Chapter 7: Scheduling

Process Scheduler
- Why do we even need a process scheduler?
  - In simplest form, CPU must be shared by OS
    - Application
  - In reality, [multiprogramming]
    - OS: many separate pieces (processes)
    - Many Applications
- Scheduling [Policy] addresses...
  - When to remove a process from CPU?
  - Which ready process to allocate the CPU to?

Model of Process Execution

Recall Resource Manager

Scheduler as CPU Res Mgr

Scheduler Components
Context Switch

- Processes are switched out using **Context Switching**
- **Context Switch**:
  - **Save** pertinent info for current process
    - PC, Register, Status, etc.
  - **Update** PC, Register, Status, etc.
    - with info for process selected to run
- **Switching User Process**
  - 2 Context switches (CTX)
    - Process 1 running
      - CTX
    - Dispatcher: selects next process
      - CTX
    - Process 2 running

---

Invoking the Scheduler

- Need a **mechanism** to call the scheduler
- Voluntary call
  - Process blocks itself
  - Calls the scheduler
- Involuntary call
  - External force (interrupt) blocks the process
  - Calls the scheduler

---

Contemporary Scheduling

- Involuntary CPU sharing – timer interrupts
  - **Time quantum** determined by interval timer – usually fixed size for every process using the system
  - Sometimes called the **time slice length**

---

Choosing a Process to Run

- **Mechanism** never changes
- Strategy = **policy** the dispatcher uses to select a process from the ready list
- Different policies for different requirements
Policy Considerations

- Policy can control/influence:
  - CPU utilization
  - Average time a process waits for service
  - Average amount of time to complete a job
- Could strive for any of:
  - Equitability
  - Favor very short or long jobs
  - Meet priority requirements
  - Meet deadlines

Optimal Scheduling

- Suppose the scheduler knows each process $p_i$’s service time, $\tau(p_i)$ -- or it can estimate each $\tau(p_i)$:
  - Policy can optimize on any criteria, e.g.,
    - CPU utilization
    - Waiting time
    - Deadline
- To find an optimal schedule:
  - Have a finite, fixed # of $p_i$
  - Know $\tau(p_i)$ for each $p_i$
  - Enumerate all schedules, then choose the best

However ...

- The $\tau(p_i)$ are almost certainly just estimates
- General algorithm to choose optimal schedule is $O(n^2)$
- Other processes may arrive while these processes are being serviced
- Usually, optimal schedule is only a theoretical benchmark -- scheduling policies try to approximate an optimal schedule

Model of Process Execution

Selection Strategies

- Motivation
  - To “optimize” some aspect of system behavior

Selection Strategies...

- Considerations...
  - Turnaround time
    - Average time / job
  - Throughput
    - Jobs / time unit
  - Response time
  - System availability
  - Deadlines
**Talking About Scheduling ...**

- Let $P = \{ p_i | 0 \leq i < n \}$ = set of processes
- Let $S(p_i)$ = \{running, ready, blocked\}
- Let $t(p_i)$ = Time process needs to be in running state (the service time)
- Let $W(p_i)$ = Time $p_i$ is in ready state before first transition to running (wait time)
- Let $T_{TRnd}(p_i)$ = Time from $p_i$ first enter ready to last exit ready (turnaround time)
- Batch Throughput rate = inverse of avg $T_{TRnd}$
- Timesharing response time = $W(p_i)$

**Definition & Terms**

- Time Quantum
  - Amount of time between timer interrupts
  - Also called Time Slice

- Service Time $T(P)$
  - Amount of time process needs to be in Running state (acquired CPU) before it is completed

- Wait Time $W(P)$
  - Time a process spends waiting in the Ready state before its first transition to the Running state

**Definition & Terms...**

- Turnaround Time $T(P)$
  - Amount of time between moment process first enters Ready state and the moment the process exits Running state for the last time (completed)

- Service time, Wait time & Turnaround time are measurable metrics used to compare scheduling algorithms

**Simplified Model**

- Simplified, but still provide analysis result
- Easy to analyze performance

**Classes of Scheduling Algorithms**

- 2 major classes
  - Non-preemptive
    - Run to completion
  - Preemptive
    - Process with highest priority always gets CPU

- Recall: Several ways to establish priority

**Nonpreemptive Schedulers**

- Try to use the simplified scheduling model
- Only consider running and ready states
- Ignores time in blocked state:
  - "New process created when it enters ready state"
  - "Process is destroyed when it enters blocked state"
- Really just looking at "small phases" of a process
First-Come-First-Served

<table>
<thead>
<tr>
<th>$p_i$</th>
<th>$T_{ma^d}(p_i) = T(p_i) = 350$</th>
<th>$W(p_i) = 0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>350</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>125</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>475</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>250</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>75</td>
<td></td>
</tr>
</tbody>
</table>

$T_{ma^d}(p_i) = T(p_i) = 350$

$T_{ma^d}(p_i) = (T(p_i) + T_{ma^d}(p_i)) = 125 + 350 = 475$

$T_{ma^d}(p_i) = (T(p_i) + T_{ma^d}(p_i)) = 475 + 950 = 950$

$W(p_i) = 0$

$W(p_i) = T_{ma^d}(p_i) = 350$

$W(p_i) = T_{ma^d}(p_i) = 475$

FCFS Average Wait Time

<table>
<thead>
<tr>
<th>$p_i$</th>
<th>$T_{ma^d}(p_i) = T(p_i) = 350$</th>
<th>$W(p_i) = 0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>350</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>125</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>475</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>250</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>75</td>
<td></td>
</tr>
</tbody>
</table>

$T_{ma^d}(p_i) = T(p_i) = 350$

$T_{ma^d}(p_i) = (T(p_i) + T_{ma^d}(p_i)) = 125 + 350 = 475$

$T_{ma^d}(p_i) = (T(p_i) + T_{ma^d}(p_i)) = 475 + 950 = 950$

$W(p_i) = 0$

$W(p_i) = T_{ma^d}(p_i) = 350$

$W(p_i) = T_{ma^d}(p_i) = 475$

$W_{avg} = (0 + 350 + 475 + 950 + 1200) / 5 = 3374 / 5 = 675$
Shortest Job Next

1  \( p_i \)
0  350
1  125
2  475
3  250
4  75

\( T_{\text{next}}(p_4) = p_4 = 75 \)

\( W(p_4) = 0 \)

Shortest Job Next

1  \( p_i \)
0  350
1  125
2  475
3  250
4  75
0  75  200  450

\( T_{\text{next}}(p_4) = p_4 + p_3 = 125 + 75 = 200 \)

\( W(p_4) = 75 \)

Shortest Job Next

1  \( p_i \)
0  350
1  125
2  475
3  250
4  75
0  75  200  450

\( T_{\text{next}}(p_4) = p_4 + p_3 + p_2 = 250 + 125 + 75 = 450 \)

\( W(p_4) = 200 \)

\( W(p_3) = 0 \)

\( W(p_2) = 0 \)

\( W(p_1) = 0 \)

Shortest Job Next

1  \( p_i \)
0  350
1  125
2  475
3  250
4  75
0  75  200  450  800  1275

\( T_{\text{next}}(p_4) = p_4 + p_3 + p_2 + p_1 = 350 + 250 + 125 + 75 = 800 \)

\( W(p_4) = 450 \)

\( W(p_3) = 75 \)

\( W(p_2) = 800 \)

\( W(p_1) = 1275 \)

\( \text{Minimizes wait time} \)

\( \text{May starve large jobs} \)

\( \text{Must know service times} \)

Minimizes wait time

May starve large jobs

Must know service times

\( W_{\text{avg}} = (450 + 75 + 800 + 200 + 0) / 5 = 1525 / 5 = 305 \)
### Priority Scheduling

<table>
<thead>
<tr>
<th>Process</th>
<th>Deadline</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>p0</td>
<td>350</td>
<td>5</td>
</tr>
<tr>
<td>p1</td>
<td>125</td>
<td>2</td>
</tr>
<tr>
<td>p2</td>
<td>475</td>
<td>3</td>
</tr>
<tr>
<td>p3</td>
<td>250</td>
<td>1</td>
</tr>
<tr>
<td>p4</td>
<td>75</td>
<td>4</td>
</tr>
</tbody>
</table>

\[
T_{\text{TRnd}}(p_0) = t(p_0) + t(p_4) + t(p_2) + t(p_1) + t(p_3) = 350 + 75 + 475 + 125 + 250 = 1275
\]

\[
T_{\text{TRnd}}(p_1) = t(p_1) + t(p_3) = 125 + 250 = 375
\]

\[
T_{\text{TRnd}}(p_2) = t(p_2) + t(p_1) + t(p_3) + t(p_4) = 475 + 125 + 250 = 850
\]

\[
T_{\text{TRnd}}(p_3) = t(p_3) = 250
\]

\[
T_{\text{TRnd}}(p_4) = t(p_4) + t(p_2) + t(p_1) + t(p_3) = 75 + 475 + 125 + 250 = 925
\]

\[
W(p_0) = 925
\]

\[
W(p_1) = 375
\]

\[
W(p_2) = 850
\]

\[
W(p_3) = 0
\]

\[
W(p_4) = 925
\]

\[
W_{\text{avg}} = \frac{925 + 375 + 850 + 0 + 925}{5} = \frac{2400}{5} = 480
\]

### Deadline Scheduling

<table>
<thead>
<tr>
<th>Process</th>
<th>Deadline</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>p0</td>
<td>350</td>
<td>5</td>
</tr>
<tr>
<td>p1</td>
<td>125</td>
<td>2</td>
</tr>
<tr>
<td>p2</td>
<td>475</td>
<td>3</td>
</tr>
<tr>
<td>p3</td>
<td>250</td>
<td>1</td>
</tr>
<tr>
<td>p4</td>
<td>75</td>
<td>4</td>
</tr>
</tbody>
</table>

\[
T_{\text{TRnd}}(p_0) = t(p_0) + t(p_4) + t(p_2) + t(p_1) = 350 + 75 + 475 + 125 + 250 = 1275
\]

\[
T_{\text{TRnd}}(p_1) = t(p_1) + t(p_3) = 125 + 250 = 375
\]

\[
T_{\text{TRnd}}(p_2) = (p_2) + t(p_1) + t(p_3) + t(p_4) = 475 + 125 + 250 = 850
\]

\[
T_{\text{TRnd}}(p_3) = t(p_3) = 250
\]

\[
T_{\text{TRnd}}(p_4) = t(p_4) + t(p_2) + t(p_1) + t(p_3) = 75 + 475 + 125 + 250 = 925
\]

\[
W(p_0) = 250
\]

\[
W(p_1) = 375
\]

\[
W(p_2) = 850
\]

\[
W(p_3) = 0
\]

\[
W(p_4) = 925
\]

\[
W_{\text{avg}} = \frac{250 + 375 + 850 + 0 + 925}{5} = \frac{2400}{5} = 480
\]

### Preemptive Schedulers

- Highest priority process is guaranteed to be running at all times
- Or at least at the beginning of a time slice
- Dominant form of contemporary scheduling
- But complex to build & analyze

### Round Robin (TQ=50)

<table>
<thead>
<tr>
<th>Process</th>
<th>Deadline</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>p0</td>
<td>350</td>
<td>5</td>
</tr>
<tr>
<td>p1</td>
<td>125</td>
<td>2</td>
</tr>
<tr>
<td>p2</td>
<td>475</td>
<td>3</td>
</tr>
<tr>
<td>p3</td>
<td>250</td>
<td>1</td>
</tr>
<tr>
<td>p4</td>
<td>75</td>
<td>4</td>
</tr>
</tbody>
</table>

\[
W(p_0) = 0
\]

\[
W(p_1) = 50
\]

\[
W(p_2) = 100
\]

\[
W(p_3) = 0
\]

\[
W(p_4) = 100
\]
Round Robin (TQ=50)

1. \( \pi(p_1) = 0 \)  
2. \( \pi(p_2) = 350 \)  
3. \( \pi(p_3) = 125 \)  
4. \( \pi(p_4) = 475 \)

\( W(p_0) = 0 \)  
\( W(p_1) = 50 \)  
\( W(p_2) = 100 \)  
\( W(p_3) = 150 \)  
\( W(p_4) = 200 \)

\( T_{\text{trav}}(p_i) \) values:
- \( T_{\text{trav}}(p_0) = 550 \)
- \( T_{\text{trav}}(p_1) = 475 \)

Round Robin (TQ=50)

1. \( \pi(p_1) = 0 \)  
2. \( \pi(p_2) = 350 \)  
3. \( \pi(p_3) = 125 \)  
4. \( \pi(p_4) = 475 \)

\( W(p_0) = 0 \)  
\( W(p_1) = 50 \)  
\( W(p_2) = 100 \)  
\( W(p_3) = 150 \)  
\( W(p_4) = 200 \)

\( T_{\text{trav}}(p_i) \) values:
- \( T_{\text{trav}}(p_0) = 550 \)
- \( T_{\text{trav}}(p_1) = 950 \)
- \( T_{\text{trav}}(p_2) = 475 \)
- \( T_{\text{trav}}(p_3) = 750 \)  
- \( T_{\text{trav}}(p_4) = 650 \)
**Round Robin (TQ=50)**

<table>
<thead>
<tr>
<th></th>
<th>pi</th>
<th>0</th>
<th>350</th>
<th>1</th>
<th>125</th>
<th>2</th>
<th>475</th>
<th>3</th>
<th>250</th>
<th>4</th>
<th>75</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T_{RR}(p_i)</td>
<td>1100</td>
<td>550</td>
<td>950</td>
<td>475</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>W(p_i)</td>
<td>0</td>
<td>50</td>
<td>150</td>
<td>200</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Overhead must be considered
- Equitable
- Most widely-used
- Fits naturally with interval timer

RR with Overhead=10 (TQ=50)

<table>
<thead>
<tr>
<th></th>
<th>pi</th>
<th>0</th>
<th>350</th>
<th>1</th>
<th>125</th>
<th>2</th>
<th>475</th>
<th>3</th>
<th>250</th>
<th>4</th>
<th>75</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T_{RR}(p_i)</td>
<td>1320</td>
<td>660</td>
<td>1275</td>
<td>565</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>W(p_i)</td>
<td>0</td>
<td>60</td>
<td>100</td>
<td>120</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Overhead must be considered

Contemporary Scheduling

- Involuntary CPU sharing -- timer interrupts
  - *Time quantum* determined by interval timer -- usually fixed for every process using the system
  - Sometimes called the *time slice length*
- *Priority* based process (job) selection
  - Select the highest priority process
  - Priority reflects policy
- With *preemption*
- Usually a variant of *Multi-Level Queues*
**BSD 4.4 Scheduling**
- Involuntary CPU Sharing
- Preemptive algorithms
- 32 Multi-Level Queues
  - Queues 0-7 are reserved for system functions
  - Queues 8-31 are for user space functions
  - `nice` influences (but does not dictate) queue level

**UNIX Scheduler**
![The UNIX scheduler is based on a multilevel queue structure](taken_from_modern_operating_systems_2nd_ed_tanenbaum_2001.png)

**Process Life Cycle**
![Dark square contains fixed, maximum number of processes](processed_life_cycle.png)

**Job and Process Scheduler**
- **Job Scheduler**
  - Controls when jobs will be allowed to contend the CPU
  - Most popular techniques
    - **FIFO** First in, first out
    - **SJF** Shortest job first

- **Process Scheduler**
  - Controls when individual jobs (processes) will actually get the CPU
  - Only interesting in multi-programming
  - Most popular technique is **Round Robin**
    - Give each process one time slice in turn until complete

**Turnaround and Weighted Turnaround Time**
Let:
- \( N \) be number of jobs
- \( A_i \) be arrival time of i-th job
- \( F_i \) be finish time of i-th job

Turnaround time for i-th job:
\[
T_i = F_i - A_i
\]

Average turnaround time for i-th job:
\[
T = \frac{\sum T_i}{N}
\]

Weighted turnaround time for i-th job:
\[
WT_i = \frac{(F_i - A_i)}{(Service-time)_i}
\]

Average Weighted Turnaround time:
\[
WT = \frac{\sum WT_i}{N}
\]

**Processor Sharing (PS)**
- **"Theoretical" Scheduling Algorithm**
  - Limit of RR as time quantum goes to zero.
  - Like giving each CPU cycle to a different process, in round robin fashion.
  - \( N \) processes scheduled by PS
    - Each job runs on dedicated \( N \)-fold slower CPU.
    - Thus, READY = RUNNING.
  - CPU Time "shared" equally among processes
Scheduling Example 2
Assume:
Multiprogramming  FIFO Job Scheduling
Processor Sharing Process Scheduling

<table>
<thead>
<tr>
<th>Job</th>
<th>Arrives</th>
<th>Run Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10.0</td>
<td>0.3</td>
</tr>
<tr>
<td>2</td>
<td>10.2</td>
<td>0.5</td>
</tr>
<tr>
<td>3</td>
<td>10.4</td>
<td>0.1</td>
</tr>
<tr>
<td>4</td>
<td>10.5</td>
<td>0.4</td>
</tr>
<tr>
<td>5</td>
<td>10.8</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Example 2 Continued

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
<th># Jobs</th>
<th>Headway</th>
<th>Time Left</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0</td>
<td>1 A</td>
<td></td>
<td>1</td>
<td>0.3</td>
</tr>
<tr>
<td>10.2</td>
<td>2 A</td>
<td>1</td>
<td>0.2</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>2 A</td>
<td>1</td>
<td>0.2</td>
<td>1.1</td>
</tr>
<tr>
<td>10.4</td>
<td>1 F</td>
<td>2</td>
<td>0.1</td>
<td>2.0</td>
</tr>
<tr>
<td></td>
<td>3 A</td>
<td>3</td>
<td>0.1</td>
<td>2.5</td>
</tr>
<tr>
<td>10.5</td>
<td>4 A</td>
<td>2</td>
<td>0.05</td>
<td>2.35</td>
</tr>
<tr>
<td></td>
<td>3 A</td>
<td>3</td>
<td>0.05</td>
<td>2.40</td>
</tr>
<tr>
<td></td>
<td>4 A</td>
<td>4</td>
<td>0.4</td>
<td>2.85</td>
</tr>
<tr>
<td>10.65</td>
<td>3 F</td>
<td>3</td>
<td>0.05</td>
<td>2.35</td>
</tr>
<tr>
<td></td>
<td>4 F</td>
<td>4</td>
<td>0.35</td>
<td>2.70</td>
</tr>
</tbody>
</table>

Example 2 Continued...

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
<th># Jobs</th>
<th>Headway</th>
<th>Time Left</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.8</td>
<td>5 A</td>
<td></td>
<td>2</td>
<td>0.075</td>
</tr>
<tr>
<td></td>
<td>4 A</td>
<td>4</td>
<td>0.075</td>
<td>2.225</td>
</tr>
<tr>
<td></td>
<td>5 A</td>
<td>5</td>
<td>0.075</td>
<td>2.325</td>
</tr>
<tr>
<td>11.1</td>
<td>5 F</td>
<td>3</td>
<td>0.1</td>
<td>2.125</td>
</tr>
<tr>
<td></td>
<td>4 F</td>
<td>4</td>
<td>0.1</td>
<td>2.275</td>
</tr>
<tr>
<td>11.35</td>
<td>2 F</td>
<td>2</td>
<td>0.125</td>
<td>4.05</td>
</tr>
<tr>
<td>11.40</td>
<td>4 F</td>
<td>1</td>
<td>0.05</td>
<td>4.00</td>
</tr>
</tbody>
</table>

T and W for Example 2

<table>
<thead>
<tr>
<th>Job</th>
<th>Run</th>
<th>Start</th>
<th>Finish</th>
<th>Ti</th>
<th>WTi</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.3</td>
<td>10.3</td>
<td>10.4</td>
<td>0.4</td>
<td>1.33</td>
</tr>
<tr>
<td>2</td>
<td>0.5</td>
<td>10.2</td>
<td>11.35</td>
<td>1.15</td>
<td>2.3</td>
</tr>
<tr>
<td>3</td>
<td>0.1</td>
<td>10.4</td>
<td>10.65</td>
<td>0.25</td>
<td>2.5</td>
</tr>
<tr>
<td>4</td>
<td>0.4</td>
<td>10.5</td>
<td>11.4</td>
<td>0.9</td>
<td>2.25</td>
</tr>
<tr>
<td>5</td>
<td>0.1</td>
<td>10.8</td>
<td>11.1</td>
<td>0.3</td>
<td>3.0</td>
</tr>
</tbody>
</table>

Check:
Because CPU was never idle, 1.4 + 10.0 must equal time of last event (11.4)

Definitions
Number processes in ready queue before this event:
Current event:
a=A=Arrival
s=Schedule
f=Finish
h=In Hold

Reduction in run time that each process in ready queue experiences since last event occurred:
Remaining run time of each process in ready queue:

Scheduling Example 4
Assume:
FIFO Job Scheduling  100 K Main Memory
5 Tape Drives  Processor Sharing Process Scheduling

<table>
<thead>
<tr>
<th>Job</th>
<th>Arrives</th>
<th>Run Time</th>
<th>Memory</th>
<th>Tapes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0</td>
<td>0.5</td>
<td>30</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1.2</td>
<td>1.0</td>
<td>50</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1.3</td>
<td>1.5</td>
<td>50</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1.4</td>
<td>2.0</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>1.7</td>
<td>0.5</td>
<td>30</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>2.1</td>
<td>1.0</td>
<td>30</td>
<td>2</td>
</tr>
</tbody>
</table>

Assume:
FIFO Job Scheduling  100 K Main Memory
5 Tape Drives  Processor Sharing Process Scheduling

<table>
<thead>
<tr>
<th>Job</th>
<th>Arrives</th>
<th>Run Time</th>
<th>Memory</th>
<th>Tapes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0</td>
<td>0.5</td>
<td>30</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1.2</td>
<td>1.0</td>
<td>50</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1.3</td>
<td>1.5</td>
<td>50</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1.4</td>
<td>2.0</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>1.7</td>
<td>0.5</td>
<td>30</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>2.1</td>
<td>1.0</td>
<td>30</td>
<td>2</td>
</tr>
</tbody>
</table>
### Example 4 Continued

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
<th># Jobs</th>
<th>HWay</th>
<th>MM</th>
<th>Tapes</th>
<th>Time Left</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>1 A,S</td>
<td>1</td>
<td>0.2</td>
<td>20</td>
<td>2</td>
<td>1 0.5</td>
</tr>
<tr>
<td>1.2</td>
<td>2 A,S</td>
<td>1</td>
<td>0.2</td>
<td>20</td>
<td>2</td>
<td>1 0.3</td>
</tr>
<tr>
<td>1.3</td>
<td>3 A,H</td>
<td>2</td>
<td>0.05</td>
<td>20</td>
<td>2</td>
<td>1 0.25</td>
</tr>
<tr>
<td>1.4</td>
<td>4 A,S</td>
<td>2</td>
<td>0.05</td>
<td>0</td>
<td>0</td>
<td>2 0.0</td>
</tr>
<tr>
<td>1.7</td>
<td>5 A,H</td>
<td>3</td>
<td>0.1</td>
<td>0</td>
<td>0</td>
<td>4 2.0</td>
</tr>
<tr>
<td>2.0</td>
<td>1 F</td>
<td>3</td>
<td>0.1</td>
<td>30</td>
<td>2</td>
<td>4 1.8</td>
</tr>
</tbody>
</table>

---

### Example 4 Continued ...

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
<th># Jobs</th>
<th>HWay</th>
<th>MM</th>
<th>Tapes</th>
<th>Time Left</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>6 A,S</td>
<td>2</td>
<td>0.05</td>
<td>0</td>
<td>0</td>
<td>2 0.65</td>
</tr>
<tr>
<td>3.0</td>
<td>1 F</td>
<td>3</td>
<td>0.35</td>
<td>30</td>
<td>2</td>
<td>3 1.15</td>
</tr>
<tr>
<td>4.6</td>
<td>4 F</td>
<td>2</td>
<td>0.75</td>
<td>50</td>
<td>4</td>
<td>4 3.04</td>
</tr>
<tr>
<td>5.0</td>
<td>5 F</td>
<td>2</td>
<td>0.1</td>
<td>70</td>
<td>2</td>
<td>5 0.1</td>
</tr>
<tr>
<td>7.5</td>
<td>5 F</td>
<td>1</td>
<td>0.1</td>
<td>100</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

### T and W for Example 4

<table>
<thead>
<tr>
<th>Job</th>
<th>Run</th>
<th>Arrives</th>
<th>Finish</th>
<th>Ti</th>
<th>WTi</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5</td>
<td>1.0</td>
<td>2.0</td>
<td>1.0</td>
<td>2.0</td>
</tr>
<tr>
<td>2</td>
<td>1.0</td>
<td>1.2</td>
<td>4.05</td>
<td>2.85</td>
<td>2.85</td>
</tr>
<tr>
<td>3</td>
<td>1.5</td>
<td>1.3</td>
<td>7.4</td>
<td>6.1</td>
<td>4.06</td>
</tr>
<tr>
<td>4</td>
<td>2.0</td>
<td>1.4</td>
<td>6.6</td>
<td>5.2</td>
<td>2.6</td>
</tr>
<tr>
<td>5</td>
<td>0.5</td>
<td>1.7</td>
<td>7.5</td>
<td>5.8</td>
<td>11.6</td>
</tr>
<tr>
<td>6</td>
<td>2.1</td>
<td>2.1</td>
<td>5.1</td>
<td>3.0</td>
<td>3.0</td>
</tr>
</tbody>
</table>

\[T = 3.99 \quad WT = 4.35\]