Process and Process Management

Prof. P.C.P. Bhatt
What is a Process?

- Recall from Module 1 that a process is a program in execution.
- A process in execution needs resources like processing resource, memory and IO resource.
- Imagine a program written in C – myProg.c.
- After compilation we get an executable.
- If we now give a command like ./a.out it becomes a process.
What is a Process - 2

- A computer can have several process running or active at any given time.

- In case of multiple users on a system, all users share a common processing resource.
Multi-programming and Time Sharing

- Let us consider a system with only one processor and one user running one program: \textit{prog}_1.
- \textit{IO} and \textit{processing} will happen \textit{alternately}.
- \textit{When IO} is required, say keyboard input, the \textit{processor idles}. This is because we are nearly a \textit{million times} slower than the processor !!!
One Program - One User - Uni-processor Operation

*Note: Most of the time the processor is idling !!
Now *think* about the processor utilization

- What *percentage* of *time* are we *engaging* the processor?
- Recall that *Von Neumann* computing requires a *program to reside in main memory* to run.
- Clearly, having *just one program* would result in *gross under utilization of the processor*.
- To *enhance utilization* we should try to have *more than one* ready-to-run program resident *in main-memory*.
A processor is the central element in a computer’s operation.

A computer’s throughput depends upon the extent of utilization of its processor.

Previous figure shows processor idling for very long periods of time when only one program is executed.

Now let us consider two ready-to-run memory resident programs and their execution sequence.
Multi-programming Support in OS - 1

Consider two programs: \textit{prog}_1 and \textit{prog}_2 resident in main memory.
In the figure,

- When `prog_1` is not engaging the processor may be utilized to run another ready-to-run program.

- Clearly, these two programs can be processed without significantly sacrificing the time required to process either of them.
Disadvantage - overhead faced while switching the context of use of the processor.

Advantage - computer resource utilization is improved.

Advantage – memory utilization is improved with multiple processes residing in main memory.

A system would give maximum throughput when all its components are busy all the time.
Consider the following scenario:

- The number of ready-to-run programs must be maximized to maximize throughput of processor.

- These programs could belong to different users.
A system with its resources being used by multiple users is called *time sharing system*.

For example, a system with multiple terminals. Also a *web server serving multiple clients*.

However, such a usage has *overheads*. Lets see some of the overheads.
In case of a switch in the context of the use of the processor, we must know *where in the program sequence* the program was suspended.

In addition, *intermediate results stored in registers* have to be *safely stored in a location before suspension*. 

Response Time - 3
When a large number of resident user programs compete for the processor resource, the frequency of storage, reloads and wait periods also increase.

If overheads are high, users will have to wait longer for their programs to execute =\Rightarrow \text{Response Time of the system becomes longer.}

Response Time is the time interval which spans the time from when the last character has been input to the time when the first character of the output appears.
Response Time: Some Facts

- In a time sharing system, it is important to achieve an acceptable response time.

- In a plant with an on-line system, system devices are continuously monitored: to determine the criticality of a plant condition. Come to think of it, even a library system is an on-line system.

- If an online system produces a response time within acceptable limits, we say it is a real-time system.
Process States

In all the previous examples, we said

- A process is in “RUN” state if it is engaging the processor,
- A process is in “WAIT” state if it is waiting for IO to be completed
- In our simplistic model we may think of 5 states:
  - New-process
  - Ready-to-run
  - Running
  - Waiting-for-IO and
  - Exit
Modeling Process States

NEW PROCESS

READY TO RUN

Dispatch To Process

Processor Time Out

RUNNING

WAITING FOR IO

IO

IO Completed

EXIT

Enter ready to run state
When a process is created, OS assigns it an ID - *pid*, and creates a *data structure* to record its progress. The *state* of the process is now *ready-to-run*.

OS has a *dispatcher* that selects a ready-to-run process and assigns to the processor.

OS allocates a *time slot* to run this process.

OS *monitors* the progress of every process during its life time.
A process may not progress till a certain event occurs - *synchronizing signal*.

A process waiting for IO is said to be *blocked for IO*.

OS manages all these process migrations between process states.
A Queuing Model

- Data structures are used for process management.

- OS maintains a *queue* for all *ready-to-run* processes.

- OS may have separate queue for each of the likely events (including completion of IO).
Queues Based Model - 1
This model helps in the study and analysis of chosen OS policies.

As an example, consider the *First Come First Served* policy for ready-to-run queue.

To compare this policy with one that prioritizes processes we can study:

The *average and maximum delays* experienced by the lowest priority process.
Comparison of the *response times and throughputs* in the two cases.

Processor utilization *in the two cases and so on.*

Such studies offer new insights - for instance, what level of *prioritization leads to starvation.*
Scheduling Considerations

- OS maintains process data in various queues.
- These queues advance based on scheduling policies
  - First Come First Served.
  - Shortest Job First
  - Priority Based Scheduling
  - Batch Processing.
- We also need to understand Preemptive and Non-Preemptive operations.
- Note that each policy affects the performance of the overall system.
Choosing a Scheduling Policy

- Scheduling policy depends on the *nature of operations*.
- An OS policy may be chosen to *suit situations with specific requirements*.
- Within a computer system, we need policies to *schedule access to processor, memory, disc, IO and shared resource (e.g. printers)*.
A scheduling policy is often determined by a machine’s configuration and its pattern of usage.

Scheduling is considered in the following context:

- We have only one processor in the system
- We have a multi-programming system – more than one ready-to-run program in memory.
We shall study the effect on the following quality parameters:

- Response time to users
- Turn around time
- Processor utilization
- Throughput of the system
- Fairness of allocation
- Effect on other resources.

We see that measures for response time and turn around are user centered parameters.
Process utilization and throughput are system centered considerations.

Fairness of allocation and effect on other resources affect both the system and users.

An OS performance can be tuned by choosing an appropriate scheduling policy.
Comparison of Policies

Let us consider 5 processes $P1$ through $P5$.

We shall make the following assumptions:

- The jobs have to run to completion.
- No new jobs arrive till these jobs are processed.
- Time required for each job is known appropriately.
- During the run of jobs there is no suspension for IO operation.
Comparison of Three Non-Preemptive Scheduling Policies

<table>
<thead>
<tr>
<th>PROCESS NUMBER</th>
<th>TIME TO RESPOND</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1  P2  P3  P4  P5</td>
<td>20 10 25 15 5</td>
</tr>
</tbody>
</table>

**THE PROCESSES FOR PROCESSING**

- FCFS (QUEUE) ORDER: P1, P2, P3, P4, P5
- AVERAGE TIME TO COMPLETE: 50

**GANTT CHART**

<table>
<thead>
<tr>
<th>TIME TO RESPOND</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1  P2  P3  P4  P5</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>TIME TO RESPOND</th>
</tr>
</thead>
<tbody>
<tr>
<td>P3  P1  P2  P5  P4</td>
</tr>
</tbody>
</table>

**PRIORITY QUEUE: P3, P1, P2, P5, P4**
- AVERAGE TIME TO COMPLETE: 52

**GANTT CHART**

<table>
<thead>
<tr>
<th>TIME TO RESPOND</th>
</tr>
</thead>
<tbody>
<tr>
<td>P3  P1  P2  P5  P4</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>TIME TO RESPOND</th>
</tr>
</thead>
<tbody>
<tr>
<td>P5  P2  P4  P1  P3</td>
</tr>
</tbody>
</table>

**SHORTEST JOB FIRST: P1, P2, P3, P4, P5**
- AVERAGE TIME TO COMPLETE: 35

**GANTT CHART**

<table>
<thead>
<tr>
<th>TIME TO RESPOND</th>
</tr>
</thead>
<tbody>
<tr>
<td>P5  P2  P4  P1  P3</td>
</tr>
</tbody>
</table>
Summary of Results

- Case B ➔ Average time to complete - 50
  (case of FCFS)

- Case C ➔ Average time to complete - 52
  (case of prioritized)

- Case D ➔ Average time to complete – 35
  (case of Shortest Job First)

Clearly it would seem that shortest job first is the best policy. Infact theoretically also this can be proved
Preemptive Policies - 1

The Processes for Processing

<table>
<thead>
<tr>
<th>PROCESS NUMBER</th>
<th>(A)</th>
<th>(B)</th>
<th>(C)</th>
<th>(D)</th>
<th>(E)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1 P2 P3 P4 P5</td>
<td></td>
<td>P1</td>
<td>P1</td>
<td>P5</td>
<td>P5</td>
</tr>
<tr>
<td>TIME 30 10 25 15 5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td>P2</td>
</tr>
<tr>
<td>THE GANTT CHARTS</td>
<td>10</td>
<td>P2</td>
<td>P2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>P3</td>
<td></td>
<td>P4</td>
<td>P2</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>P4</td>
<td></td>
<td>P1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>P5</td>
<td>P3</td>
<td></td>
<td>P4</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>P1</td>
<td></td>
<td>P2</td>
<td>P1</td>
</tr>
<tr>
<td></td>
<td>35</td>
<td>P2</td>
<td>P4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>P3</td>
<td></td>
<td>P1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>45</td>
<td>P4</td>
<td>P5</td>
<td>P3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>P1</td>
<td></td>
<td>P4</td>
<td>P4</td>
</tr>
<tr>
<td></td>
<td>55</td>
<td>P3</td>
<td>P1</td>
<td></td>
<td>P4</td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>P4</td>
<td>P3</td>
<td>P1</td>
<td>P1</td>
</tr>
<tr>
<td></td>
<td>65</td>
<td>P1</td>
<td>P4</td>
<td>P3</td>
<td>P3</td>
</tr>
<tr>
<td></td>
<td>70</td>
<td>P3</td>
<td>P4</td>
<td>P3</td>
<td>P3</td>
</tr>
<tr>
<td></td>
<td>75</td>
<td>P3</td>
<td>P3</td>
<td></td>
<td>P3</td>
</tr>
</tbody>
</table>

FCFS (QUEUE) ORDER : P1, P2, P3, P4, P5
TIME : 5 UNITS; AVERAGE : 55; DIFF : 50

TIME TO COMPLETE

<table>
<thead>
<tr>
<th>PROCESS NUMBER</th>
<th>(C)</th>
<th>(D)</th>
<th>(E)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1 P2 P3 P4 P5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIME 65 35 75 60 25</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

FCFS (QUEUE) ORDER : P1, P2, P3, P4, P5
TIME : 10 UNITS; AVERAGE : 53; DIFF : 55

TIME TO COMPLETE

<table>
<thead>
<tr>
<th>PROCESS NUMBER</th>
<th>(D)</th>
<th>(E)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1 P2 P3 P4 P5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIME 65 30 75 50 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SHORTEST JOB FIRST ORDER : P5, P2, P4, P1, P3
TIME : 5 UNITS; AVERAGE : 45; DIFF : 70

TIME TO COMPLETE

<table>
<thead>
<tr>
<th>PROCESS NUMBER</th>
<th>(E)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1 P2 P3 P4 P5</td>
<td></td>
</tr>
<tr>
<td>TIME 60 15 75 50 5</td>
<td></td>
</tr>
</tbody>
</table>
Preemptive Policies - 2

In the figure, we compare four cases:

- Round-robin allocation with time slice = 5 units (CASE B)
- Round-robin allocation with time slice = 10 units (CASE C)
- Shortest Job First within the Round-robin; time slice = 5 units (CASE D)
- Shortest Job First within the Round-robin; time slice = 10 units (CASE E)
Summary of Results

- Case B → Average time to complete - 52
- Case C → Average time to complete - 53
- Case D → Average time to complete - 45
- Case E → Average time to complete - 41

Clearly it would seem that shortest job first is the best policy. In fact theoretically also this can be proved.
Yet Another Variation

- It was assumed before that all jobs were present initially.
- A more realistic situation is when processes arrive at different times.
- Each job is assumed to arrive with an estimate of time required to complete.
- Considering the estimated remaining time, variation of SJF is designed.
Shortest Remaining Time Schedule

<table>
<thead>
<tr>
<th>PROCESS NUMBER</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIME</td>
<td>20</td>
<td>10</td>
<td>25</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>ARRIVAL</td>
<td>0</td>
<td>3</td>
<td>5</td>
<td>15</td>
<td>17</td>
</tr>
<tr>
<td>SERVICE START</td>
<td>0</td>
<td>5</td>
<td>50</td>
<td>15</td>
<td>20</td>
</tr>
<tr>
<td>COMPLETED</td>
<td>50</td>
<td>15</td>
<td>75</td>
<td>35</td>
<td>25</td>
</tr>
<tr>
<td>DELAY</td>
<td>50</td>
<td>12</td>
<td>70</td>
<td>20</td>
<td>8</td>
</tr>
</tbody>
</table>

Time slice: 5 units;
Average time to complete: 32 time units
How to Estimate Completion Time? - 1

<table>
<thead>
<tr>
<th></th>
<th>1.7</th>
<th>2.9</th>
<th>1.9</th>
<th>3.7</th>
<th>2.5</th>
<th>1.8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>10</th>
<th>10</th>
<th>10</th>
<th>10</th>
<th>10</th>
<th>10</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

With the assumption that we are allocating 10 units of time for each burst, we notice: for the first scenario its inadequate where as for the second one it is too large.
How to Estimate Completion Time - 2

The following strategies can be observed:

- Allocate the next larger time slice to the time actually used.
- Allocate the average over the last several time slice utilizations. It gives all previous utilizations equal weightages to find the next time slice allocation.
- Use the entire history but give lower weightages to the utilization in past (Exponential Averaging technique).
Exponential Averaging Technique - 1

We denote our current, $n$th, CPU usage burst by $t_n$. Also, we denote the average of all past usage bursts up to now by $\tau_n$. Using a weighting factor $0 \leq \alpha \leq n$ with $t_n$ and $1 - \alpha$ with $\tau_n$, we estimate the next CPU usage burst. The predicted value of $\tau_{n+1}$ is computed as:

$$\tau_{n+1} = \alpha \cdot t_n + (1 - \alpha) \cdot \tau_n$$

This formula is called an exponential averaging formula.
Exponential Averaging Technique - 2

Let us briefly examine the role of $\alpha$. If $\alpha$ is made 0 then we ignore the immediate past utilisation altogether. Obviously both would be undesirable choices. In choosing a value of $\alpha$ in the range of 0 to 1 we have an opportunity to weigh the immediate past usage, as well as, the previous history of a process with decreasing weightage. It is worth while to expand the formula further.

$$\tau_{n+1} = \alpha * t_n + (1-\alpha) * \tau_n = \alpha * t_n + \alpha * (1-\alpha) * t_{n-1} + (1-\alpha) * \tau_{n-1}$$

which on full expansion gives the following expression:

$$\tau_{n+1} = \alpha * t_n + \alpha * (1-\alpha) * t_{n-1} + \alpha * (1-\alpha)^2 * t_{n-2} + \alpha * (1-\alpha)^3 * t_{n-3}...$$
In the figure above we see the effect of the choice of $\alpha$ has in determining the weightages for past utilisations.
Process Context Switching - 1

- OS maintains a lot of information about the resources used by a running process.
- The information stored establishes the context for the process. Usually the following is stored.
  - The program computed.
  - The values in various registers.
  - The process states etc..
When a process is switched the context information needs to be changed as follows:

- **For the outgoing process**: Store the information of the current process in some area of memory.
- **For incoming process**: Copy the previously stored information from memory.

The information context that is switched is illustrated in the figure in the following slide.
Process Context Switching - 3

Execution Context of a process

The OS Process information
MAY BE A QUEUE OR ANY OTHER DATA STRUCTURE
An OS maintains, and keeps updating, a lot of information about the resources in use for a running process. For instance, each process in execution uses the program counter, registers and other resources within the CPU. So, whenever a process is switched, the OS moves out, and brings in, considerable amount of context switching information as shown in the previous figure. We see that process $P_x$ is currently executing (note that the program counter is pointing in executable code area of $P_x$).
Let us now switch the context in favor of running process $P_y$.

The following must happen:

- All the current context information about process $P_x$ must be updated in its own context area.
- All context information about process $P_y$ must be downloaded in its own context area.
- The program counter should have an address value to an instruction of process $P_y$, and process $P_y$ must be now marked as “running”.

The process context area is also called *process control block*. As an example when the process P x is switched the information stored is:

1. Program counter
2. Registers (like stack, index etc.) currently in use
3. Changed state (changed from Running to ready-to-run)
4. The base and limit register values
5. IO status (Files opened; IO blocked or completed etc.)
6. Accounting
7. Scheduling information
8. Any other relevant information.

When the process P_y is started its context must be loaded and then alone it can run.