Questions answered in this lecture:

What are different scheduling policies, such as FCFS, SJF, STCF, RR and MLFQ?
What type of workload performs well with each scheduler?

CPU VIRTUALIZATION: SCHEDULING

CPU VIRTUALIZATION: TWO COMPONENTS

Dispatcher (Previous lecture)
- Low-level mechanism
- Performs context-switch
  - Switch from user mode to kernel mode
  - Save execution state (registers) of old process in PCB
  - Insert PCB in ready queue
  - Load state of next process from PCB to registers
  - Switch from kernel to user mode
  - Jump to instruction in new user process

- Scheduler (Today)
  - Policy to determine which process gets CPU when

ANNOUNCEMENTS

- Reading:
  - Today cover Chapters 7-9
- Project 1: Sorting and System Calls
  - Sorting: Warm-up with using C
    - Finish Part A this week
  - Competition:
    - Free textbook or t-shirt to fastest (average) sort in each discussion section
    - Handin directories not yet available
    - Goal is for everyone to learn material
    - Do not copy code from others!

REVIEW: STATE TRANSITIONS

How to transition? ("mechanism")
When to transition? ("policy")
**VOCABULARY**

**Workload**: set of job descriptions (arrival time, run_time)
- Job: View as current CPU burst of a process
- Process alternates between CPU and I/O process moves between ready and blocked queues

**Scheduler**: logic that decides which ready job to run

**Metric**: measurement of scheduling quality

---

**SCHEDULING PERFORMANCE METRICS**

- **Minimize turnaround time**
  - Do not want to wait long for job to complete
  - Completion_time = arrival_time

- **Minimize response time**
  - Schedule interactive jobs promptly so users see output quickly
  - Initial_schedule_time = arrival_time

- **Minimize waiting time**
  - Do not want to spend much time in Ready queue

- **Maximize throughput**
  - Want many jobs to complete per unit of time

- **Maximize resource utilization**
  - Keep expensive devices busy

- **Minimize overhead**
  - Reduce number of context switches

- **Maximize fairness**
  - All jobs get same amount of CPU over some time interval

---

**WORKLOAD ASSUMPTIONS**

1. Each job runs for the same amount of time
2. All jobs arrive at the same time
3. All jobs only use the CPU (no I/O)
4. Run-time of each job is known

---

**SCHEDULING BASICS**

<table>
<thead>
<tr>
<th>Workloads:</th>
<th>Schedulers:</th>
<th>Metrics:</th>
</tr>
</thead>
<tbody>
<tr>
<td>arrival_time</td>
<td>FIFO</td>
<td>turnaround_time</td>
</tr>
<tr>
<td>run_time</td>
<td>SJF</td>
<td>response_time</td>
</tr>
<tr>
<td>STCF</td>
<td>RR</td>
<td></td>
</tr>
</tbody>
</table>

---
**EXAMPLE: WORKLOAD, SCHEDULER, METRIC**

<table>
<thead>
<tr>
<th>JOB arrival_time (s)</th>
<th>run_time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A  -0</td>
<td>10</td>
</tr>
<tr>
<td>B  -0</td>
<td>10</td>
</tr>
<tr>
<td>C  -0</td>
<td>10</td>
</tr>
</tbody>
</table>

**FIFO**: First In, First Out
- Also called FCFS (first come first served)
- Run jobs in *arrival_time* order

What is our turnaround?: completion_time - arrival_time

---

**FIFO: EVENT TRACE**

<table>
<thead>
<tr>
<th>JOB arrival_time (s)</th>
<th>run_time (s)</th>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>A  -0</td>
<td>10</td>
<td>0</td>
<td>A arrives</td>
</tr>
<tr>
<td>B  -0</td>
<td>10</td>
<td>0</td>
<td>B arrives</td>
</tr>
<tr>
<td>C  -0</td>
<td>10</td>
<td>0</td>
<td>C arrives</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>run A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
<td>complete A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
<td>run B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20</td>
<td>complete B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20</td>
<td>run C</td>
</tr>
<tr>
<td></td>
<td></td>
<td>30</td>
<td>complete C</td>
</tr>
</tbody>
</table>

---

**FIFO (IDENTICAL JOBS)**

<table>
<thead>
<tr>
<th>JOB arrival_time (s)</th>
<th>run_time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A  -0</td>
<td>10</td>
</tr>
<tr>
<td>B  -0</td>
<td>10</td>
</tr>
<tr>
<td>C  -0</td>
<td>10</td>
</tr>
</tbody>
</table>

Gantt chart:
Illustrates how jobs are scheduled over time on a CPU

---

**FIFO (IDENTICAL JOBS)**

What is the average turnaround time?
Def: turnaround_time = completion_time - arrival_time

---
FIFO (IDENTICAL JOBS)

A: 10s
B: 20s
C: 30s

What is the average turnaround time?
Def: \( \text{turnaround time} = \text{completion time} - \text{arrival time} \)

\[
\frac{10 + 20 + 30}{3} = 20s
\]

WORKLOAD ASSUMPTIONS
1. Each job runs for the same amount of time
2. All jobs arrive at the same time
3. All jobs only use the CPU (no I/O)
4. The run-time of each job is known

SCHEDULING BASICS

Workloads:
- arrival_time
- run_time

Schedulers:
- FIFO
- SJF
- STCF
- RR

Metrics:
- turnaround_time
- response_time

ANY PROBLEMATIC WORKLOADS FOR FIFO?

Workload: ?
Scheduler: FIFO
Metric: turnaround is high
**EXAMPLE: BIG FIRST JOB**

<table>
<thead>
<tr>
<th>JOB</th>
<th>arrival_time (s)</th>
<th>run_time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>~0</td>
<td>60</td>
</tr>
<tr>
<td>B</td>
<td>~0</td>
<td>10</td>
</tr>
<tr>
<td>C</td>
<td>~0</td>
<td>10</td>
</tr>
</tbody>
</table>

Draw Gantt chart for this workload and policy…
What is the average turnaround time?

**Average turnaround time:** 70s

**CONVOY EFFECT**

**PASSING THE TRACTOR**

*Problem with Previous Scheduler:*
FIFO: Turnaround time can suffer when short jobs must wait for long jobs

*New scheduler:*
SJF (Shortest Job First)
Choose job with smallest run_time
**SHORTEST JOB FIRST**

<table>
<thead>
<tr>
<th>JOB</th>
<th>arrival_time (s)</th>
<th>run_time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>~0</td>
<td>60</td>
</tr>
<tr>
<td>B</td>
<td>~0</td>
<td>10</td>
</tr>
<tr>
<td>C</td>
<td>~0</td>
<td>10</td>
</tr>
</tbody>
</table>

What is the average turnaround time with SJF?

**SJF TURNAROUND TIME**

A: 80s  
B: 10s  
C: 20s  

What is the average turnaround time with SJF?

\[
\frac{80 + 10 + 20}{3} = \approx 36.7s
\]

For minimizing average turnaround time (with no preemption):  
SJF is provably optimal

Moving shorter job before longer job improves turnaround time of short job more than it harms turnaround time of long job

**SCHEDULING BASICS**

**Workloads:**  
arrival_time  
run_time

**Schedulers:**  
FIFO  
SJF  
STCF  
RR

**Metrics:**  
turnaround_time  
response_time

**WORKLOAD ASSUMPTIONS**

1. Each job runs for the same amount of time  
2. All jobs arrive at the same time  
3. All jobs only use the CPU (no I/O)  
4. The run-time of each job is known
**SHORTEST JOB FIRST**

**ARRIVAL TIME**

<table>
<thead>
<tr>
<th>JOB</th>
<th>arrival_time(s)</th>
<th>run_time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>~0</td>
<td>60</td>
</tr>
<tr>
<td>B</td>
<td>~10</td>
<td>10</td>
</tr>
<tr>
<td>C</td>
<td>~10</td>
<td>10</td>
</tr>
</tbody>
</table>

What is the average turnaround time with SJF?

**PREEMPTIVE SCHEDULING**

Prev schedulers:
- FIFO and SJF are non-preemptive
- Only schedule new job when previous job voluntarily relinquishes CPU (performs I/O or exits)

New scheduler:
- Preemptive: Potentially schedule different job at any point by taking CPU away from running job
- STCF (Shortest Time-to-Completion First)
- Always run job that will complete the quickest

**STUCK BEHIND A TRACTOR AGAIN**

<table>
<thead>
<tr>
<th>JOB</th>
<th>arrival_time(s)</th>
<th>run_time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>~0</td>
<td>60</td>
</tr>
<tr>
<td>B</td>
<td>~10</td>
<td>10</td>
</tr>
<tr>
<td>C</td>
<td>~10</td>
<td>10</td>
</tr>
</tbody>
</table>

What is the average turnaround time?

\[
(60 + (70 - 10) + (80 - 10)) / 3 = 63.3s
\]

**NON-PREEMPTIVE: SJF**

<table>
<thead>
<tr>
<th>JOB</th>
<th>arrival_time(s)</th>
<th>run_time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>~0</td>
<td>60</td>
</tr>
<tr>
<td>B</td>
<td>~10</td>
<td>10</td>
</tr>
<tr>
<td>C</td>
<td>~10</td>
<td>10</td>
</tr>
</tbody>
</table>

Average turnaround time:

\[
(60 + (70 - 10) + (80 - 10)) / 3 = 63.3s
\]
PREEMPTIVE: STCF

Average turnaround time with STCF: 36.6
Average turnaround time with SJF: 63.3

SCHEDULING BASICS

Workloads: arrival_time
run_time

Schedulers:
FIFO
SJF
STCF
RR

Metrics:
turnaround_time
response_time

RESPONSE TIME

Sometimes care about when job starts instead of when it finishes

New metric:
response_time = first_run_time - arrival_time

RESPONSE VS. TURNAROUND

B's turnaround: 20s
B's response: 10s
**ROUND-ROBIN SCHEDULER**

Prev schedulers:
- FIFO, SJF, and STCF can have poor response time

New scheduler: RR (Round Robin)
- Alternate ready processes every fixed-length time-slice

---

**FIFO VS RR**

### Graph

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>0</td>
<td>5</td>
<td>10</td>
<td>15</td>
<td>20</td>
</tr>
</tbody>
</table>

Avg Response Time?
(0+5+10)/3 = 5

Avg Response Time?
(0+1+2)/3 = 1

In what way is RR worse?
Ave. turn-around time with equal job lengths is horrible

Other reasons why RR could be better?
If don’t know run-time of each job, gives short jobs a chance to run and finish fast

---

**SCHEDULING BASICS**

**Workloads:**
- arrival_time
- run_time

**Schedulers:**
- FIFO
- SJF
- STCF
- RR

**Metrics:**
- turnaround_time
- response_time

---

**WORKLOAD ASSUMPTIONS**

1. Each job runs for the same amount of time
2. All jobs arrive at the same time
3. All jobs only use the CPU (no I/O)
4. The run-time of each job is known
Workload Assumptions:

1. Each job runs for the same amount of time
2. All jobs arrive at the same time
3. All jobs only use the CPU (no I/O)
4. The run-time of each job is known (smart server needs scheduler)

MLFQ (Multi-Level Feedback Queue)

Goal: general-purpose scheduling

Must support two job types with distinct goals
- “interactive” programs care about response time
- “batch” programs care about turnaround time

Approach: multiple levels of round-robin;
each level has higher priority than lower levels and preempts them
**Priorities**

Rule 1: If priority(A) > Priority(B), A runs
Rule 2: If priority(A) == Priority(B), A & B run in RR

Q3 → A “Multi-level”
Q2 → B How to know how to set priority?
Q1
Q0 → C → D Approach 1: nice
Approach 2: history “feedback”

**History**

- Use past behavior of process to predict future behavior
- Common technique in systems
- Processes alternate between I/O and CPU work
- Guess how CPU burst (job) will behave based on past CPU bursts (jobs) of this process

**More MLFQ Rules**

Rule 1: If priority(A) > Priority(B), A runs
Rule 2: If priority(A) == Priority(B), A & B run in RR

More rules:
- Rule 3: Processes start at top priority
- Rule 4: If job uses whole slice, demote process (longer time slices at lower priorities)

**One Long Job (Example)**

- Q3
- Q2
- Q1
- Q0

Graph showing one long job scheduled in different priority queues.
**AN INTERACTIVE PROCESS JOINS**

Interactive process never uses entire time slice, so never demoted

**PROBLEMS WITH MLFQ?**

Problems
- unforgiving + starvation
- gaming the system

**PREVENT STARVATION**

Problem: Low priority job may never get scheduled

Periodically boost priority of all jobs (or all jobs that haven’t been scheduled)

**PREVENT GAMING**

Problem: High priority job could trick scheduler and get more CPU by performing I/O right before time-slice ends

Fix: Account for job's total run time at priority level (instead of just this time slice); downgrade when exceed threshold
Lottery Scheduling

Goal: proportional (fair) share

Approach:
- give processes lottery tickets
- whoever wins runs
- higher priority => more tickets

Amazingly simple to implement

Lottery Code

```c
int counter = 0;
int winner = getrandom(0, totaltickets);
node_t *current = head;
while (current) {
    counter += current->tickets;
    if (counter > winner) break;
    current = current->next;
}
// current is the winner
```

Lottery Example

```c
int counter = 0;
int winner = getrandom(0, totaltickets);
ode_t *current = head;
while (current) {
    counter += current->tickets;
    if (counter > winner) break;
    current = current->next;
}
// current gets to run
```

Who runs if winner is:
50 350 0

Other Lottery Ideas

- Ticket Transfers
- Ticket Currencies
- Ticket Inflation
  (read more in OSTEP)
SUMMARY

Understand goals (metrics) and workload, then design scheduler around that

General purpose schedulers need to support processes with different goals

Past behavior is good predictor of future behavior

Random algorithms (lottery scheduling) can be simple to implement, and avoid corner cases.