor is self-scheduling, all processes in common ready queue, or each has its own private queue of ready processes.

Given a set of runnable processes, now it has to be decided which process to dispatch and which CPU should run it. Some processes may have a **processor affinity** meaning they have an affinity for the processor on which it will run.

- Soft affinity $\implies$ preference
- Hard affinity $\implies$ requirement
- Variations including processor sets (e.g with affinity masks where a process will use a bitmask to show which sets of processes it will run on).

5.2 Non-Uniform Memory Access (NUMA)

NUMA is one of the reasons for implementing processor affinity. Sometimes certain processors will have faster access to certain memory blocks than other processors. Naturally if a process uses data within one of the processor-specific blocks, it would have a greate affinity for the respective processor.

5.3 Load Balancing in Multiprocessors

If SMP, need to keep all CPUs loaded for efficiency.

- **Load balancing** attempts to keep workload evenly distributed.
- **Push migration** is a periodic task which checks the load on each processor and if it finds the processor is overloaded, it pushes tasks from the overloaded CPU to other CPUs.
- **Pull migration** is when idle processors pull waiting tasks from busy processors.