OPERATING SYSTEMS

Scheduling Algorithms

Suresh Jamadagni
Department of Computer Science

OPERATING SYSTEMS

Slides Credits for all the PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

OPERATING SYSTEMS

Multi-level Queue & Feedback Queue Scheduling

Suresh Jamadagni

Department of Computer Science

OPERATING SYSTEMS

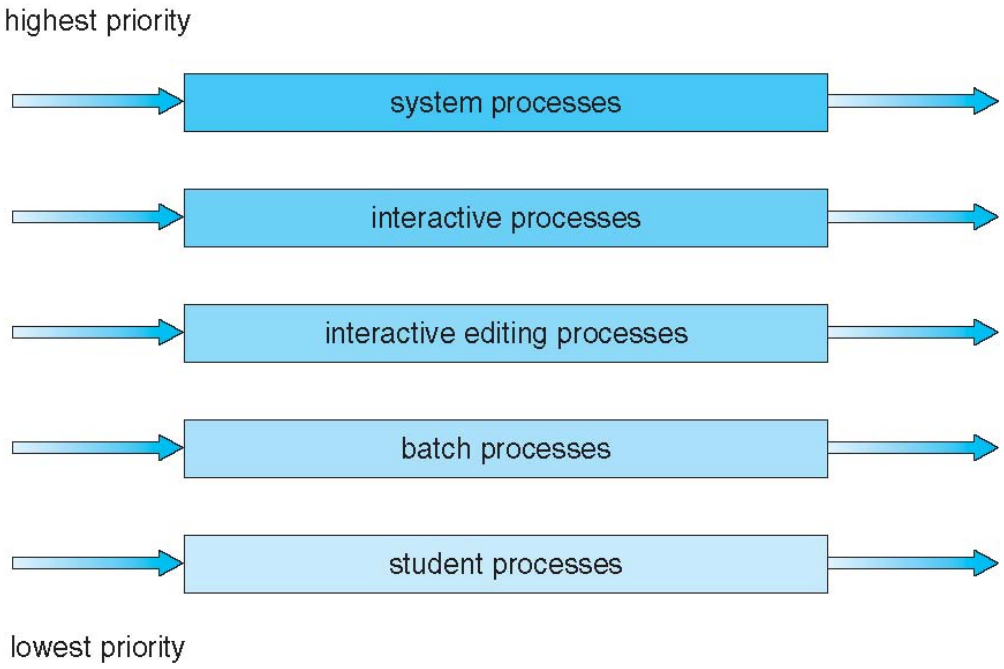
Multilevel Queue

- Ready queue is partitioned into separate queues:
  - **foreground** (interactive)
  - **background** (batch)
- Process permanently assigned to a queue
- Each queue has its own scheduling algorithm:
  - foreground – RR
  - background – FCFS
- In addition, scheduling must be done between the queues
  - Fixed priority scheduling; (serve all from foreground then from background). Possibility of starvation.
  - Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e., 80% to foreground in RR, 20% to background in FCFS

OPERATING SYSTEMS

Multilevel Queue Scheduling

- Each queue has absolute priority over lower-priority queues
- No process in the batch queue, for example, could run unless the queues for system processes, interactive processes, and interactive editing processes were all empty
- If an interactive editing process entered the ready queue while a batch process was running, the batch process would be preempted.



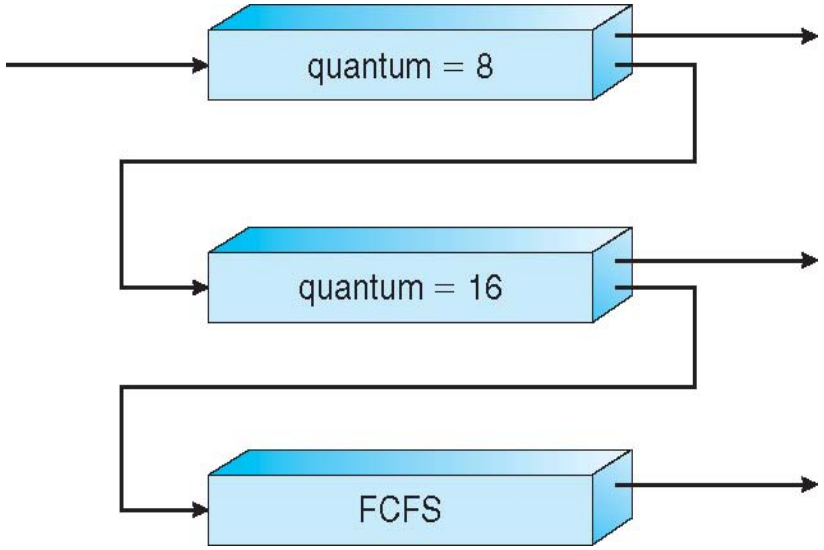
Multilevel Feedback Queue

- A process can move between the various queues; aging can be implemented this way
- Multilevel-feedback-queue scheduler 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

OPERATING SYSTEMS

Example of Multilevel Feedback Queue

- Three queues:
  - Q_0 – RR with time quantum 8 milliseconds
  - Q_1 – RR time quantum 16 milliseconds
  - Q_2 – FCFS



- Scheduling
  - A new job enters queue Q_0 which is served FCFS
    - When it gains CPU, job receives 8 milliseconds
    - If it does not finish in 8 milliseconds, job is moved to queue Q_1
  - At Q_1 job is again served FCFS and receives 16 additional milliseconds
    - If it still does not complete, it is preempted and moved to queue Q_2

OPERATING SYSTEMS

Multilevel Feedback Queue

- Three queues:
  - Q_0 – RR with time quantum 8 milliseconds
  - Q_1 – RR time quantum 16 milliseconds
  - Q_2 – FCFS

  - The scheduler first executes all processes in queue 0.
  - Only when queue 0 is empty will it execute processes in queue 1.
  - Processes in queue 2 will be executed only if queues 0 and 1 are empty.
  - A process that arrives for queue 1 will preempt a process in queue 2.
  - A process in queue 1 will in turn be preempted by a process arriving in queue 0.

OPERATING SYSTEMS

Multiple-processor Scheduling

If multiple CPUs are available, **load sharing** becomes possible, but scheduling problems become correspondingly more complex.

1. **Asymmetric multiprocessing**
   - CPU scheduling in a multiprocessor system has all scheduling decisions, I/O processing, and other system activities handled by a single processor—the master server.
   - The other processors execute only user code.
   - Asymmetric multiprocessing is simple because only one processor accesses the system data structures, reducing the need for data sharing.
2. **Symmetric multiprocessing (SMP)**
   - Each processor is self-scheduling. All processes may be in a common ready queue, or each processor may have its own private queue of ready processes.
   - All modern operating systems support SMP

OPERATING SYSTEMS

Processor Affinity

- When a process has been running on a specific processor, the data most recently accessed by the process populate the cache of the processor.
- Successive memory accesses by the process are often satisfied in cache memory.
- If the process migrates to another processor, the contents of cache memory must be invalidated for the first processor and the cache for the second processor must be repopulated.
- Because of the high cost of invalidating and repopulating caches, most SMP systems try to avoid migration of processes from one processor to another and instead attempt to keep a process running on the same processor. This is known as **processor affinity**
- **Soft affinity** - OS will attempt to keep a process on a single processor, but it is possible for a process to migrate between processors
- **Hard affinity** – OS provides system calls for process to specify a subset of processors on which it may run

OPERATING SYSTEMS

Load Balancing

- On SMP systems, it is important to keep the workload balanced among all processors to fully utilize the benefits of having more than one processor.
- **Load balancing** attempts to keep the workload evenly distributed across all processors in an SMP system.
- Load balancing is typically necessary only on systems where each processor has its own private queue of eligible processes to execute

1. **Push migration** - a specific task periodically checks the load on each processor and if it finds an imbalance, evenly distributes the load by moving (or pushing) processes from overloaded to idle or less-busy processors
2. **Pull migration** occurs when an idle processor pulls a waiting task from a busy processor.

- Push and pull migration need not be mutually exclusive

THANK YOU

Suresh Jamadagni
Department of Computer Science Engineering

sureshjamadagni@pes.edu