Operating Systems

Process Scheduling

- Scheduler manages the running processes in the system
- Scheduler is an integral part of the operating system
- Scheduler runs a scheduling algorithm to service running processes
- Different scheduling algorithms are better for particular application types
  - Real time
  - Batch
  - User-Interactive

Scheduling Algorithms

- Scheduler is managing the running processes
- There are various criteria/goals by which a scheduling algorithm can be evaluated:
  1. FAIRNESS: Make sure each process gets its fair share of the CPU
  2. EFFICIENCY: Keep the CPU busy 100% of the time
  3. RESPONSE TIME: Minimize response time for interactive users
  4. TURNAROUND: Minimize the time batch users must wait for output
  5. THROUGHPUT: Maximize the number of jobs processed in a given timeframe

Managing the Process Run Time

- Scheduler uses clock hardware to ensure that processes do not run for too long
- Scheduler uses the interrupt property of the clock to gain control of the CPU and implement a scheduling policy
- Scheduler programs the clock to interrupt at a predetermined frequency
Scheduling Algorithms

- Preemptive Scheduling
  - Scheduler suspends a running process
  - Allows other processes to run without each process having to complete
  - Must be careful with real-time
  - Must be careful to avoid race conditions
  - Multiuser friendly

- Nonpreemptive Scheduling
  - Run each process to completion
  - Not efficient for I/O bound applications
  - Easy to understand and implement
  - No race condition issues
  - Must prevent starvation
  - No way to guarantee turnaround

Dispatcher

- Dispatcher is part of the scheduler responsible for managing the actual task switch
- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program
- Dispatch latency – time it takes for the dispatcher to stop one process and start another running.

Optimization Criteria

- Each scheduling algorithm must be evaluated from how it optimizes the following variables:
  - Max CPU utilization
  - Max throughput
  - Min turnaround time
  - Min waiting time
  - Min response time

Round Robin Scheduling

- **Quantum**: A time interval that a process is allowed to run
  - A process may not use its entire quantum if it needs to block for I/O
- With **Round Robin** scheduling each process is given a quantum of time
- When quantum expires or process blocks the scheduler picks the next process to run
- Round robin scheduling requires a queue
  - **Dequeue** to get next process to run
  - After process has run for its quantum, **Enqueue** it.

**ISSUE**: How long should the quantum be?
Priority Scheduling

- Used when all processes are **not** equally important.
- Priority Scheduling
  - Each process is assigned a priority.
  - The process with the highest priority is chosen to run by the scheduler.
- Scheduler must ensure fairness
  - May lower or increase priorities of some processes to ensure that:
    - Processes are not getting too much CPU time.
    - Process are not getting too little CPU time.
- Often priorities are grouped into classes.
- Each process within a priority class must also be scheduled.
  - Can use round robin scheduling.

Multiple Queues

- Idea: Give CPU-bound processes a long quantum once in a while.
- Solution: Setup multiple queues
  - Each queue would run its processes for multiple quantum.
  - After a process was run in a particular priority queue, its priority would be increased by moving it to another queue.

Shortest Job First

- Priority and round robin scheduling work well for interactive systems.
- Other algorithms are more appropriate for a batch environment
  - Batch programs often have predictable runtimes.
- Shortest Job First: When multiple batch jobs are sitting in a queue with the same priority, the scheduler runs the shortest job first.

```
Run time: 8 4 4 4  4 4 4 8
Turnaround: 8 12 16 20
(A)  4 8 12 20  (B)
```

Mean Turnaround Time

For (A): \( \frac{32 + 12 + 8 + 4}{4} = \frac{56}{4} = 14 \)

For (B): \( \frac{16 + 12 + 8 + 30}{4} = \frac{64}{4} = 16 \)

Guaranteed Scheduling

- Have Scheduler make promises to processes and live up to them.
- Example: N process, each process is given 1/N of the available CPU time.
- Good for realtime systems where turnaround time is critical.

```
Sensor 1  Sensor 2  Sensor N
Sensor N

Real Time Operating System

TIMED SIGNALS

PROCESSES

Handler 1  Handler 2  Handler N
```

Each process must be given enough CPU time to ensure that it can keep up with all of the signals that are generated by the individual sensors.
Scheduling Mechanism versus Scheduling Policy

- The scheduling mechanism is the scheduling algorithm being used by the operating system
- Sometimes it is desirable to let processes influence the amount of CPU time that they are awarded
- Accomplished by system calls to let a process change its priority or the priority of one of its children
- Must be careful when manually setting priorities because this will influence the other processes in the system
- Scheduler still runs algorithm (mechanism) but processes can assert influence (a policy) on the scheduler

Two Level Scheduling

- It is not always possible to keep all processes in main memory
- Swapping and paging are techniques that enable process images to be saved on disk
- The images can be reloaded into memory later and run where they left off
- Saving and loading a process image from disk is very slow
- Two level scheduling
  - **Scheduler 1**: Handles running processes that are in main memory. Use a standard scheduling algorithm
  - **Scheduler 2**: Periodically swap process into and out from memory to disk. Run at a lower priority
- **Goal**: Balance the amount of time that a process is in memory and on disk

Two Level Scheduling

- The higher level scheduler that moves process to and from disk must weigh many criteria:
  1. How long has it been since the process was swapped in or out?
  2. How much CPU time has the process had recently?
  3. How big is the process? (Smaller processes are easier to manage than larger processes)
  4. How high is the priority of the process?

The scheduler must evaluate and set a policy for the above criteria to ensure efficiency and fairness