Frequently asked questions from the previous class survey

- Turnstiles: Queue for threads blocked on a lock
- Serializability?
- Timestamps? Who generates this?
- Checkpoints made only if previous transactions were successful?

Topics covered in this lecture

- CPU Scheduling
- Scheduling Criteria
- Scheduling Algorithms
  - First Come First Serve (FCFS)
  - Shortest Job First (SJF)

When there are multiple things to do, how do you choose which one to do first?

- At any point in time, some tasks are running on the system’s processor
- Others are waiting their turn for a processor
- Still other tasks are blocked waiting for I/O to complete, a condition variable to be signaled, or for a lock to be released
- When there are more runnable tasks than processors?
- The processor scheduling policy determines which tasks to run first

Just do the work in the order in which it arrives?

- After all, that seems to be the only fair thing to do
- Because of this, almost all government services work this way
- When you go to your local DMV to get a driver’s license, you take a number and wait your turn
- Although fair, the DMV often feels slow
- Advertising that your OS uses the same scheduling algorithm as the DMV is probably not going to increase your sales!
Multiprogramming organizes jobs so that the CPU always has one to execute.

- A single program (generally) cannot keep CPU & I/O devices busy at all times.
- A user frequently runs multiple programs.
- When a job needs to wait, the CPU switches to another job.
- Utilizes resources effectively:
  - CPU, memory, and peripheral devices.

**Observed Property of Process execution:**

**CPU-I/O burst cycle**

- Processes alternate between CPU-I/O bursts.
- Utilizes resources effectively: CPU, memory, and peripheral devices.

**Distribution of the duration of CPU bursts**

- Large number of short CPU bursts:
  - A typical I/O bound process.
- Small number of long CPU bursts:
  - A typical CPU-bound process.

**Bursts of CPU usage alternate with periods of waiting for I/O**

- CPU-bound processes tend to have short burst cycles.
- I/O-bound processes tend to have long burst cycles.
- Waiting for I/O.

**As CPUs get faster ...**

- Processes tend to get more I/O bound.
- CPUs are improving faster than disks.
- Scheduling of I/O bound processes will continue to be important.

**When CPU is idle, OS selects one of the processes in the ready queue to execute**

- Records in the ready queue are process control blocks (PCB).
- Implemented as:
  - FIFO queue.
  - Priority queue.
  - Tree.
  - Linked list.

---

**SLIDES CREATED BY: SHRIDEEP PALICKARA**
The Process Control Block (PCB)

- When a process is not running,
  - The kernel maintains the hardware execution state of a process within the PCB
    - Program counter, stack pointer, registers, etc.
- When a process is being context-switched away from the CPU
  - The hardware state is transferred into the PCB

The Process Control Block (PCB) is a data structure with several fields

- Includes process ID, execution state, program counter, registers, priority, accounting information, etc.
- In Linux:
  - Kernel stores the list of tasks in a circular, doubly-linked list called the task list
  - Each element in the task list is a process descriptor of the type struct task_struct, which is defined in <linux/sched.h>
  - Relatively large data structure: 1.7 KB on a 32-bit machine with ~100 fields

CPU scheduling takes place under the following circumstances

- Process keeps CPU until it relinquishes it when:
  1. It terminates
  2. It switches to the waiting state

Nonpreemptive or cooperative scheduling

- Process keeps CPU until it relinquishes it when:
  1. It terminates
  2. It switches to the waiting state

Preemptive scheduling

- Pick a process and let it run for a maximum of some fixed time
- If it is still running at the end of time interval?
  - Suspend it ...
  - Pick another process to run

Preemptive scheduling: Requirements

- A clock interrupt at the end of the time interval to give control of CPU back to the scheduler
- If no hardware timer is available?
  - Nonpreemptive scheduling is the only option
Preemptive scheduling impacts ...

- Concurrency management
- Design of the OS
- Interrupt processing

Preemptive scheduling incurs some costs:

- Manage concurrency
  - Access to shared data
    - Processes A and B share data
    - Process A is updating when it is preempted to let Process B run
    - Process B tries to read data, which is now in an inconsistent state

- System call processing
  - Kernel may be changing kernel data structure (I/O queue)

- Process preempted in the middle AND
  - Kernel needs to read/modify same structure?

- Solutions: Before context switch
  - Wait for system call to complete OR
  - I/O blocking to occur

Preemptive scheduling incurs some costs:

- Interrupt processing
  - Interrupts can occur at any time
    - Cannot always be ignored by kernel
      - Consequences: Inputs lost or outputs overwritten
  - Guard code affected by interrupts from simultaneous use:
    - Disable interrupts during entry
    - Enable interrupts at exit
  - Caveat: Should not be done often, and critical section must contain few instructions

The dispatcher is invoked during every process switch

- Gives control of CPU to process selected by the scheduler
- Operations performed:
  - Switch context
  - Switch to user mode
  - Restart program at the right location
- Dispatch latency
  - Time to stop one process and start another

SCHEDULING CRITERIA
Scheduling Algorithms: Goals

- Fairness
- Response time
- Priority enforcement
- Balance
- Meeting deadlines
- Predictability

CPU Utilization

- Difference between elapsed time and idle time
- Average over a period of time
  - Meaningful only within a context

Scheduling Criteria: Choice of scheduling algorithm may favor one over another

- **CPU Utilization**: Keep CPU as busy as possible
  - 40% for lightly loaded system
  - 90% for heavily loaded system

- **Throughput**: Number of completed processes per time unit
  - Long processes: 1/hr
  - Short processes: 10/sec

Scheduling Criteria: Choice of scheduling algorithm may favor one over another [2/2]

- **Predictability**: Low variance in response times to repeated requests
- **Fairness**: Equality in the number and timeliness of resources given to each task
- **Starvation**: Lack of progress for one task, due to resources being given to a higher priority task

What are we trying to achieve?

- Objective is to maximize the average measure
- Sometimes averages are not enough
  - Desirable to optimize minimum & maximum values
    - For good service put a ceiling on maximum response time
- Minimize the variance instead of the average
  - Predictability more important
  - High variability, but faster on average, not desirable

CS370: Operating Systems [Fall 2018]
Dept. Of Computer Science, Colorado State University

SLIDES CREATED BY: SHRIDEEP PALICKARA
Decides which process in the ready queue is allocated the CPU
Could be preemptive or nonpreemptive
Optimize measure of interest
We will use Gantt charts to illustrate schedules
Bar chart with start and finish times for processes

It is important to note that
Scheduling policy is not a panacea
Without enough capacity, performance may be poor regardless of what task you run first
There is no one right answer!
Scheduling policies pose a complex set of tradeoffs between various desirable properties

Scheduling Algorithms

First-Come, First-Served Scheduling (FCFS)
Process requesting CPU first, gets it first
Managed with a FIFO queue
When process enters ready queue?
PCC is tacked to the tail of the queue
When CPU is free?
It is allocated to process at the head of the queue
Simple to write and understand
FIFO minimizes overhead: Switches between tasks only when each one completes

First-Come, First-Served Scheduling (FCFS)
Process requesting CPU first, gets it first
Managed with a FIFO queue
When process enters ready queue?
PCC is tacked to the tail of the queue
When CPU is free?
It is allocated to process at the head of the queue
Simple to write and understand
FIFO minimizes overhead: Switches between tasks only when each one completes

Average waiting times in FCFS depend on the order in which processes arrive

Disadvantages of the FCFS scheme
Once a process gets the CPU, it keeps it
Till it terminates or does I/O
Unsuitable for time-sharing systems
Average waiting time is generally not minimal
In fact, FCFS is a poor choice for average response times
Varies substantially if CPU burst times vary greatly
Disadvantages of the FCFS scheme

- Poor performance in certain situations
  - 1 CPU-bound process and many I/O-bound processes
  - Convoy effect: Smaller processes wait for the one big process to get off the CPU

Shortest Job First (SJF) scheduling algorithm

- When CPU is available it is assigned to process with smallest CPU burst
- Moving a short process before a long process?
  - Reduction in waiting time for short process
  - Increase in waiting time for long process
- Gives us minimum average waiting time for a set of processes that arrived simultaneously
- Provably Optimal

SJF is optimal ONLY when ALL the jobs are available simultaneously

- Consider 5 processes A, B, C, D and E
  - Run times: 2, 4, 1, 1, 1
  - Arrival times: 0, 0, 3, 3, 3
  - SJF will run jobs: A, B, C, D and E
    - Average wait time: \((0 + 2 + 3 + 4 + 5)/5 = 2.8\)
    - But if you run B, C, D, E and A?
      - Average wait time: \((7 + 0 + 1 + 2 + 3)/5 = 2.6\)

Shortest Job First (SJF)

Depiction of SJF in action

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>6</td>
</tr>
<tr>
<td>P2</td>
<td>8</td>
</tr>
<tr>
<td>P3</td>
<td>7</td>
</tr>
<tr>
<td>P4</td>
<td>3</td>
</tr>
</tbody>
</table>

Wait time = \((3 + 16 + 9 + 0)/4 = 7\)

SJF is optimal ONLY when ALL the jobs are available simultaneously

Visualizing the different runs of A, B, C, D and E

Average wait time: \((0 + 2 + 3 + 4 + 5)/5 = 2.8\)

Average wait time: \((7 + 0 + 1 + 2 + 3)/5 = 2.6\)
Preemptive SJF

- What counts as “shortest” is the remaining time left on the task, not its original length.
  - If you are a nanosecond away from finishing an hour-long task, stay on that task instead of preempting for a minute-long task.
- Also known as shortest-remaining-time-first (SRTF).

Preemptive SJF

- A new process arrives in the ready queue.
  - If it is shorter (i.e. shorter time remaining) than the currently executing process?
    - Preemptive SJF will preempt the current process.

Process | Arrival | Burst |
--------|---------|-------|
P1       | 0       | 8     |
P2       | 1       | 4     |
P3       | 2       | 9     |
P4       | 3       | 5     |

Wait time = \[\frac{(10-1) + (1-1) + (17-2) + (5-3)}{4}\]
\[= \frac{26}{4} = 6.5\]

Characteristics of Preemptive SJF

- Can suffer from starvation and frequent context switches.
  - If enough short tasks arrive, long tasks may never complete.
- Analogy:
  - Supermarket manager switching to SJF to reduce waiting times.

Does Preemptive SJF has any other downsides?

- Turns out, SJF is pessimal for variance in response time.
  - By doing the shortest tasks as quickly as possible, SJF necessarily does longer tasks as slowly as possible.
- Fundamental tradeoff between reducing average response time and reducing the variance in average response time.

The contents of this slide-set are based on the following references: