

CS241 Operating Systems

CPU Scheduling (3)

Klara Nahrstedt

Lecture 12

2/15/2006

Content

Scheduling algorithms

- Batch systems
 - Shortest job first
- Interactive systems
 - Multi Queue
 - Real-Time Scheduling
- Thread Scheduling
- Introduction to Signals
- Summary

Administrative Notes

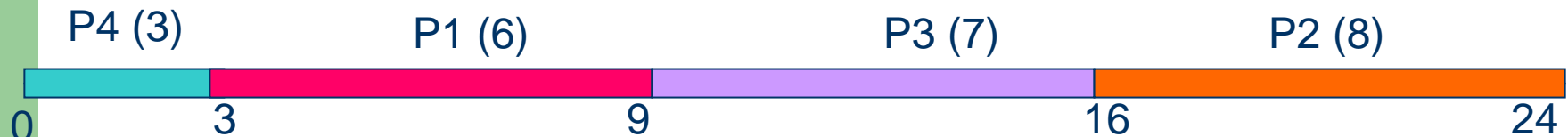
- MP2 on scheduling is running
- Quiz 3 will be on Friday, February 17, 2006
- Quiz 3 will include material from Tanenbaum, 3rd edition, section 2.2, 2.3, 2.4 AND LECTURE NOTES (see lecture note files)
 - lec6-syn1
 - lec7-syn2
 - lec8-syn3
 - lec8-syn4
 - lec10-sched
 - lec11-sched

Shortest Job First (SJF)

- Schedule the job with the shortest elapse time first
- Scheduling in Batch Systems
- Two types:
 - Non-preemptive
 - Preemptive
- Requirement: **the elapse time needs to be known in advance**
- **Optimal** if all jobs are available simultaneously (provable)
 - Gives the best possible AWT (average waiting time)

Non-preemptive SJF: Example

Process	Duration	Order	Arrival Time
P1	6	1	0
P2	8	2	0
P3	7	3	0
P4	3	4	0



P4 waiting time: 0
P1 waiting time: 3
P3 waiting time: 9
P2 waiting time: 16

The total time is: 24

The average waiting time (AWT):
 $(0+3+9+16)/4 = 7$

CS 241 - System Programming,
Klara Nahrstedt

Comparing to FCFS

Process	Duration	Order	Arrival Time
P1	6	1	0
P2	8	2	0
P3	7	3	0
P4	3	4	0



P1 waiting time: 0
P2 waiting time: 6
P3 waiting time: 14
P4 waiting time: 21

The total time is the same.
The average waiting time (AWT):
 $(0+6+14+21)/4 = 10.25$
(comparing to 7)

SJF is not always optimal

- Is SJF optimal if all the jobs are not available simultaneously?

Process	Duration	Order	Arrival Time
P1	10	1	0
P2	2	2	2



P1 waiting time: 0
P2 waiting time: 8

The average waiting time (AWT):
 $(0+8)/2 = 4$

Preemptive SJF

- Also called **Shortest Remaining Time First**
 - Schedule the job with the shortest remaining time required to complete
- Requirement: the elapse time needs to be known in advance

Preemptive SJF: Same Example

Process	Duration	Order	Arrival Time
P1	10	1	0
P2	2	2	2



The average waiting time (AWT):

P1 waiting time: $4-2 = 2$ $(0+2)/2 = 1$
P2 waiting time: 0

No CPU waste!!!

A Problem with SJF

- Starvation
 - In some condition, a job is waiting for ever
 - Example: SJF
 - Process A with elapse time of 1 hour arrives at time 0
 - But ever 1 minute, a short process with elapse time of 2 minutes arrive
 - Result of SJF: A never gets to run

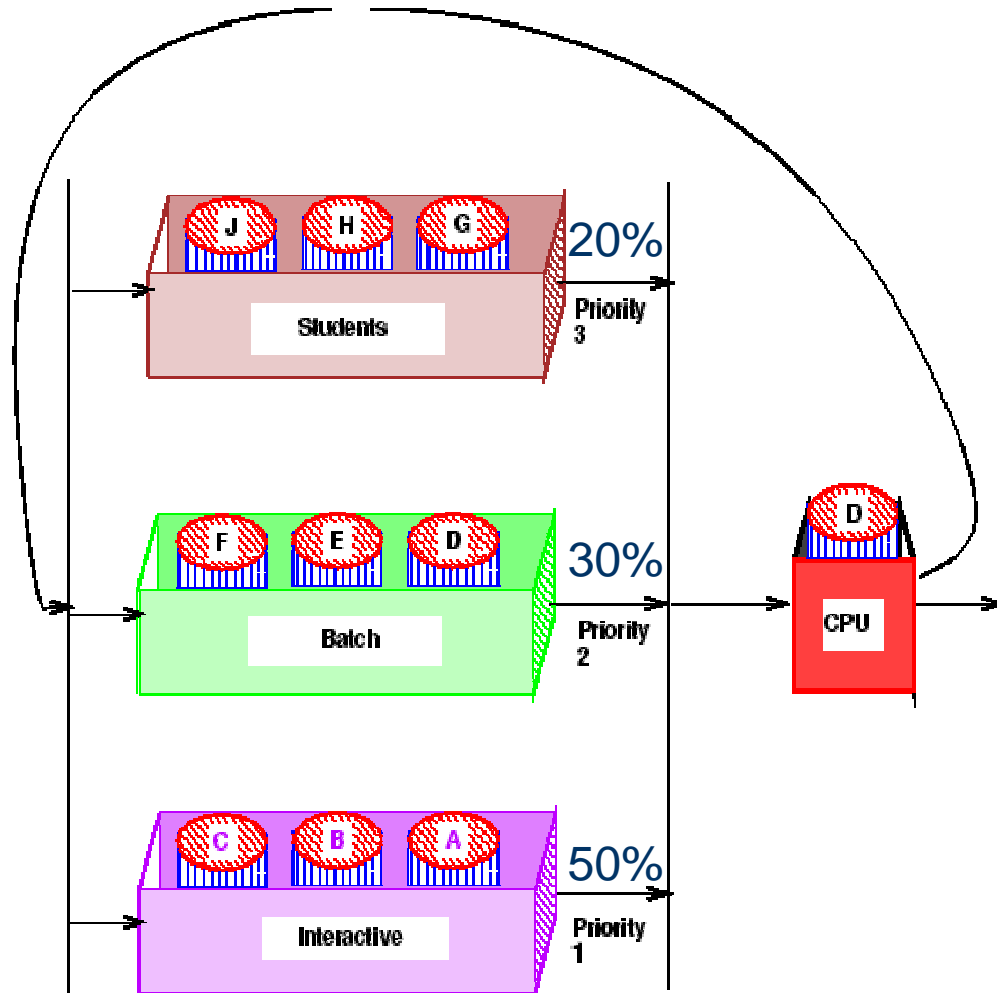
Interactive Scheduling Algorithms

- Usually preemptive
 - Time is **sliced** into quantum (time intervals)
 - Scheduling decision is also made at the beginning of each quantum
- Performance Criteria
 - Min Response time
 - best proportionality
- Representative algorithms:
 - Priority-based
 - Round-robin
 - Multi Queue & Multi-level Feedback
 - Shortest process time
 - Guaranteed Scheduling
 - Lottery Scheduling
 - Fair Sharing Scheduling

Multi-Queue Scheduling

- Hybrid between priority and round-robin
- Processes assigned to one queue permanently
- Scheduling between queues
 - Fixed Priorities
 - % CPU spent on queue
- Example
 - System processes
 - Interactive programs
 - Background Processes
 - Student Processes
- Address the starvation and infinite blocking problems

Multi-Queue Scheduling: Example



Real-Time Scheduling

- Hard-Real Time v Soft-Real Time
- Periodic vs Aperiodic
- A set of m periodic event streams is schedulable if the condition $\sum C_i/P_i \leq 1$ is satisfied, where $i=1, \dots, m$; C_i is the processing CPU time of an event stream i ; P_i is the period at which the event i occurs
- If the set of event streams are schedulable then the real-time scheduling policy **Earliest Deadline First** works
- Implementation of the scheduling policy: Earliest Deadline process is mapped to the highest priority

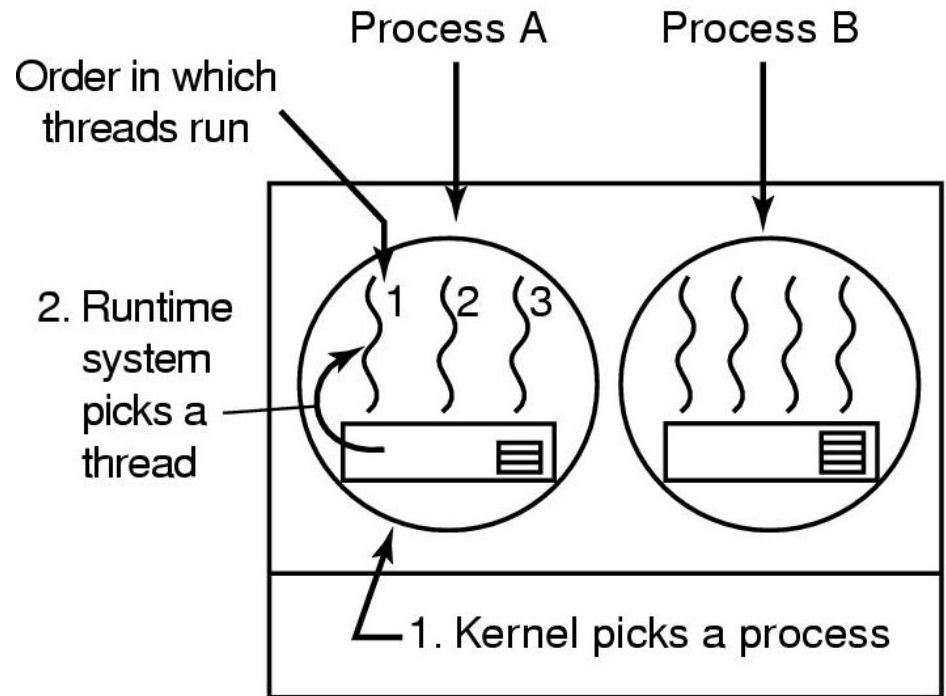
Priority Inversion and Inheritance

- **Priority inversion**
 - When a higher priority process needs to read or modify kernel data that are currently being accessed by a lower priority process.
 - The higher priority process must wait!
 - But the lower priority cannot proceed quickly due to scheduling.
- **Solution: priority inheritance** (Result of Prof. Sha, Rajkumar and Lehoczky Work)
 - When a lower-priority process accesses a resource, it inherits the priority level of the waiting high-priority process until it is done with the resource in question. And then its priority reverses to its natural value.

User-level Thread Scheduling

Possible Scheduling

- 50-msec process quantum
- run 5 msec/CPU burst



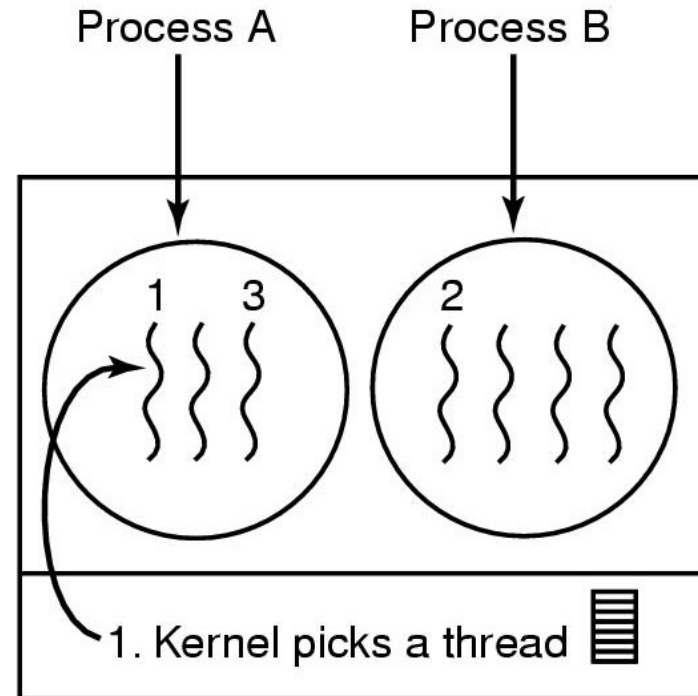
Possible: A1, A2, A3, A1, A2, A3

Not possible: A1, B1, A2, B2, A3, B3

Kernel-level Thread Scheduling

Possible scheduling

- 50-msec process quantum
- threads run 5 msec/CPU burst



Possible: A1, A2, A3, A1, A2, A3
Also possible: A1, B1, A2, B2, A3, B3

Introduction to Signals

(Ch8 pp255-283)

- **Signal is a software notification** to a process of an event.
 - Examples of such events
 - Process waits on a semaphore
 - Process waits to be scheduled
- A signal is *generated* when the event that causes the signal occurs.
- A signal is *delivered* when the process takes action based on the signal.
- The *lifetime* of a signal is the interval between its generation and delivery.
- A signal that has been generated but not yet delivered is *pending*.
- A process *catches* a signal if it executes a *signal handler* when the signal is delivered.

Introduction to Signals (2)

- Alternatively, a process can *ignore* a signal when it is delivered, that is to take no action.
- The function `sigaction` is used in signal handler to specify *what is to happen* to a signal when it is delivered.
- The *signal mask* determines the action to be taken when the signal is generated. It contains a list of currently *blocked signals*.
- A *blocked* signal is not delivered to a process until it is unblocked.
- The function `sigprocmask` is used to *modify the signal mask*.
- Each signal has a *default action* which is usually to terminate the process.

POSIX Required Signals (1)

Signal	Description	default action
SIGABRT	process abort	implementation dependent
SIGALRM	alarm clock	abnormal termination
SIGBUS	access undefined part of memory object	implementation dependent
SIGCHLD	child terminated, stopped or continued	ignore
SIGCONT	execution continued if stopped	continue
SIGFPE	error in arithmetic operation as in division by zero	implementation dependent
SIGHUP	hang-up (death) on controlling terminal (process)	abnormal termination
SIGILL	invalid hardware instruction	implementation dependent
SIGINT	interactive attention signal (usually ctrl-C)	abnormal termination
SIGKILL	terminated (cannot be caught or ignored)	abnormal termination

POSIX Required Signals (2)

Signal	Description	default action
SIGPIPE	write on a pipe with no readers	abnormal termination
SIGQUIT	Interactive termination core dump	implementation dependent
SIGSEGV	Invalid memory reference	implementation dependent
SIGSTOP	Execution stopped	stop
SIGTERM	termination	Abnormal termination
SIGTSTP	Terminal stop	stop
SIGTTIN	Background process attempting read	stop
SIGTTOU	Background process attempting write	stop
SIGURG	Highbandwidth data available on socket	ignore
SIGUSR1	User-defined signal 1	abnormal termination

Generating Signals

- All signal names are in `<signal.h>`
- You can send a signal to a process from the command line using `kill`
- `kill -l` will list the signals the system understands
- `kill [-signal] pid` will send a signal to a process.
- The optional argument may be a signal name or a number.
- The default is `SIGTERM`.
- To unconditionally kill a process, use:
`kill -9 pid` which uses the traditional signal (`SIGKILL`)

Programming a signal (1)

From a program you can use the kill system call as follows:

```
#include <sys/types.h>
#include <signal.h>
int kill(pid_t pid, int sig);
```

Example 8.4: send SIGUSR1 to process 3423:

```
if (kill(3423, SIGUSR1) == -1)
    perror("Failed to send the SIGUSR1 signal");
```

Example 8.5: a child kills its parent:

```
if (kill(getppid(), SIGTERM) == -1)
    perror("Failed to kill parent");
```

Programming a signal (2)

(A process sends a signal to itself)

```
#include <signal.h>
```

```
int raise(int sig);
```

- Example 8.6: send SIGUSR1 to itself

```
if (raise(SIGUSR1) != 0)
    perror("Failed to raise
    SIGUSR1");
```

```
#include <unistd.h>
```

```
Unsigned alarm(unsigned
    seconds)
```

- Example 8.8: kill an infinite loop after 10 seconds:

```
int main(void) {
    alarm(10);
    for ( ; ; ) ;
}
```

Summary

- Shortest job first (SJF)
- Multi-Queue
- Real-time Scheduling
- Thread Scheduling
- Introduction to Signals