5. Process and thread scheduling

5.1 Organization of Schedulers

- Embedded and Autonomous Schedulers
- Priority Scheduling

5.2 Scheduling Methods

- A Framework for Scheduling
- Common Scheduling Algorithms
- Comparison of Methods

5.3 Priority Inversion

5.4 Multiprocessor and Distributed Scheduling

Process and Thread Scheduling

- Process scheduling
  - Long term scheduling
  - Move process to *Ready List (RL)* after creation (When and in which order?)
- Dispatching
  - Short term scheduling
  - Select process from Ready List to run
- We use the term *scheduling* to refer to both

Organization of Schedulers

- Embedded
  - Called as function at end of kernel call
  - Runs as part of calling process
- Autonomous
  - Separate process
  - May have dedicated CPU on a multiprocessor
  - On single-processor, run at every quantum: scheduler and other processes alternate

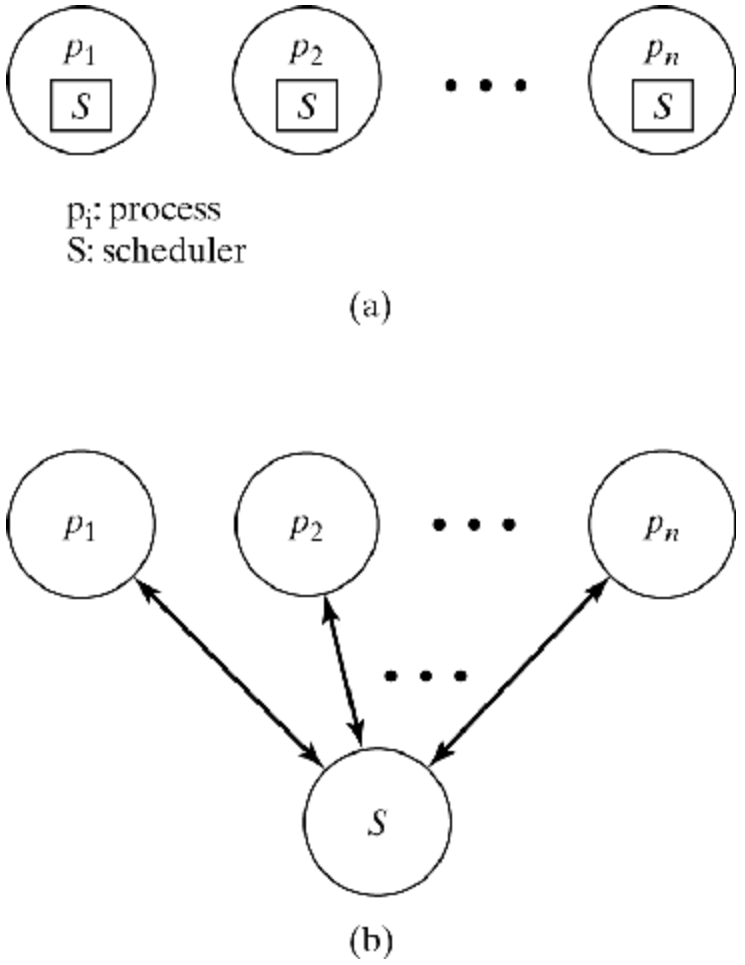

Figure 5-1

Priority Scheduling

- Priority function returns numerical value ***P*** for process ***p***: ***P = Priority(p)***
  - Static priority: unchanged for lifetime of ***p***
  - Dynamic priority: changes at runtime
- Priority divides processes into levels
  - implemented as multi-level Run List
  - ***p*** at RL[***i***] run before ***q*** at RL[***j***] if ***i>j***
  - ***p***, ***q*** at same level are ordered by other criteria

An Embedded Scheduler

```
Scheduler()
{
   do {
      Find highest priority process p with p.status == ready_a;
      Find a free cpu;
      if (cpu != NIL) Allocate_CPU(p,cpu);
   } while (cpu != NIL);
   do {
      Find highest priority process p with p.status == ready_a;
      Find lowest priority process q with p.status == running;
      if (Priority(p) > Priority(q)) Preempt(p,q);
   } while (Priority(p) > Priority(q));
   if (self->Status.Type!='running') Preempt(p,self);
}
```

Scheduling Methods

- When is scheduler invoked?
  - Decision mode
    - *Preemptive:* scheduler called periodically (quantum-oriented) or when system state changes
    - *Nonpreemptive:* scheduler called when process terminates or blocks
- How does it select highest priority process?
  - Priority function: ***P = Priority(p)***
    - Some common choices on next few slides
  - *Arbitration rule* for breaking ties
    - Random
    - Chronological (First In First Out = FIFO)
    - Cyclic (Round Robin = RR)

Priority function Parameters

- Possible parameters:
  - Attained service time (***a***)
  - Real time in system (***r***)
  - Total service time (***t***)
  - Period (***d***)
  - Deadline (explicit or implied by period)
  - External priority (***e***)
  - Memory requirements (mostly for batch)
  - System load (not process-specific)

Some Priority functions

- First in/First out (FIFO)
- Shortest Job First (SJF)
- Shortest Remaining Time (SRT)
- Round Robin (RR)
- Multi-Level (ML)

Scheduling algorithms

Name,	Decision mode,	Priority,	Arbitration
FIFO:	nonpreemptive	$P = r$	random
SJF:	nonpreemptive	$P = -t$	chronological/random
SRT:	preemptive	$P = -(t-a)$	chronological/random
RR:	preemptive	$P = 0$	cyclic
ML:	preemptive	$P = e$	cyclic
	nonpreemptive	$P = e$	chronological

- n fixed priority levels
- level P is serviced when n through P+1 empty

MLF (Multilevel Feedback)

- Like ML, but priority changes dynamically
- Every process enters at highest level ***n***
- Each level P prescribes maximum time t_P
- t_P increases as P decreases
- Typically:

  $t_n = T$ **(a constant)**

  $t_P = 2 \times t_{P+1}$

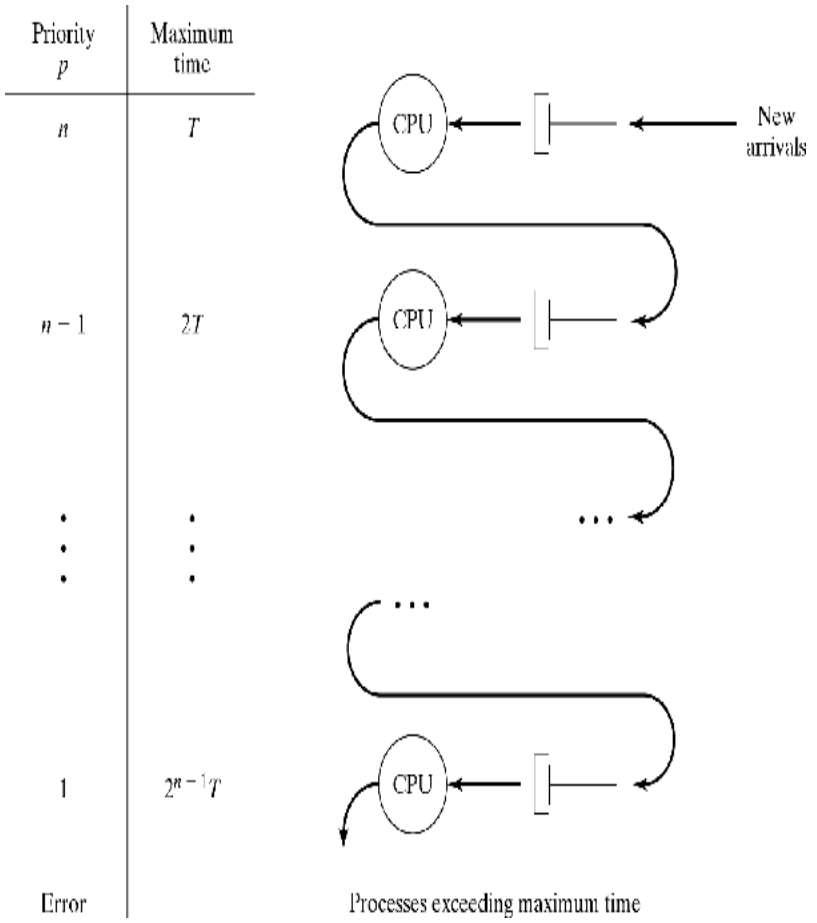

Figure 5-3

Scheduling algorithms

MLF priority function:

Find $P = n–i$ for given a:

priority	attained time
n	$a<T$
$n–1$	$a<T+2T$
$n–2$	$a<T+2T+4T$
. . .	. . .
$n–i$	$a<(2^{i+1}–1)T$

- Find smallest i such that $a<(2^{i+1}–1)T$:
- Solve for i: $\mathbf{i = \lfloor \log_2(a/T+1) \rfloor}$
- $\mathbf{P = n–i = n–\lfloor \log_2(a/T+1) \rfloor}$

Scheduling Algorithms

Rate Monotonic (RM):

- Intended for periodic (real-time) processes
- Preemptive
- Highest priority: shortest period: ***P = –d***

Earliest Deadline First (EDF):

- Intended for periodic (real-time) processes
- Preemptive
- Highest priority: shortest time to next deadline
  - $r \div d$ number of completed periods
  - $r \,\%\, d$ time in current period
  - $d - r \,\%\, d$ time remaining in current period
  - $P = -(d - r \,\%\, d)$ priority function

Comparison of Methods

- FIFO, SJF, SRT: Primarily for batch systems
  - FIFO simplest
  - SJF & SRT have better average turnaround times: ***(r1+r2+…+rn)/n***

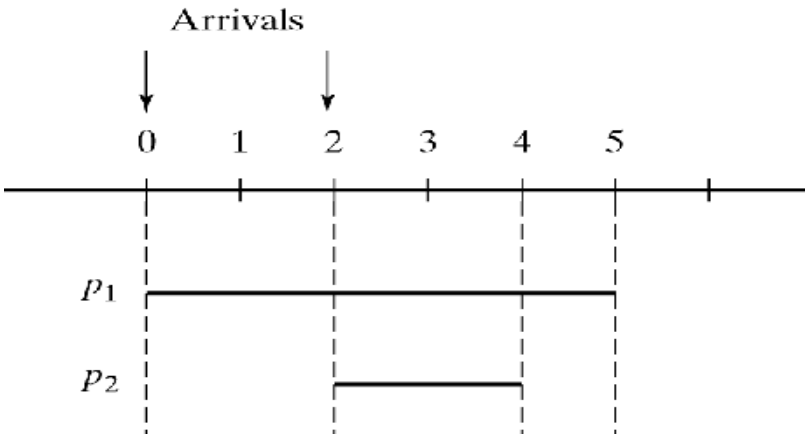

Figure 5-2

Average turnaround times:

FIFO: ((0+5) + (3+2))/2 = 5.0

SRT: ((2+5) + (0+2))/2 = 4.5

Comparison of Methods

- Time-sharing systems
  - Response time is critical
  - RR or MLF with RR within each queue are suitable
  - Choice of quantum determines overhead
    - When $q \rightarrow \infty$, RR approaches FIFO
    - When $q \rightarrow 0$, context switch overhead $\rightarrow$ 100%
    - When q is much greater than context switch overhead, n processes run concurrently at $1/n$ CPU speed

Comparison of Methods

- Real-time systems
  - *Feasible:* All deadlines are met
  - *CPU utilization* is defined as: $U=\sum t_i/d_i$
  - If schedule is feasible, $U \leq 1$
  - EDF always yields feasible schedule *provided* $U \leq 1$.
  - RM yields feasible schedule if U is not too big (no more than approximately 0.7). Otherwise, it may fail.

Example where RM fails

- Process p1 has service time 1.5, period 4
- Process p2 has service time 3, period 5
- U=(1.5/4) +3/5=.975 < 1
- RM fails

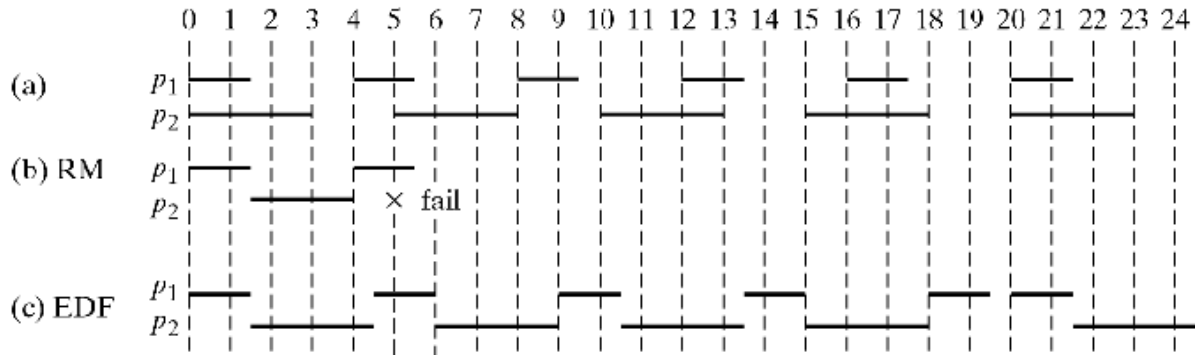

Figure 5-9

Priority Inversion Problem

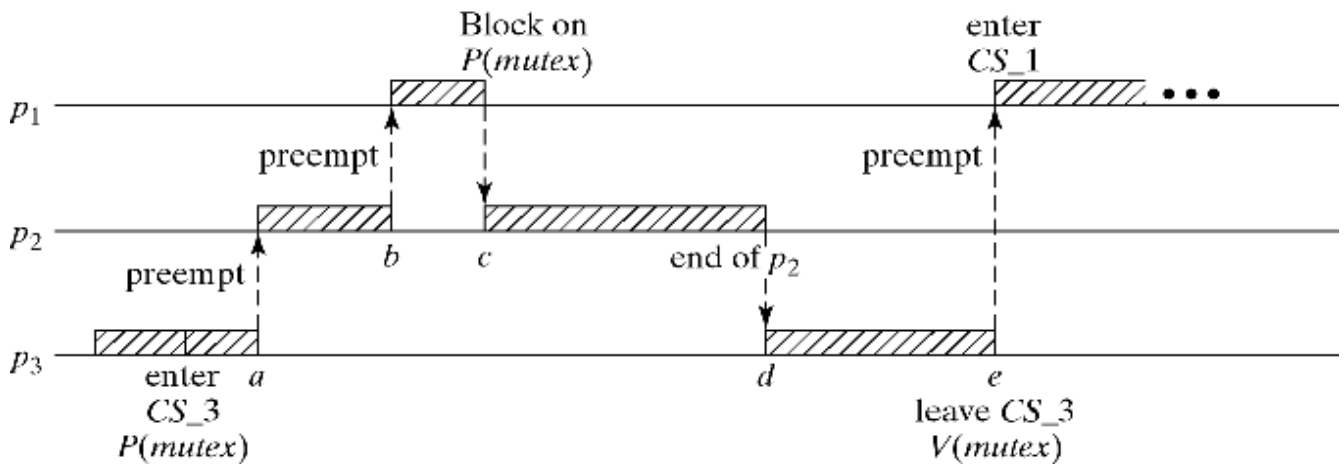

Figure 5-10

- Assume priority order ***p1>p2>p3***
- ***p3*** enters CS; ***p2*** preempts ***p3***; ***p1*** preempts ***p2***; ***p1*** blocks on CS
- Effect: process ***p2***, *unrelated to* ***p1*** *and of lower priority*, may delay ***p1*** indefinitely.
- Note: problem is not simply that ***p1*** blocks. This is unavoidable. The problem is that ***p1*** is waiting on ***p2***.
- Problem would not occur if ***p3*** in CS had priority greater than ***p2***

Priority Inversion Problem

- Naïve "solution": Always run CS at priority of highest process that shares the CS.
- Problem: ***p1*** cannot interrupt a lower-priority process inside its CS even if ***p1*** is not trying to enter its CS. This is a different form of priority inversion.
- Better solution: "Dynamic Priority Inheritance"…

Priority Inversion Problem

Dynamic Priority Inheritance:

- When ***p3*** is in its CS and ***p1*** attempts to enter its CS…
  - ***p3*** inherits ***p1***'s (higher) priority for the duration of CS

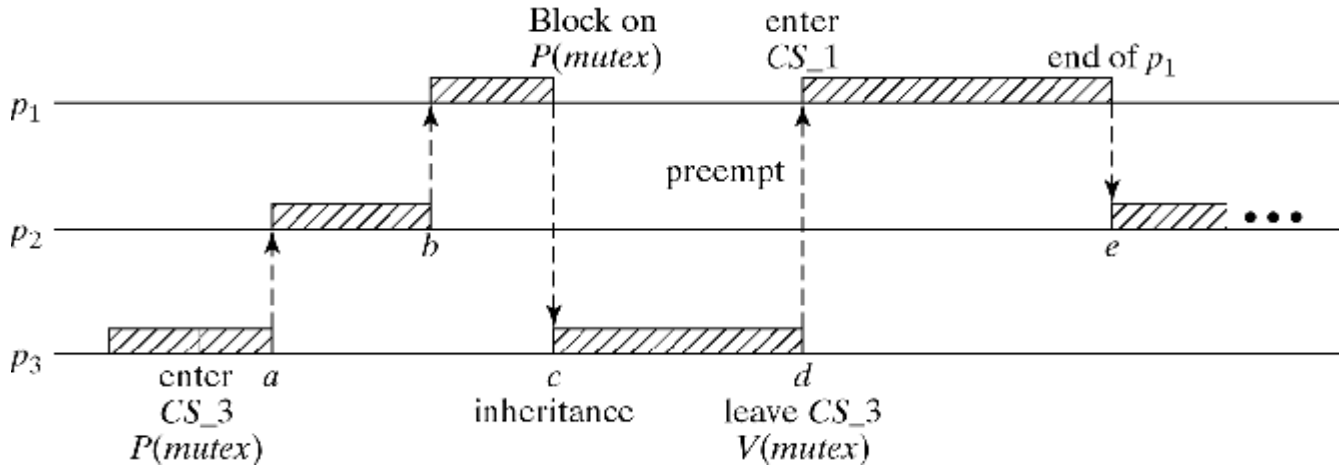

Figure 5-11

Multiprocessor and Distributed Scheduling

- Two Principle approaches
  - *Single Scheduler*
    - All processors are in the same resource pool
    - Any process can be allocated to any processor
  - *Multiple Schedulers*
    - Processors are divided into sets of separately schedule machines, each with its own scheduler
    - Each process is permanently preallocated to a particular group
    - Useful when different processors have different characteristics and functions
- Key problem: *load balancing*
  - Evenly distributing load over multiple machines

History

- Originally developed by Steve Franklin
- Modified by Michael Dillencourt, Summer, 2007
- Modified by Michael Dillencourt, Spring, 2009