



CPU Scheduling

Operating Systems

Yu-Ting Wu

(with slides borrowed from Prof. Jerry Chou)

1

Operating Systems 2022

Outline

- Basic concepts
- Scheduling algorithms
- Special scheduling issues
- Scheduling case study

2

2

Operating Systems 2022

Basic Concepts

3

3

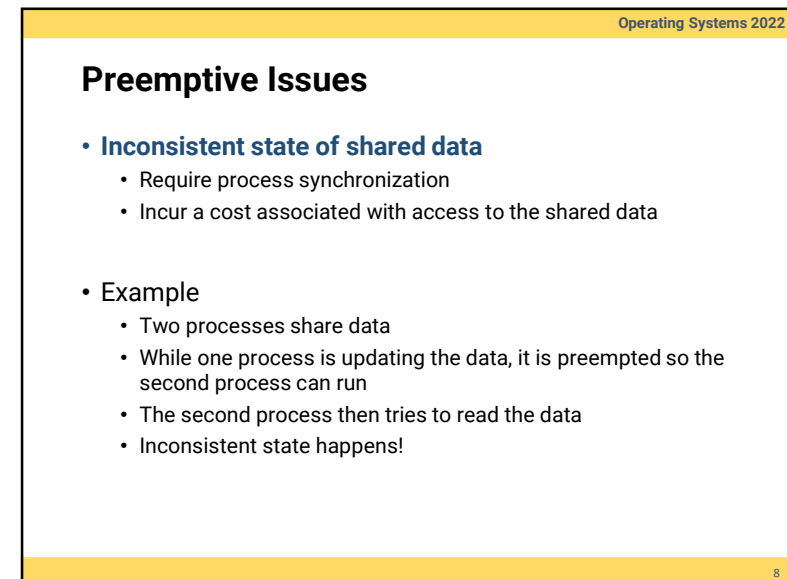
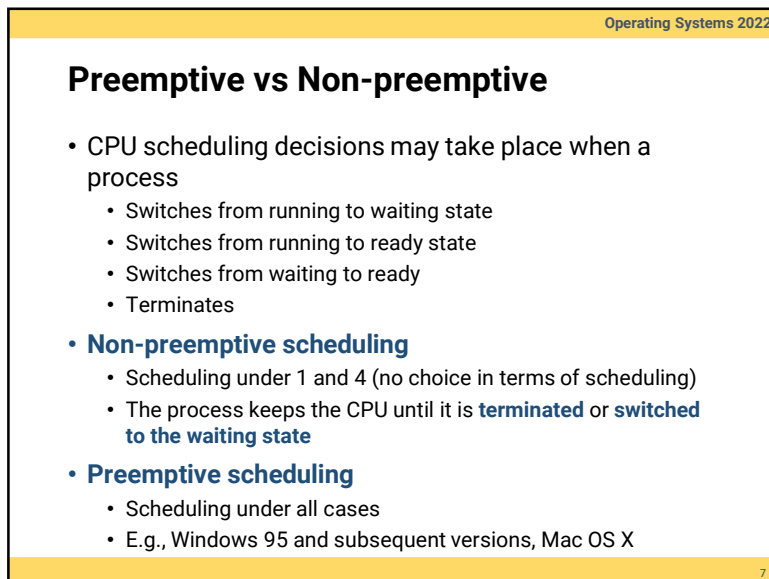
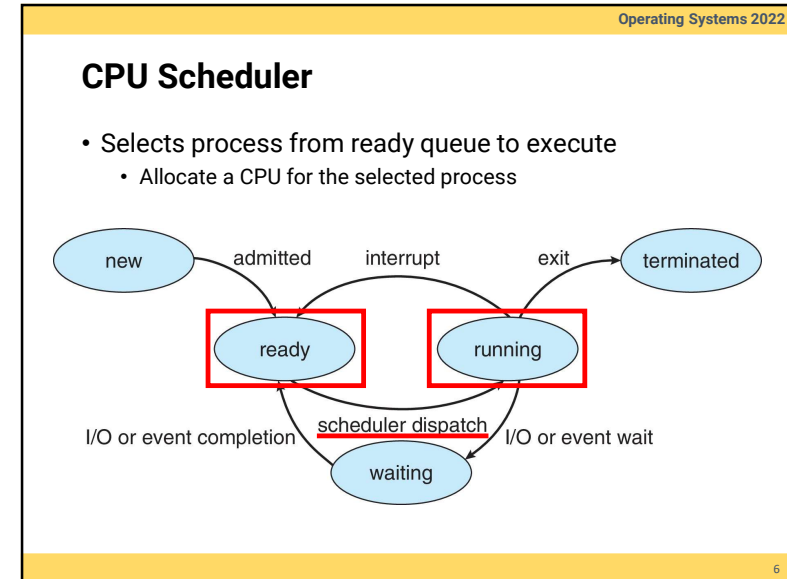
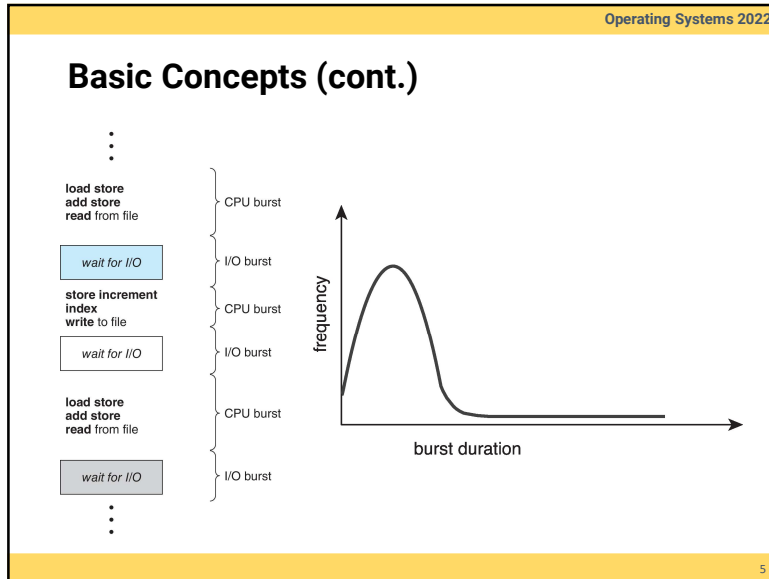
Operating Systems 2022

Basic Concepts

- The idea of multi-programming
 - Keep several processes in memory
 - Every time one process has to wait, another process takes over the use of the CPU
- **CPU-I/O burst cycle**
 - Process execution consists of a cycle of **CPU execution (CPU burst)** and **I/O wait (I/O burst)**
 - Generally, there is a large number of short CPU bursts, and a small number of long CPU bursts
 - An I/O-bound program would typically has many very short CPU bursts
 - A CPU-bound program might have a few long CPU bursts

4

4



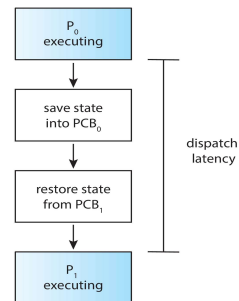
Dispatcher

- Dispatcher module gives control of the CPU to the process selected by scheduler

- Switch context
- Jump to the proper location in the selected program

- Dispatch latency**

- Time for the dispatcher to stop one process and start another process
 - Context switch time



9

Scheduling Algorithms

10

10

Scheduling Criteria

- CPU utilization**

- Theoretically 0% ~ 100%
- Real systems: 40% (light) ~ 90% (heavy)

- Throughput**

- Number of completed processes per time unit

system view

- Turnaround time**

- Submission ~ completion

- Waiting time**

- Total waiting time in the ready queue

- Response time**

- Submission ~ the first response is produced

single job view

11

11

Scheduling Criteria

- Max** CPU utilization
- Max** Throughput
- Min** Turnaround time
- Min** Waiting time
- Min** Response time

12

12

Algorithms

- First-Come, First-Served (FCFS) scheduling
- Shortest-Job-First (SJF) scheduling
- Priority scheduling
- Round-Robin scheduling
- Multi-level queue scheduling
- Multi-level feedback queue scheduling

13

FCFS Scheduling

- Process (burst time) in arriving order
 - P1 (24), P2 (3), P3 (3)
- The Gantt Chart of the schedule



- Waiting time: P1 = 0, P2 = 24, P3 = 27
- Average Waiting Time (AWT): $(0 + 24 + 27) / 3 = 17$
- **Convoy effect**
 - Short processes behind a long process

14

FCFS Scheduling (cont.)

- Process (burst time) in arriving order
 - P2 (3), P3 (3), P1 (24)
- The Gantt Chart of the schedule



- Waiting time: P1 = 6, P2 = 0, P3 = 3
- Average Waiting Time (AWT): $(6 + 0 + 3) / 3 = 3$

15

Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst
- A process with shortest burst length gets the CPU first
- SJF provides the **minimum (optimal) average waiting time**
- Two schemes
 - **Non-preemptive**
 - Once the CPU is given to a process, it cannot be preempted until its completion
 - **Preemptive**
 - If a new process arrives with shorter burst length, preemption happens

16

Non-Preemptive SJF Example

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

Ready queue at t = 0

P1 (7)

Scheduling

P1



17

17

Non-Preemptive SJF Example (cont.)

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

Ready queue at t = 7

P2 (4) P3 (1) P4 (4)

Scheduling

P1 P3



18

18

Non-Preemptive SJF Example (cont.)

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

Ready queue at t = 8

P2 (4) P4 (4)

Scheduling

P1 P3 P2



19

19

Non-Preemptive SJF Example (cont.)

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

Ready queue at t = 12

P4 (4)

Scheduling

P1 P3 P2 P4



- **Wait time = completion time - arrival time - run time**
- **AWT** = $[(7-0-7) + (12-2-4) + (8-4-1) + (16-5-4)] / 4 = 4$
- **Response time:** P1 = 0, P2 = 6, P3 = 3, P4 = 7

20

20

Preemptive SJF Example

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

Ready queue at $t = 0$

P1 (7)

Scheduling

P1



21

21

Preemptive SJF Example

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

Ready queue at $t = 2$

P1 (5) P2 (4)

Scheduling

P1 P2



22

22

Preemptive SJF Example

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

Ready queue at $t = 4$

P1 (5) P2 (2) P3 (1)

Scheduling

P1 P2 P3



23

23

Preemptive SJF Example

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

Ready queue at $t = 5$

P1 (5) P2 (2) P4 (4)

Scheduling

P1 P2 P3 P2



24

24

Preemptive SJF Example

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

Ready queue at t = 7



Scheduling



25

25

Preemptive SJF Example

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

Ready queue at t = 11



Scheduling



- **Wait time = completion time - arrival time - run time**
- **AWT** = $[(16-0-7) + (7-2-4) + (5-4-1) + (11-5-4)] / 4 = 3$
- **Response time**: P1 = 0, P2 = 0, P3 = 0, P4 = 2

26

26

Approximate Shortest-Job-First (SJF)

• SJF difficulty

- No way to know length of the next CPU burst

• Approximate SJF

- The next burst can be predicted as an exponential average of the measured length of previous CPU bursts

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

new one history

$$= \alpha t_n + (1 - \alpha) \alpha t_{n-1} + (1 - \alpha)^2 \alpha t_{n-2} + \dots$$

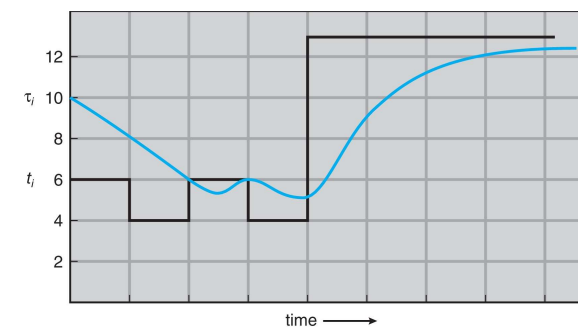
Example:

$$\alpha = 1/2 \quad = \left(\frac{1}{2}\right)t_n + \left(\frac{1}{2}\right)^2 t_{n-1} + \left(\frac{1}{2}\right)^3 t_{n-2}$$

27

27

Exponential Prediction of Next CPU Burst



CPU burst (t_i)

"guess" (τ_i)

28

28

Priority Scheduling

- A **priority number** is associated with each process
- The CPU is allocated to the highest priority process
 - Preemptive
 - Non-preemptive
- **SJF is a priority scheduling** where priority is the predicted next CPU burst time
- Problem: **starvation**
 - Low priority processes never execute
 - Example: IBM 7094 shutdown at 1973, a 1967-process never run
 - Solution: **aging**
 - As time progresses, increase the priority of processes

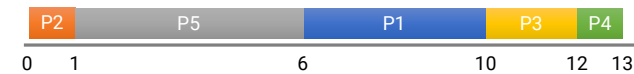
29

29

Priority Scheduling (cont.)

Process	Burst Time	Priority
P1	4	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

Gantt Chart



30

30

Round-Robin (RR) Scheduling

- Each process gets a small unit of CPU time (**time quantum, TQ**), usually 10 ~ 100 ms
 - Context switch time usually < 10 microseconds
- After TQ elapsed, 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 , each process gets $1/n$ of the CPU time (q time units)
 - No process waits more than $(n-1)q$ time units
- Performance
 - TQ large \rightarrow FIFO
 - TQ small \rightarrow (context switch) overhead increases

31

31

Round-Robin (RR) Scheduling (cont.)

Process	Burst Time
P1	53
P2	17
P3	68
P4	24

- If TQ = 20, the Gantt Chart is



- Typically, higher average turnaround than SJF, but better response

32

32

Priority Scheduling with Round-Robin

Process	Burst Time	Priority
P1	4	3
P2	5	2
P3	4	2
P4	7	1
P5	3	3

- Run the process with the highest priority
- Processes with the same priority run round-robin

Gantt Chart (TQ = 2)

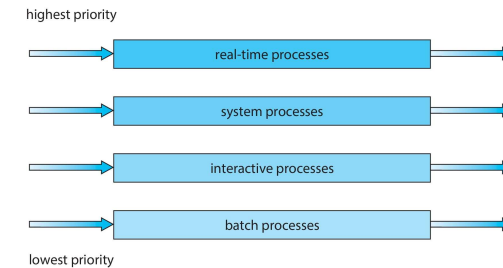


33

33

Multi-level Queue Scheduling

- Ready queue is partitioned into separate queues
- Each queue has its own scheduling algorithm
- Scheduling must be done **between** queues
 - Time slice: each queue gets a certain amount of CPU

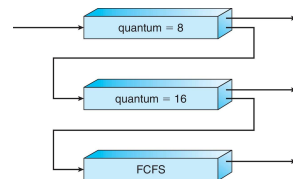


34

34

Multi-level Feedback Queue Scheduling

- A process can move between the various queues
 - Aging must be implemented
- Idea: separate processes according to the characteristic of their CPU burst
 - I/O-bound and interactive processes in **higher priority** queue → short CPU burst
 - CPU-bound processes in **lower priority** queue → long CPU burst



35

35

Multi-level Feedback Queue Scheduling

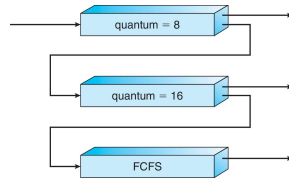
- In general, multi-level feedback queue scheduler is defined by the following parameters
 - Number of queues
 - Scheduling algorithm for each queue
 - Method used to determine when to upgrade a process
 - Method used to determine when to demote a process

36

36

Multi-level Feedback Queue (cont.)

- Three queues
 - Q0: RR with TQ 8 ms.
 - Q1: RR with TQ 16 ms.
 - Q2: FCFS
- Scheduling
 - A new process enters queue Q0 which is served in RR
 - When it gains CPU, the process receives 8 ms
 - If it does not finish in 8 ms., the process is moved to queue Q1
 - At Q1, job is again served in RR and receives 16 additional ms.
 - If it still does not complete, it is preempted and moved to queue Q2



37

37

Evaluation Methods

- **Deterministic modeling**
 - Take a particular predetermined workload and define the performance of each algorithm for that workload
- **Queueing model**
 - Mathematical analysis
- **Simulation**
 - Random number generator or trace tapes for workload generation
- **Implementation**
 - The only completely accurate way for algorithm evaluation

38

38

Deterministic Modeling

- Take a particular predetermined workload and define the performance of each algorithm for that workload
- Example: 5 processes arriving at time 0

	P1	P2	P3	P4	P5
Burst Time	10	29	3	7	12

FCFS AWT = 28

Non-preemptive SFJ AWT = 13

RR (TQ = 23) AWT = 22

39

39

Queueing Models

- Describe the arrival of processes, and CPU and I/O bursts **probabilistically**
- Compute average throughput, utilization, waiting time, etc.

40

40

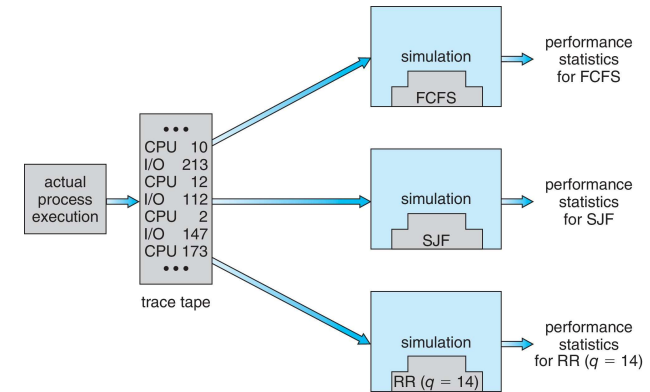
Simulations

- Queueing models are limited
- Simulations are more accurate
- Consider
 - Programmed model of computer system
 - Clock is a variable
- Gather statistics indicating algorithm performance
- Data to drive simulation
 - Random number generator according to probabilities
 - Distributions defined mathematically or empirically
 - Trace tapes record sequences of real events in real systems

41

41

Simulations (cont.)



42

42

Implementation

- Even simulations have limited accuracy
- The only completely accurate way for algorithm evaluation is to implement new scheduler and test in real systems
 - High cost and risk
 - Also need to consider the varieties of environments

43

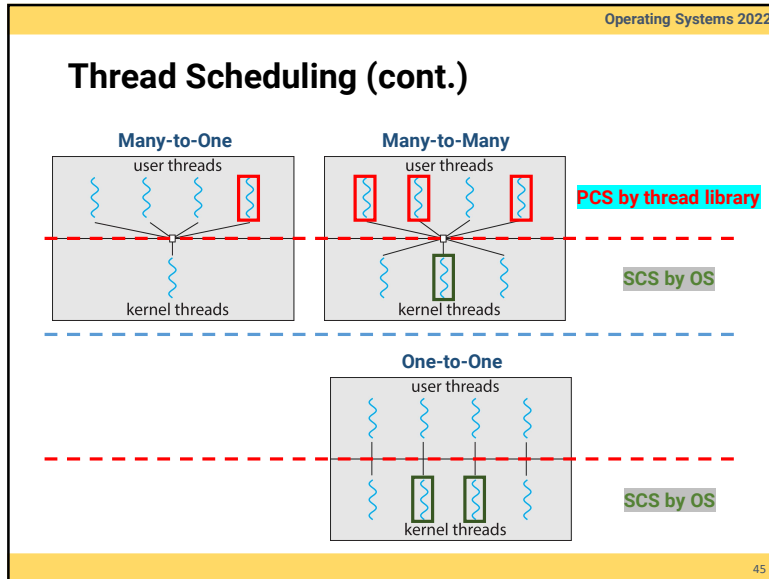
43

Thread Scheduling

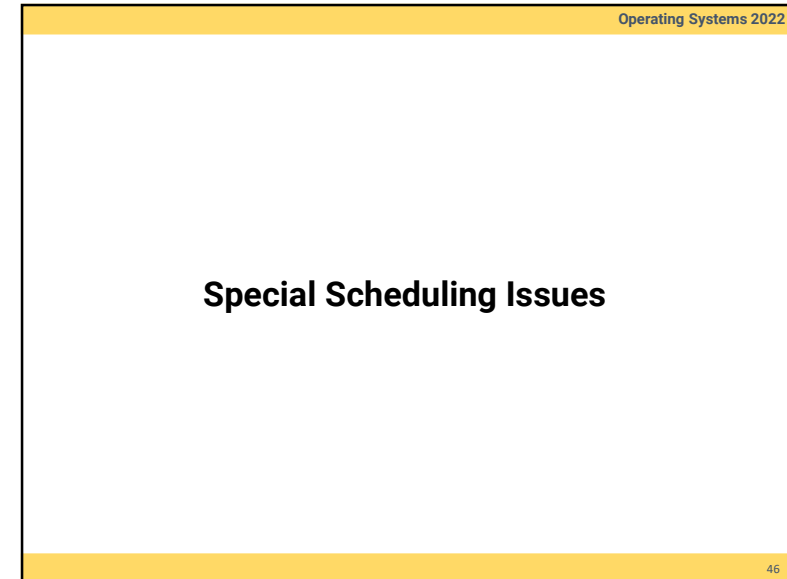
- Distinction between user-level and kernel-level threads
- When threads supported, threads scheduled, not processes
- For **many-to-one** and **many-to-many** models, **thread library** schedules **user-level threads** to run on light-weight process
 - Known as **process-contention scope (PCS)** since scheduling competition is within the process
 - Typically done via **priority** set by programmer
- Kernel thread scheduled onto available CPU is **system-contention scope (SCS)** – competition among all threads in system

44

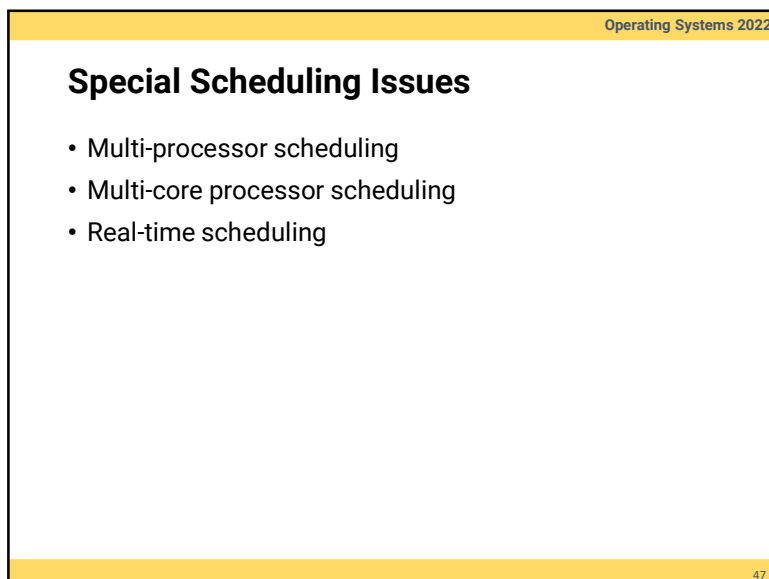
44



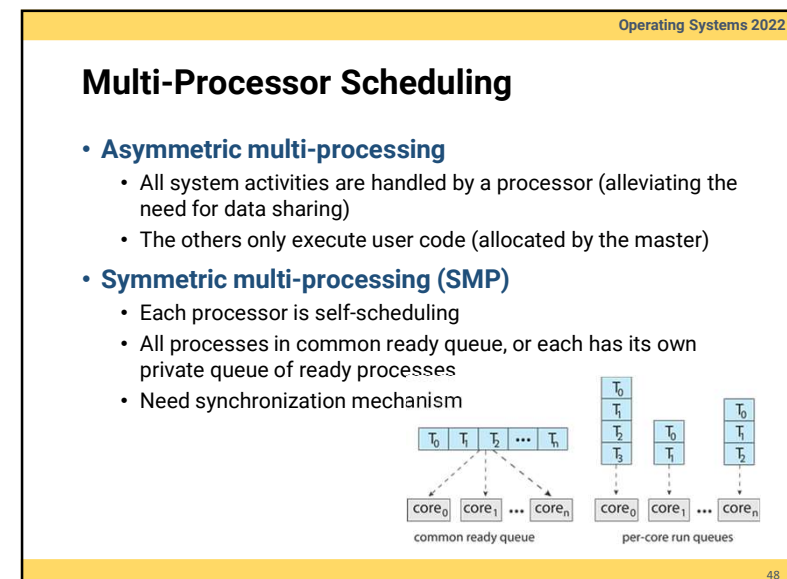
45



46



47



48

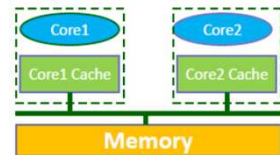
Processor Affinity

• Processor affinity

- A process has an affinity for the processor on which it is currently running
 - A process populates its recent used data in cache
 - Cache invalidation and repopulation has high cost

• Solution

- Soft affinity**
 - Possible to migrate to other processors
- Hard affinity**
 - Not to migrate to other processor

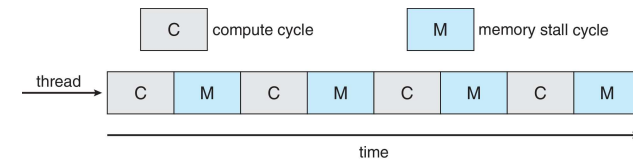


49

49

Multi-core Processors

- Recent trend to place multiple processor cores **on same physical chip**
- Faster and consumes less power**
- Memory stall
 - When access memory, it spends a significant amount of time waiting for the data become available (e.g., cache miss)

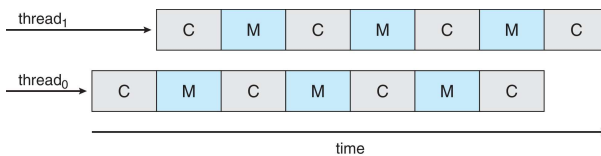


50

50

Multi-threaded Multi-core System

- Two (or more) hardware threads are assigned to each core
 - Each core has > 1 hardware threads
- Take advantage of memory stall to make progress on another thread while memory retrieve happens
 - If one thread has a memory stall, switch to another thread



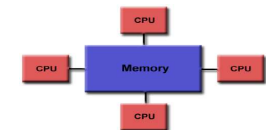
51

51

Memory Access Architecture

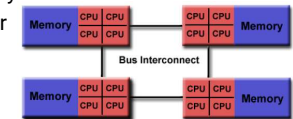
• Uniform Memory Access (UMA)

- Most commonly represented today by Symmetric multi-processor (SMP) machines
- Equal access times to memory
- Example: most commodity computers



• Non-Uniform Memory Access (NUMA)

- Often made by physically linking two or more SMPs
- One SMP can directly access memory of another SMP
- Memory access across link is slower
- Example: IBM Blade server

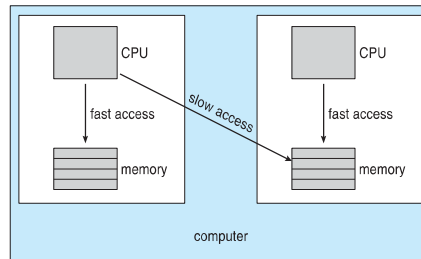


52

52

NUMA and CPU Scheduling

- Occurs in systems containing combined CPU and memory boards
- CPU scheduler and memory-placement works together



53

Load Balancing

- Keep the workload evenly distributed across all processors
 - Only necessary on systems where each processor has its own private queue of eligible processes to execute
- Two strategies**
 - Push migration**
 - Move (push) processes from overloaded to idle or less-busy processor
 - Pull migration**
 - Idle processor pulls a waiting task from a busy processor
- Load balancing often counteracts the benefits of processor affinity

54

Real-time Scheduling

- Real-time does not mean speed, but **keeping deadlines**
- Soft real-time requirements**
 - Missing the deadline is unwanted, but is not immediately critical
 - Example: multimedia streaming
- Hard real-time requirements**
 - Missing the deadline results in a fundamental failure
 - Example: nuclear power plant controller

55

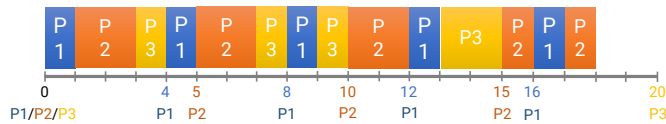
Real-time Scheduling Algorithms

- Must support **preemptive, priority-based** scheduling
 - But only guarantees **soft real-time**
- Description**
 - $T1 = (0, 4, 10) == (\text{Ready}, \text{Execution}, \text{Period})$
 - $T2 = (1, 2, 4)$
- Rate-Monotonic (RM) algorithm**
 - Shorter period \rightarrow high priority**
 - Fixed-priority real-time system scheduling algorithm
- Earliest-deadline-first (EDF) algorithm**
 - Earlier deadline \rightarrow higher priority**
 - Dynamic priority algorithm

56

Rate-Monotonic (RM) Scheduler

- **Fixed-priority scheduling**
 - All jobs of the same task have same priority
 - The task's priority is fixed
- **The shorter period, the higher priority**
- **Example (Preempted)**
 - $T1 = (4, 1)$, $T2 = (5, 2)$, $T3 = (20, 5)$ for (period, execution)
 - Period: $4 < 5 < 20$
 - Priority: $T1 > T2 > T3$

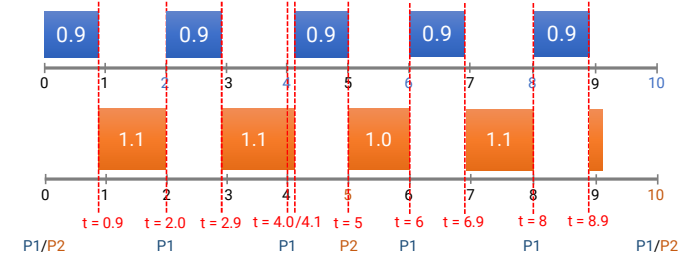


57

57

Early Deadline First (EDF) Scheduler

- **Dynamic-priority scheduler**
 - Task's priority is not fixed
 - Task's priority is determined by deadline
- **Example (preempted)**
 - $T1 = (2, 0.9)$, $T2 = (5, 2.3)$ for (period, execution)



58

58

Scheduling Case Study

59

59

Operating System Examples

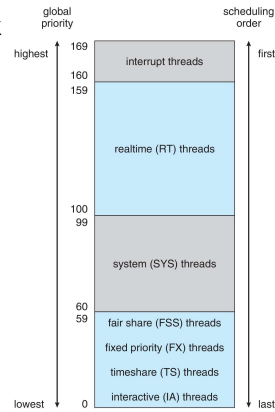
- Solaris
- Windows
- Linux

60

60

Solaris Scheduler

- **Priority-based** multi-level feedback queue scheduling
- Six classes of scheduling
 - Real-time
 - System
 - Time sharing
 - Interactive
 - Fair share
 - Fixed priority
- Each class has its **own priorities and scheduling algorithm**
- The scheduler converts the class specific priorities into global priorities



61

Solaris Scheduler (cont.)

- For time sharing and interactive processes
 - Inverse relationship between priorities and time slices:
the higher the priority, the smaller the time slice

- **Time quantum expired**
 - The new priority of a thread that has used its entire time quantum without blocking
- **Return from sleep**
 - The new priority of a thread that is returning from sleeping (I/O wait)

priority	time quantum	time quantum expired	return from sleep
0	200	0	50
5	200	0	50
10	160	0	51
15	160	5	51
20	120	10	52
25	120	15	52
30	80	20	53
35	80	25	54
40	40	30	55
45	40	35	56
50	40	40	58
55	40	45	58
59	20	49	59

62

Windows XP Scheduler

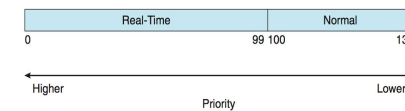
- Similar to Solaris: Multi-level feedback queue
- Scheduling
 - From the highest priority queue to lowest priority queue (0 ~ 31)
 - The highest-priority thread always run
 - Round-robin in each priority queue
 - Priority changes dynamically except for real-time class

	real-time	high	above normal	normal	below normal	idle priority
time-critical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1

63

Linux Scheduler

- Preemptive priority based scheduling
 - But allows only user mode processes to be preempted
 - Two separate process priority ranges
 - **Lower values indicate higher priorities**
 - Higher priority with longer time quantum
- **Real-time tasks (0 ~ 99)**
 - Static priorities
- **Other tasks (100 ~ 140)**
 - Dynamic priorities based on task interactivity



64

61

62

63

64

Objectives Review

- Describe various CPU scheduling algorithms
- Explain the issues related to multiprocessor and multicore scheduling
- Describe various real-time scheduling algorithms
- Describe the scheduling algorithms used in the Windows, Linux, and Solaris operating systems