# *Process Management*

CS 502
Spring 99
WPI MetroWest/Southboro Campus

## *Processes*

- Process Concept
- Process Scheduling
- Operations on Processes
- Cooperating Processes
- Threads
- Interprocess Communication

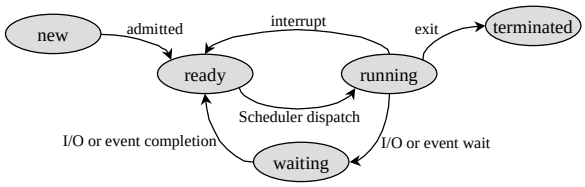
## *Process Concept*

- An operating system must execute a variety of programs:
  - Batch system -- jobs
  - Time-shared systems -- user programs or tasks
- Textbook uses the term job and process almost interchangeably
- Process -- a program in execution; process execution must progress in a sequential fashion,
- A process includes:
  - Instruction Address
  - Stack
  - Data Section

2

## *Process State*

- As a process executes, it changes *state*.
  - New: The process is being created.
  - Running: Instructions are being executed.
  - Waiting: The process is waiting for some event to occur.
  - Ready: The process is waiting to be assigned to a processor.
  - Terminated: The process has finished execution.
- Diagram of process state:




3

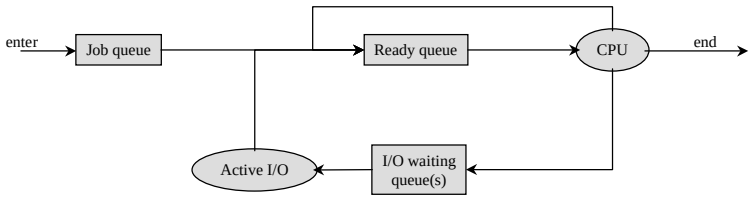
## *Process Control Block (PCB)*

- Information associated with each process
  - Process State
  - Instruction Address
  - CPU registers
  - CPU scheduling information
  - Memory-management information
  - Accounting information
  - I/O status information

4

## *Process Scheduling Queues*

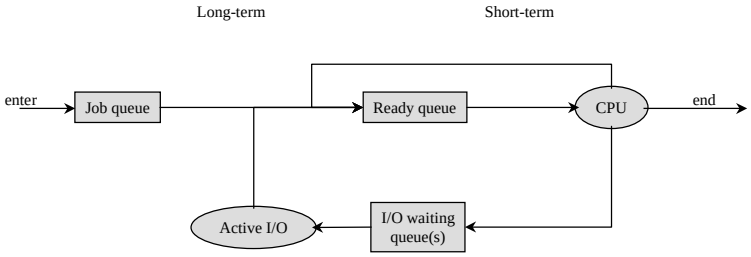
- Job queue -- set of all processes in the system.
- Ready queue -- set of all processes residing in main memory, ready and waiting to execute.
- Device queues -- set of processes waiting for an I/O device. Typically one per device or device controller.
- As processes execute, they migrate between the various queues:




5

## *Schedulers*

- Long-term scheduler (or job scheduler) -- selects which processes should be brought into the read queue.
- Short-term scheduler (or CPU scheduler) selects which process should be executed next and allocates the CPU.




6

## *Schedulers (Cont.)*

- Short-term scheduler is invoked very frequently (milliseconds) => *must be fast*
- Long-term scheduler is invoked very infrequently (seconds, minutes) => may be slow
- The long-term scheduler controls the degree of multiprogramming
- Processes can be described as either:
  - I/O-bound process -- spends more time doing I/O than computations; many short CPU bursts.
  - CPU-bound process -- spends more time doing computations; a few very long CPU bursts.

7

## *Context Switch*

- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process.
- Context-switch time is overhead; the system does no useful work while switching.
- Time dependent on hardware support.

8

## *Process Creation*

- Parent process creates children processes, which in turn create other processes, forming a tree of processes.
- Resource sharing possibilities
  - Parent and child share all resources
  - Children share subset of parent's resources
  - Parent and child share no resources
- Execution model possibilities
  - Parent and children execute concurrently
  - Parent waits until children terminate

9

## *Process Creation (Cont.)*

- Address Space
  - Child is a duplicate of the parent
  - Child has a program loaded into it
- UNIX examples
  - fork system call creates new process
  - execve system call used after a fork to replace the process' memory space with a new program

10

## *Process Termination*

- Process executes last statement and asks the operating system to delete it (exit)
  - Ability to output data from child to parent (via wait)
  - Process' resources are deallocated by the operating system
- Parent may terminate execution of children processes (abort).
  - Child has exceeded allocated resources
  - Task assigned to the child is no longer required
  - Parent is exiting.
    - Operating system may not allow child to continue if its parent terminates.
    - This may result in cascading termination

11

## *Cooperating Processes*

- Independent process cannot affect or be affected by the execution of another process.
- Cooperating process can affect or be affected by the execution of another process.
- Advantages of process cooperation:
  - Information sharing
  - Computation speed-up
  - Modularity
  - Convenience
- Mechanism for cooperation
  - File system
  - Shared Memory
  - Message passing
  - Other IPC ...

12

## *Producer-Consumer Problem*

- Paradigm for cooperating processes; producer produces information that is consumed by a consumer process.
  - Unbounded-buffer places no practical limit on the buffering between the producer and consumer
  - bounded-buffer assumes that there is a limit on the buffering between the producer and consumer

13

## *Bounded-Buffer -- Shared-Memory Solution*

- Shared data

```
const n = MAX_BUF;
typedef struct {…} item_t;
int in, out;
item_t * buffer[n];
```

- Producer process

```
item_t * nextp;

repeat
  …
  // produce an item in nextp
  …
  while (((in + 1) % n) == out) do no-op
  buffer[in] = nextp;
  in = (in + 1) % n;
until false;
```

14

## *Bounded–Buffer (Cont.)*

- Consumer process

```
item_t * nextc;

repeat
  while (in == out) no-op;
  nextc = buffer[out];
  out = (out + 1) % n;
  ...
  < consume buffer >
  ...
forever
```

- Solution is correct, but can only fill up n-1 buffer.

15

## *Threads*

- A thread (or lightweight process) is a basic unit of CPU utilization; it consists of:
  - instruction address
  - register set
  - stack space
- A thread shares with its peer threads its:
  - code section
  - data section
  - operating system resources
  - … collectively known as a task
- A traditional or heavyweight process is equal to a task with one thread.

16

## *Threads (Cont.)*

- In a multiple threaded task, while one server thread is blocked and waiting, another thread in the same task can run.
  - Cooperation of multiple threads in same task confers higher throughput and improved performance.
  - Applications that require sharing a common buffer (I.e., producer-consumer) benefit from thread utilization.
- Threads provide a mechanism that allows sequential processes to make blocking system calls while also achieving parallelism.
- Kernel supported threads (Mach and OS/2).
- User-level threads; supported above the kernel via a set of library calls at the user level (Project Andrew from CMU).
- Hyprid approach implements both user-level and kernel-supported threads (NT and Solaris 2).

17

## *Interprocess Communication (IPC)*

- Mechanisms for processes to communicate and to synchronize their actions.
- Message system -- processes communicate with each other without resorting to shared variables.
- IPC facility provides two operations:
  - send(message) -- message size fixed or variable
  - receive(message)
- If P and Q with to communicate, they need to:
  - establish a communication link between them
  - exchange messages via send/receive
- Implementation of the communication link
  - physical (e.g. shared memory, hardware bus)
  - logical (e.g. logical properties)

18

## *Implementation Questions*

- How are links established?
- Can a link be associated with more than two processes?
- How many links can there be between every pair of communicating processes?
- What is the capacity of a link?
- Is the size of a message that the link can accommodate fixed or variable?
- Is a link unidirectional or bidirectional?

19

## *Direct Communication*

- Processes must name each other explicitly:
  - send(P, message) -- send a message to process P
  - receive(Q, message) -- receive a message from process Q
- Properties of the communication link:
  - Links are established automatically.
  - A link is associated with exactly one pair of communicating processes.
  - Between each pair there exists exactly one link.
  - The link may be unidirectional, but is usually bidirectional.

20

## *Indirect Communication*

- Messages are directed and received from mailboxes (also referred to as ports).
  - Each mailbox has a unique id.
  - Processes con communicate only if they share a mailbox.
- Properties of communication link:
  - Link established only if processes share a common mailbox.
  - A link may be associated with many processes.
  - Each pair of processes may share several communication links.
  - Links may be unidirectional or bidirectional.
- Operations
  - Create a new mailbox
  - Send and receive messages through mailbox
  - Destroy a mailbox

21

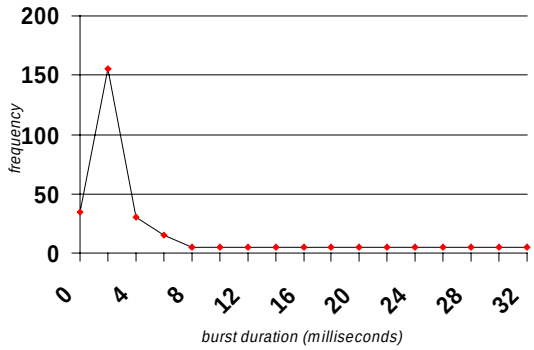
## CPU Scheduling

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Multiple-Processor Scheduling
- Real-Time Scheduling
- Algorithm Evaluation

22

## Basic Concepts

- Maximum CPU utilization obtained with multiprogramming.
- CPU-I/O Burst Cycle -- Process execution consists of a cycle of CPU execution and I/O wait.
- CPU burst duration:




23

## *CPU Scheduler*

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of therm.
- CPU Scheduling decisions may take place when a process:
  - switches from running to waiting state.
  - switches from running to read state.
  - switches from waiting to ready.
  - terminates
- Scheduling only under the first and last is *nonpreemptive*.
- All other scheduling is *preemptive*.

24

## *Dispatcher*

- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program
- *Dispatch latency* -- time it takes for the dispatcher to stop one process and start another running.

25

## *Scheduling Criteria*

- CPU utilization -- keep the CPU as busy as possible
- Throughput -- number of processes that complete their execution per time unit
- Turnaround time -- amount of time (completion - arrival) to execute a particular process
- Waiting time -- amount of time a process has been in the ready queue
- Response time -- amount of time it takes from when a request was submitted until the first response is produced, **not** output.

26

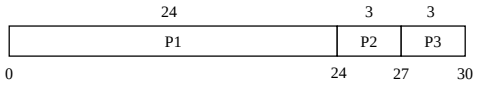
## *Optimization Criteria*

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

27

## First-Come, First-Served (FCFS) Scheduling

- Example:
  - P1 with CPU burst of 24
  - P2 with CPU burst of 3
  - P3 with CPU burst of 3
- Suppose that the processes arrive in the order P1, P2, P3
  The Gantt chart for the schedule is:

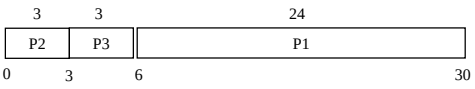



- Waiting time for P1=0; P2=24; P3=27
- Average waiting time: (0 + 24 + 27)/3 = 17

28

## FCFS Scheduling (Cont.)

- Suppose the processes arrive in the order:
  - P2; P3; P1
- The Gantt chart for the schedule is:




- Waiting time for P1 = 6; P2 = 0; P3 = 3
- Average wating time: (6 + 0 + 3)/3 = 3
- Convoy effect: short processes stack up behind the long process

29

## *Shortest-Job-First (SJF) Scheduling*

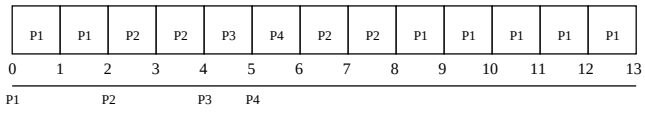
- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time.
- Two schemes:
  - Nonpreemptive -- once the CPU is given to the process, it cannot be preempted until it *completes* its CPU burst.
  - Preemptive -- if a new process arrives with CPU burst length less than the remaining time of the current executing process, then preempt. This scheme is also known as Shortest-Remaining-Time_First (SRTF).
- SJF is provably optimal with respect to the average waiting time (i.e. it always gives the minimum average waiting time for a given set of processes).

30

## *Example of Preemptive SJF (SRTF)*

| Process | Arrival | Burst |
|---|---|---|
| P1 | 0 | 7 |
| P2 | 2 | 4 |
| P3 | 4 | 1 |
| P4 | 5 | 1 |

- SJF (preemptive)




- Average waiting time = ?

31

## *Determining Length of Next CPU Burst*

- Can only estimate the length.
- Can be done by using the length of previous CPU bursts, using exponential averaging.
  - $t_n$ = actual length of $n^{th}$ CPU burst

    $\tau_{n+1}$ = predicted value for the next CPU burst

    $\alpha$, $0 \le \alpha \le 1$
  - Define:

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

32

## *Examples of Exponential Averaging*

$\alpha = 0$

$\tau_{n+1} = \tau_n$

- Recent history does not count

$\alpha = 1$

$\tau_{n+1} = t_n$

- Only the actual last CPU burst counts

- Consider a CPU burst sequence of 6, 4, 6, 4, 12, 12 and an initial guess of 10, and $\alpha = 1/2$

| **Index** | **0** | **1** | **2** | **3** | **4** | **5** | **6** |
|---|---|---|---|---|---|---|---|
| $\tau$ | 10 | | | | | | |
| ***t*** | 6 | 4 | 6 | 4 | 12 | 12 | |

33

## *Priority Scheduling*

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority
  (smallest integer ≡ highest priority)
  - preemptive
  - nonpreemptive
- SJF is priority scheduling where priority is the predicted next CPU burst time.
- Problem :: Starvation -- low priority processes may never execute.
- Solution :: Aging -- a variation of the scheme where the priority of a process increases over time.

34

## *Round Robin (RR)*

- Each process gets a small unit of CPU time (time quantum), usually 1- to 100 milliseconds. After the time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are n processes in the ready queue and the time quantum is q, then each process gets 1/n of the CPU time in chunks of at most q time units at once. No process waits more than (n-1)q time units.
- Performance
  - large q  FCFS
  - small q  q must be large with respect t context switch, otherwise overhead is too high.
- Typically, higher average turnaround time than SRTF, but better *response*.

35

## *Example: RR with Time Quantum = 20*

- The Gantt chart is:

36

## *Multilevel Queue*

- Ready queue is partitioned into separate queues
  - Foreground (interactive)
  - Background (batch)
- Each queue has its own scheduling algorithm,
  - Foreground = RR
  - Background = FCFS
- Scheduling moust be done between the queues.
  - Fixed priority scheduling; i.e. serve all from foreground then from background. Possiblity of starvation.
  - Time slice -- each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e.,
    - 80% to foreground in RR
    - 20% to background in FCFS

37

## *Multilevel Feedback Queue*

- A process can move between the various queues; aging can be implemented this way.
- Multilevel feedback queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithm for each queue
  - method used to determine when to upgrade a process
  - method used to determine when to demote a process
  - method used to determine which queue a process will enter when that process needs service.

38

## *Example of Multilevel Feedback Queue*

- Three queues:
  - Q0 -- time quantum 8 milliseconds
  - Q1 -- time quantum 16 milleseconds
  - Q2 -- FCFS
- Scheduling
  - A new job enters Q0, which is seved FCFS. When it gains CPU, job receives 8 msec. If it does not minish, job is moved to Q1.
  - At Q1, job is again served FCFS and receives 16 additional msec. If it still does not complete, it is preempted and moved to queue Q2.
  - Strict priority between queues.

39

## *Multiple-Processor Scheduling*

- CPU scheduling becomes more complex when multiple CPUs are available.
- SMP -- Homogeneous processors within a multiprocessor.
  - Each processor runs scheduling code
  - Single ready queue
  - Locks to protext data structures
- AMP -- Asymmetric multiprocessing; only one processor access the system data structures and runs OS code, alleviates the need for data sharing.

40