CPU SCHEDULING

SCHEDULING CONCEPTS:

Multiprogramming A number of programs can be in memory at the same time. Allows overlap of CPU and I/O.

Jobs (batch) are programs that run without user interaction.

User (time shared) are programs that may have user interaction.

Process is the common name for both.

CPU - I/O burst cycle Characterizes process execution, which alternates, between CPU and I/O activity. CPU times are generally much shorter than I/O times. **<<< FIGURE 5.1 >>>**

Preemptive Scheduling An interrupt causes currently running process to give up the CPU and be replaced by another process.

CRITERIA FOR PERFORMANCE EVALUATION - SCHEDULING CRITERIA

Note usage of the words **DEVICE, SYSTEM, REQUEST, JOB.**

UTILIZATION The fraction of time a device is in use. (ratio of in-use time / total observation time)

THROUGHPUT The number of job completions in a period of time. (jobs / second)

SERVICE TIME The time required by a device to handle a request. (seconds)

QUEUEING TIME Time on a queue waiting for service from the device. (seconds)

RESIDENCE TIME The time spent by a request at a device.
RESIDENCE TIME = SERVICE TIME + QUEUEING TIME.

RESPONSE TIME Time used by a system to respond to a User Job. (seconds)

THINK TIME The time spent by the user of an interactive system to figure out the next request. (seconds)

The goal is to optimize both the average and the amount of variation. (but beware the ogre predictability.)

SCHEDULING ALGORITHMS:

FIRST-COME, FIRST SERVED:

- (FCFS) same as FIFO
- Simple, fair, but poor performance. Average queueing time may be long.

EXAMPLE

Process	Arrival Time	Service Time
1	0	12
2	1	4
3	2	3
4	3	6

- What are the average queueing and residence times for this scenario?
- How do average queueing and residence times depend on ordering of these processes in the queue?

SHORTEST JOB FIRST:

- Optimal for minimizing queueing time, but impossible to implement. Tries to predict the process to schedule based on previous history.
- Predicting the time the process will use on its next schedule:

$$t(n+1) = w * t(n) + (1 - w) * T(n)$$

Here: t(n+1) is time of next burst.
t(n) is time of current burst.
T(n) is average of all previous bursts .
W is a weighting factor emphasizing current or previous bursts.

<<< FIGURE 5.3 >>>

PRIORITY:

- Assign each process a priority. Schedule highest priority first. All processes within same priority are FCFS.

- Priority may be determined by user or by some default mechanism. The system may determine the priority based on memory requirements, time limits, or other resource usage.

- **Starvation** occurs if a low priority process never runs. Solution: build aging into a variable priority.

PREEMPTIVE ALGORITHMS:

- Yank the CPU away from the currently executing process when a higher priority process is ready.

- Can be applied to both Shortest Job First or to Priority scheduling.

- Avoids "hogging" of the CPU

- On time sharing machines, this type of scheme is required because the CPU must be protected from a run-away low priority process.

- Give short jobs a higher priority – perceived response time is thus better.

- What are average queueing and residence times? Compare with FCFS and SJF.

ROUND ROBIN:

- Use a timer to cause an interrupt after a predetermined time. Preempts if task exceeds it's quantum.

- Train of events
  a) Dispatch
  b) Time slice occurs OR process suspends on event
  c) Put process on some queue and dispatch next

- Use numbers in last example to find queueing and residence times. (Use quantum = 4 sec.)

- Definitions:

  Context Switch Changing the processor from running one task (or process) to another. Implies changing memory.

  Processor Sharing Use of a small quantum such that each process runs frequently at speed 1/n.

  Reschedule latency How long it takes from when a process requests to run, until it finally gets control of the CPU.

- Choosing a time quantum

  Too short - inordinate fraction of the time is spent in context switches.

  Too long - reschedule latency is too great. If many processes want the CPU, then it's a long time before a particular process can get the CPU. This then acts like FCFS.

  Adjust so most processes won't use their slice. As processors have become faster, this is less of an issue.

MULTI-LEVEL QUEUES:

- Each queue has its scheduling algorithm.
- Then some other algorithm (perhaps priority based) arbitrates between queues.
- Can use feedback to move between queues
- Method is complex but flexible.
- For example, could separate system processes, interactive, batch, favored, unfavored processes

<<< FIGURE 5.6 >>>

MULTIPLE PROCESSOR SCHEDULING:

- Different rules for homogeneous or heterogeneous processors.
- Load sharing in the distribution of work, such that all processors have an equal amount to do.
- Each processor can schedule from a common ready queue (equal machines) OR can use a master slave arrangement.