LECTURE 1.4
CPU SCHEDULING ALGORITHMS II

COP4600

Dr. Matthew Gerber

2/3/2016

Multilevel Queue (6.3.5)

- Partition the ready queue into multiple separate queues
- *Permanently* assign each process to a given queue based on *either* input *or* an inherent property of the process
- Queues are given a priority order
- Not terribly useful in non-preemptive systems

Multilevel Queue (6.3.5)

- Queue 0 – System Processes
  - OS processes, device drivers
- Queue 1 – Quasi-Real-Time User Processes
  - Media players, games
- Queue 2 – Interactive User Processes
  - Traditional applications
- Queue 3 – Non-Interactive User Processes
  - Compilers, media encoders
- Queue 4 – True Background Processes
  - Virus scanners, search indexers

Multilevel Queue (6.3.5)

- WE COULD…
- Schedule so that higher queues have absolute priority over lower queues
- Advantages:
  - Higher-importance processes less likely to be stalled by lower-importance processes
- Disadvantages:
  - Starvation, as with anything involving absolute priority

Digression: Absolute Priority

- A digression with regard to absolute priority
- Observation:
  - Absolute priority causes starvation
  - Absolute priority is always a bad idea
  - As with everything else, absolute anything is generally a bad idea
- Further observation:
  - **Absolute statements are always wrong**
  - This explicitly includes the above statement about absolute statements
- Modified observation:
  - Absolute priority is *usually* a bad idea
- This will not be the last time in the course we deal with this sort of issue

Multilevel Queue (6.3.5)

- WE COULD ALSO…
- Schedule so that higher queues have *relative* priority over lower queues
  - Give each queue its own set of time slices to slice
- Advantages:
  - Avoids starvation
- Disadvantages:
  - Higher-priority processes more likely to be disrupted by lower-priority processes

Multilevel Queue (6.3.5)

- WE COULD ALSO…
- Use a combination of absolute and relative approaches
- Schedule so that the system queue and quasi-real-time queue have absolute priority over other processes, and priority within those two tiers of queues is relative
- We will avoid the temptation to refer to this as Multimultilevel Queue scheduling

Multilevel Queue (6.3.5)

- Scheduling *within* the queues is its own issue
- Simplest version: Round-robin within each queue
- Might use other schedulers if we know the characteristics of the processes in a given queue

Multilevel Feedback Queue (6.3.6)

- Just like Multilevel Queue, except processes can move between queues
- Defined by:
  - The number of queues
  - The scheduling algorithm for each queue
  - The algorithm used to decide when to move processes between queues
  - The algorithm used to decide which queue a process starts in
- This provides a *fairly* general definition for a scheduler

Real-Time Scheduling (6.6)

- In general, real-time scheduling is used for processes that need to respond to events as quickly as possible
- Obviously, non-preemptive multitasking is a non-starter
- For soft real-time, it is enough to ensure that real-time processes run with absolute priority over other processes
- For hard real-time, we need to ensure that we will either service a process’s latency requirements or refuse the process
- The rest of this discussion will cover hard real-time scheduling algorithms

Real-Time Scheduling (6.6)

- Each process in a hard real-time scheduler has three characteristics, at least some of which will be required by the scheduler
  - Its *period* or *rate* – how often it needs to run
  - Its *deadline* – how quickly its burst must be **completed** when it becomes ready
  - Its *processing time* – how long it takes to run each period
- *Admission control* algorithms determine whether it is possible to accept a given process given these characteristics

Rate-Monotonic (6.6.3)

- Priority-based, strictly preemptive
- Every process must, when it is initially scheduled, declare its period and its deadline
- Each process is assigned a priority based on its *period*
- Processes that run more often have a better priority
- The deadline is assumed to be the beginning of the next period

Rate-Monotonic (6.6.3)

Process	Period	Burst
P1	50	20
P2	100	35

Processes Ready/Deadlines

P1,2 (50, 100)   P1 (100)

P1	**P2**	**P1**	**P2**	**Idle**

0 10 20 30 40 50 60 70 80 90 100

Rate-Monotonic (6.6.3)

Process	Period	Burst
P1	50	25
P2	80	35

Processes Ready (Deadlines)

P1,2 (50, 80) P1 (100) P2 (160)

P1	P2	P1	P2	

0 10 20 30 40 50 60 70 80 90 100

Rate-Monotonic (6.6.3)

Process	Period	Burst
P1	50	25
P2	80	35

Processes Ready (Deadlines)

P1,2 (50, 80) P1 (100) P2 (160)

P1	P2	P1	P2	

0 10 20 30 40 50 60 70 80 90 100

FAILURE

(P2’s first cycle fails to complete by tick 80)

Rate-Monotonic (6.6.3)

- Rate-monotonic scheduling “throws away” about 30% of CPU cycles
- Rate-monotonic scheduling is still proven to be the best we can do for static-priority processes given its assumptions
- So let’s try not having static priority

Earliest Deadline First (6.6.4)

- Priority-based, strictly preemptive
- Each process must declare its deadline *every time it becomes ready*
- Each process is assigned a priority based on its *deadline*
- The process with the earliest deadline is immediately assigned to the processor

Earliest Deadline First (6.6.4)

Process	Period	Burst
P1	50	25
P2	80	35

Processes Ready (Deadlines)

P1,P2 (50, 80) P1 (100) P2 (160) P1 (150)

P1	P2	P1	P2	P1	P2	i

0 10 20 30 40 50 60 70 80 90 100 110 120 130 140

Earliest Deadline First (6.6.4)

Process	Period	Burst
P1	50	25
P2	80	35

Processes Ready (Deadlines)

P1,P2 (50, 80) P1 (100) P2 (160) P1 (150)

P1	P2	P1	P2	P1	P2	i

0 10 20 30 40 50 60 70 80 90 100 110 120 130 140

Success

P1 and P2 cycle with both completing their bursts by deadline

Earliest Deadline First (6.6.4)

- Disadvantages versus Rate-Monotonic
  - Each process must declare its deadline each period
  - Less predictable – we may not know we're going to fail until a process becomes ready
- Advantages versus Rate-Monotonic
  - The processes do not need to be periodic
  - The processes can take different amounts of time each period
  - Can achieve higher processor utilization

NEXT TIME:
CASE STUDIES AND INTRODUCTION TO IPC