Soongsil Univ. http://design.ssu.ac.kr Digital System Design Lab.

Ch. 05 CPU Scheduling

Chanho Lee
Soongsil University

1

Objectives

- Describe various CPU scheduling algorithms.
- Scheduling criteria.
- Multiprocessor and multicore scheduling.
- Real-time scheduling algorithms.
- Windows, Linux, and Solaris operating systems.
- Modeling and simulations to evaluate CPU scheduling algorithms.
- Implements several different CPU scheduling algorithms.

2

Soongsil Univ.
http://design.ssu.ac.kr

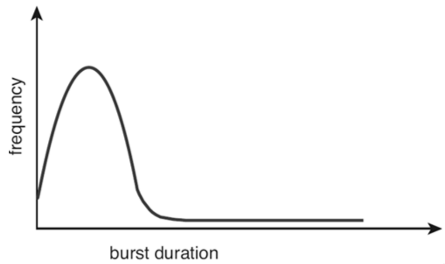
Digital System Design Lab.

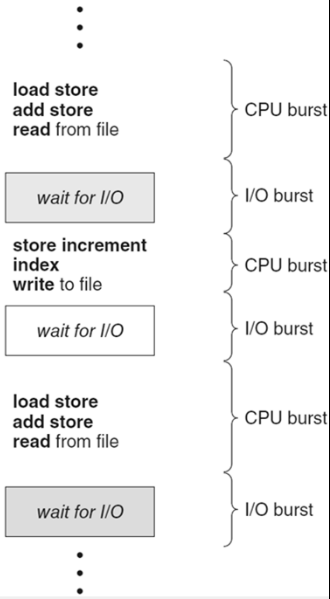
7주차

3

5.1 Basic Concepts

- Objective of multiprogramming: to maximize CPU utilization.
  - ►Process execution until waiting state: typically, I/O request.
  - ►Multiple processes in memory (ready queue) → scheduling before use
  - ►Fundamental OS function
- CPU–I/O Burst Cycle
  - ►Process execution: alternating sequence of CPU and I/O bursts
  - ►Histogram of CPU-burst durations
  - ►I/O-bound program: many short CPU bursts
  - ►CPU-bound program: a few long CPU bursts





Soongsil Univ. Digital System Design Lab.

Operating Systems

 4

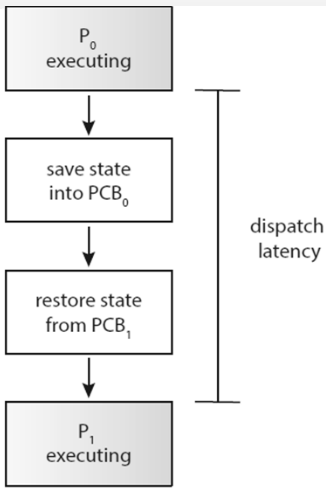
4

5.1 Basic Concepts

- CPU Scheduler
  - ►selects a process in ready queue and allocates a CPU
  - ►Ready queue: a FIFO queue, a priority queue, a tree, or simply an unordered linked list
  - ►PCBs have the records in the queues
- Preemptive and Nonpreemptive Scheduling
  - ►CPU-scheduling decisions when a process
    - 1. switches from the running to the waiting state or an invocation of wait()
    - 2. switches from the running to the ready state (for example, an interrupt)
    - 3. switches from the waiting to the ready state (for example, at completion of I/O)
    - 4. terminates
  - ►Nonpreemptive or cooperative scheduling scheme:
    - Process keeps the CPU until released
    - Scheduling for 1 and 4 only (no choice in terms of scheduling)
  - ►Preemptive: switching states by scheduling scheme (for 2 and 3). Modern OS.
    - results in race conditions when data are shared among several processes (synchronization: ch.6)
      - Ex. Process A updates the data → preempted → process B reads the data (before updated)

5.1 Basic Concepts

- Dispatcher
  - ►gives control of the CPU's core to the process
  - ►functions
    - Switching context
    - Switching to user mode
    - Jumping to the proper location in the user program to resume
  - ►dispatch latency: time to stop one process and start another
    - should be as short as possible
  - ►Frequency of the context switches
    - Linux: `vmstat 1 3` : 3 lines of output over a 1-second delay




```
------cpu-----
24      : average number of context switches over 1 second since the system booted
225     : number of context switches in the past second
339     : number of context switches in the second prior to above
```

- `cat /proc/2166/status` : list various statistics for the process with pid = 2166

```
voluntary ctxt switches      150
nonvoluntary ctxt switches   8       : preempted
```

5.2 Scheduling Criteria

- Affects the choice of scheduling algorithm
- CPU utilization
  - Rate of CPU being busy
  - Ideally, 0 to 100 percent. In a real system, from 40 percent (for a lightly loaded system) to 90 percent (for a heavily loaded system).
  - Linux, macOS, and UNIX: `top` command for CPU utilization
- Throughput
  - Number of completed processes per time unit
- Turnaround time
  - Time to execute: from submission to completion
  - sum of the periods for waiting in the ready queue, executing on the CPU, and doing I/O.
- Waiting time
  - sum of the periods for waiting in the ready queue
  - affected by scheduling algorithm

5.2 Scheduling Criteria

- Response time.
  - Time to start responding, not the time to output the response.
  - Interactive system
- It is desirable
  - to maximize CPU utilization and throughput
  - to minimize turnaround time, waiting time, and response time
- Optimization
  - Average measure in most cases, or
  - minimum or maximum values
    - Ex. To guarantee service time, to minimize the maximum response time
  - for interactive systems (such as a PC desktop or laptop system), it is more important to minimize the variance in the response time than to minimize the average response time
    - Little work has been done

5.3 Scheduling Algorithms

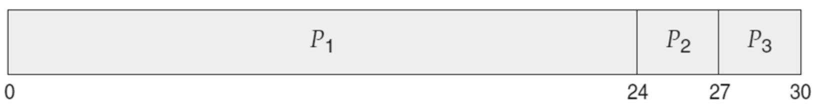
- Assumption
  - ► one CPU burst (in milliseconds) per process
  - ► comparing the average waiting time
  - ► only one processing core available
- First-Come, First-Served (FCFS) Scheduling
  - ► Simplest
  - ► FIFO queue
  - ► convoy effect:
    - All the other processes wait for the one big process to get off the CPU
    - Ex. many I/O-bound processes and one CPU-bound process
    - Lower CPU and device utilization
    - Shorter processes go first
  - ► Nonpreemptive
    - troublesome for interactive systems

5.3 Scheduling Algorithms

- First-Come, First-Served (FCFS) Scheduling
  - ► Ex.

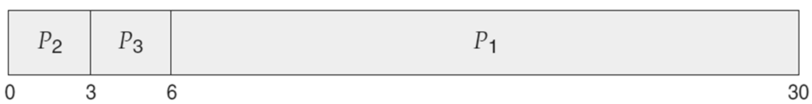
Process	Burst Time
P_1	24
P_2	3
P_3	3

average waiting time: 17



Process	Burst Time
P_2	3
P_3	3
P_1	24

average waiting time: 3

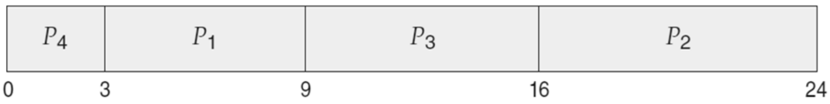


5.3 Scheduling Algorithms

- Shortest-Job-First (SJF) Scheduling
  - ► *shortest-next-CPU-burst*
  - ► Dispatch the process with the smallest next CPU burst; FCFS for tie break
  - ► gives the minimum average waiting time
  - ► Nonpreemptive or preemptive (shortest-remaining-time-first: SRTF)

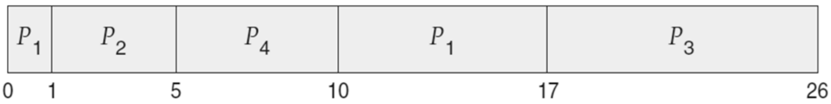
Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

average waiting time(SJF): 7
average waiting time(FCFS): 10.25



Process	Arrival Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

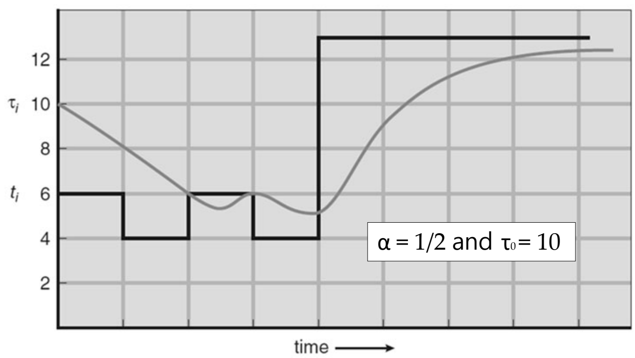
average waiting time(SJF): 7.75
average waiting time(SRTF): 6.5



Soongsil Univ. Digital System Design Lab.

5.3 Scheduling Algorithms

- Shortest-Job-First (SJF) Scheduling
  - ► Implementation problem: no way to know the length of the next CPU burst
    - ▪ approximate SJF scheduling: the next CPU burst ≈ the previous ones
    - ▪ exponential average of the measured lengths of previous CPU bursts
      - – $\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$
      - – τ_{n+1}: predicted value for the next CPU burst
      - – t_n: length of the n[th] CPU burst. most recent i
      - – α: $0 \le \alpha \le 1$
      - – τ_n: stores the past history√



CPU burst (t_i)		6	4	6	4	13	13	13	…
"guess" (τ_i)	10	8	6	6	5	9	11	12	…

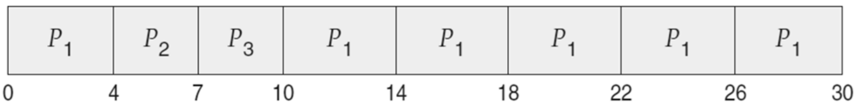
Soongsil Univ. Digital System Design Lab.

5.3 Scheduling Algorithms

- Round-Robin Scheduling
  - ▶Preemptive: to enable the system to switch between processes
  - ▶Time quantum (or time slice): generally, 10 ~ 100 ms.
  - ▶Ready queue: FIFO with circular operation.
  - ▶The CPU scheduler allocates the CPU for up to 1 time quantum using a timer
    - CPU burst < 1 TQ: the process releases CPU (T or W)
    - CPU burst > 1 TQ: context switch by interrupt (timer) (R)

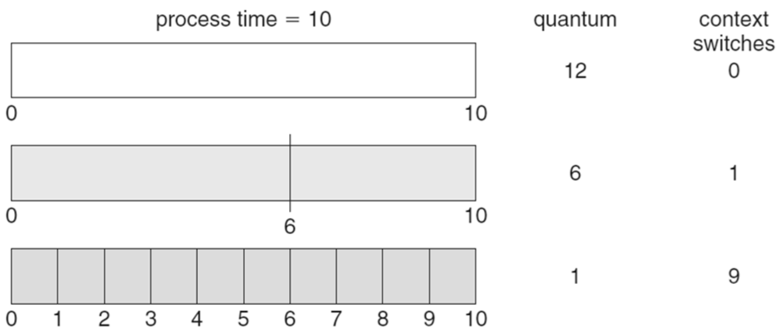
Process	Burst Time
P_1	24
P_2	3
P_3	3

average waiting time: 5.66
small average response time



5.3 Scheduling Algorithms

- Round-Robin Scheduling
  - ▶The size of the time quantum
    - Not too small, not too large: rule of thumb: 80% of CPU bursts < TQ
    - Context switching time: typically, < 10us (< 0.1% of TQ)



- Turnaround time
  - Ex. 3 processes, T=10, TQ=1: average turnaround time=29
  - TQ=10: average turnaround time=20 (FCFS)

5.3 Scheduling Algorithms

- Priority Scheduling
  - ►Priority: generally indicated by fixed range of numbers.
    - Assigned to each process
    - High: may be '0' or max. number
  - ►CPU is allocated to the process with the highest priority
    - FCFS: equal priorities for all processes
    - SJF: priority (p) is the inverse of the (predicted) next CPU burst
  - ►Ex.

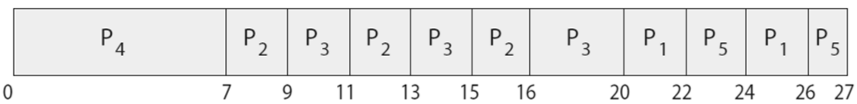
Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

average waiting time: 8.2

5.3 Scheduling Algorithms

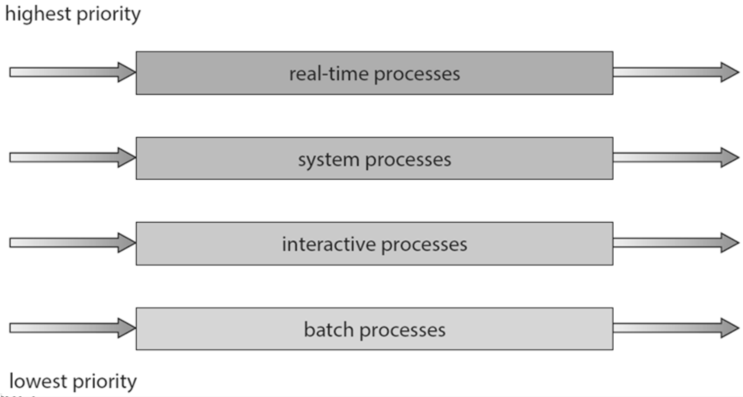
- Priority Scheduling
  - ►Preemptive or nonpreemptive
  - ►Starvation (indefinite blocking) of low priority process
    - Aging: gradually increasing the priority of long-waiting processes
  - ►To combine round-robin and priority scheduling
    - runs processes with the same priority using round-robin scheduling
    - Ex.

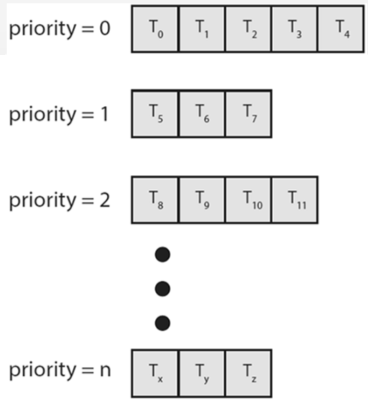
Process	Burst Time	Priority
P_1	4	3
P_2	5	2
P_3	8	2
P_4	7	1
P_5	3	3



5.3 Scheduling Algorithms

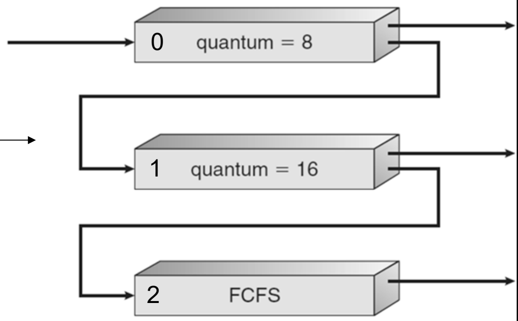
- Multilevel Queue Scheduling
  - ▶Multilevel scheduling using multiple queues
  - ▶Scheduling among the queues and within each queue
  - ▶Ex. Separate queues for each priority
    round-robin within a queue √
  - ▶separate queues based on the process type





5.3 Scheduling Algorithms

- Multilevel Feedback Queue Scheduling
  - ▶a process moves between queues
  - ▶The most general CPU-scheduling algorithm
    - most complex
  - ▶Ex. Long CPU-burst process to lower-priority queue
    long-waiting process to higher-priority queue
  - ▶EX.
  - ▶Parameters
    - # of queues
    - scheduling algorithm for each queue
    - when to upgrade to a higher priority queue
    - when to demote to a lower priority queue
    - starting queue



5.4 Thread Scheduling

- Scheduling issues involving user-level and kernel-level threads
- Contention Scope
  - ►Process-contention scope (PCS): scheduling by thread library for user level threads to an LWP
    - many-to-one or many-to-many
    - Typically, based on priority set by programmers
  - ►System-contention scope (SCS): scheduling by OS LWP's kernel thread onto a CPU core
- Pthread Scheduling
  - ►PTHREAD_SCOPE_PROCESS: PCS scheduling.
  - ►PTHREAD_SCOPE_SYSTEM: SCS scheduling.

5.4 Thread Scheduling

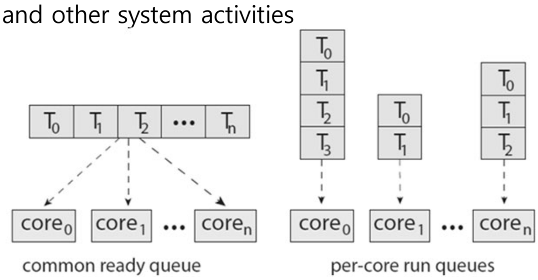
```c
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS 5
int main(int argc, char *argv[])
{
 int i, scope;
 pthread_t tid[NUM THREADS];
 pthread_attr_t attr;
/* get the default attributes */
 pthread_attr_init(&attr);
/* first inquire on the current scope */
 if (pthread_attr_getscope(&attr, &scope) != 0)
   fprintf(stderr, "Unable to get scheduling scope\n");
 else {
  if (scope == PTHREAD_SCOPE_PROCESS)
   printf("PTHREAD_SCOPE_PROCESS");
  else if (scope == PTHREAD_SCOPE_SYSTEM)
        printf("PTHREAD_SCOPE_SYSTEM");
       else
        fprintf(stderr, "Illegal scope value.\n");
 }
/* set the scheduling algorithm to PCS or SCS */
 pthread_attr_setscope(&attr, PTHREAD_SCOPE_SYSTEM);
/* create the threads */
 for (i = 0; i < NUM_THREADS; i++)
  pthread_create(&tid[i],&attr,runner,NULL);
/* now join on each thread */
  for (i = 0; i < NUM THREADS; i++)
   pthread_join(tid[i], NULL);
}

/* Each thread will begin control in this function */
void *runner(void *param)
{
/* do some work ... */
 pthread_exit(0);
}
```

- Linux and macOS systems allow only PTHREAD_SCOPE_SYSTEM

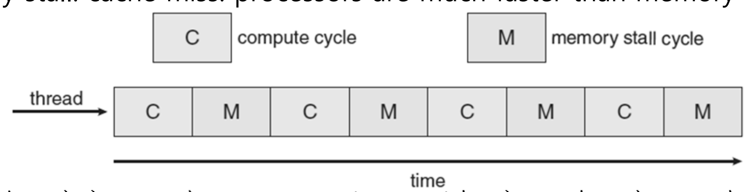
5.5 Multi-Processor Scheduling

- Multiprocessor: scheduling issues become more complex
  - ► Multicore CPUs
  - ► Multithreaded cores
  - ► NUMA systems
  - ► Heterogeneous multiprocessing
- Approaches to Multiple-Processor Scheduling
  - ► asymmetric multiprocessing
    - Master server: all scheduling decisions, I/O processing, and other system activities
    - Others: Execute users codes
    - potential bottleneck by the master server
  - ► Symmetric multiprocessing (SMP)
    - each processor is self-scheduling.
    - possible strategies
      1. All threads may be in a common ready queue.
         - possible race condition
      2. Each processor may have its own private queue of threads.
         - load balancing algorithm

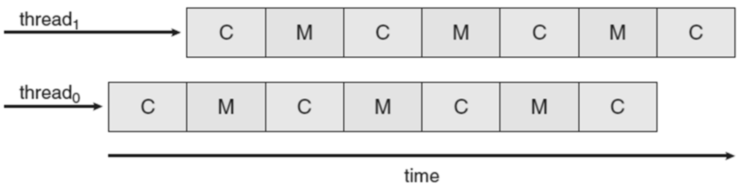


5.5 Multi-Processor Scheduling

- Multicore Processors
  - ► SMP systems with multicore processors: faster and consuming less power
  - ► memory stall: cache miss: processors are much faster than memory

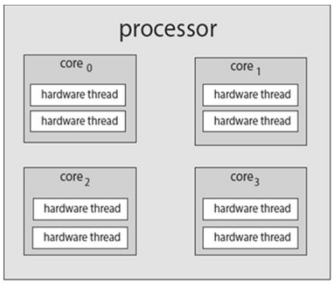


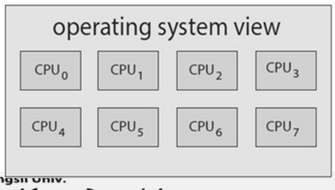
- multithreaded processing cores: two (or more) hardware threads to each core



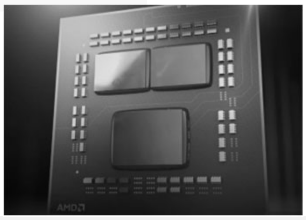
5.5 Multi-Processor Scheduling

- Multicore Processors
  - ►chip multithreading (CMT)
    - Intel: hyper-threading (aka simultaneous multithreading or SMT): 2 threads per core
    - Oracle Sparc M7 (8 cores): 8 threads per core → 64 logical cores



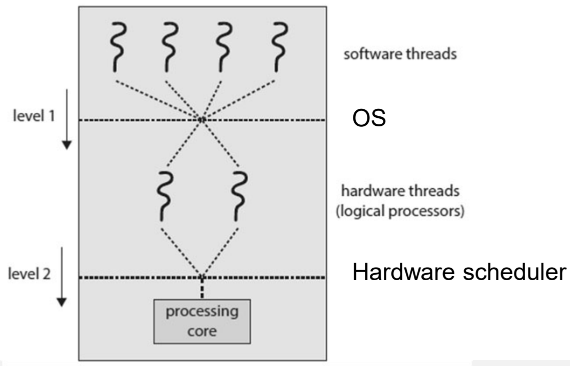


AMD Ryzen™ 9 5950X
CPU 코어 수: 16
스레드 수: 32
최대 부스트 클럭: 최대 4.9GHz
기본 클럭: 3.4GHz
총 L2 캐시: 8MB
총 L3 캐시: 64MB
기본 TDP/TDP: 105W
Processor Technology for CPU Cores: TSMC 7nm FinFET



5.5 Multi-Processor Scheduling

- Multicore Processors
  - ►Multithreading a processing core
    1. Coarse grained: execution until a long-latency event.
       - –Pipeline flushing → high cost of thread switching
    2. Fine-grained (interleaved): thread switching at the boundary of instruction cycle
       - –No pipeline flushing → small cost; extra logic required
    - Concurrent, not parallel:
      - –Holding multiple threads;
      - –sharing hardware for execution
    - Two levels of scheduling →
      - – sharing-aware algorithm
        - » Ex. 2 cores and 2 threads/core

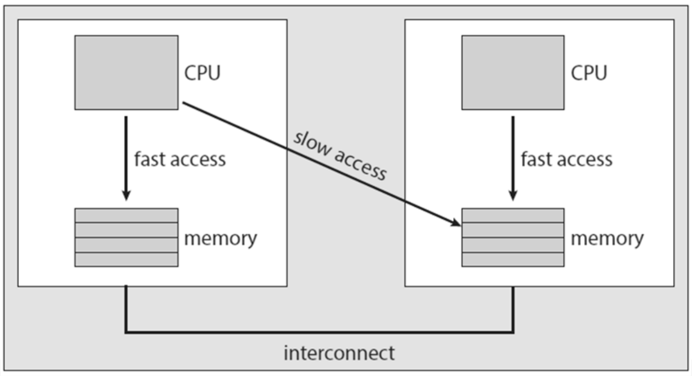


5.5 Multi-Processor Scheduling

- Load Balancing
  - ▶evenly distribute workload across all processors
  - ▶Approaches
    1. Push migration: periodical checking → moving (or pushing) threads if necessary
    2. Pull migration: an idle processor pulls a waiting task
    - not mutually exclusive: ex. Linux CFS scheduler, FreeBSD ULE scheduler
  - ▶Balanced load
    - approximately the same number of threads in all queues or
    - equal distribution of thread priorities across all queues

5.5 Multi-Processor Scheduling

- Processor Affinity
  - ▶to keep a thread running on the same processor and take advantage of a warm cache
  - ▶Easy for private, per-processor ready queues
  - ▶Processor affinity forms
    1. Soft affinity: process running on the same processor is not guaranteed
    2. Hard affinity: a process specifies a subset of processors using system calls
    - Linux: soft affinity and `sched_setaffinity()` system call for hard affinity
  - ▶non-uniform memory access (NUMA)
    - NUMA-aware algorithm
      - Process scheduling to near CPU
  - ▶Counteraction between load balancing and processor affinity



5.5 Multi-Processor Scheduling

- Heterogeneous Multiprocessing (HMP)
  - ARM's big.LITTLE architecture and Intel's hybrid technology (P/E core)
    - Qualcomm SD8Gen2: 1 x Cortex-X3 + 2 x Cortex-A715 + 2 x Cortex-A710 + 3 x Cortex-A510
    - Intel i9 13900k: 8P + 16E
  - Power management
  - Big/P cores: interactive tasks, high performance and short burst tasks
    - disabled in power-saving mode
  - LITTLE/E cores: low performance or long bursts task, background jobs
  - Windows 10 supports HMP scheduling
    - allowing a thread to select a scheduling policy that best supports its power management demands

Summary

- CPU scheduling
  - selecting a waiting process from the ready queue and allocating the CPU using the dispatcher.
- Scheduling algorithms:
  - either preemptive or nonpreemptive
  - Evaluating criteria
    - (1) CPU utilization, (2) throughput, (3) turnaround time, (4)waiting time, and (5) response time.
  - FCFS: simplest; not efficient for waiting time
  - SJF: optimal for WT; difficult in implementing
  - RR: preemption, time quantum, short response time
  - Priority: the highest priority first.
  - Multilevel queue: multiple queues by priority, different scheduling in each queue
  - Multilevel feedback queues: process migration between queues
- Multicore processors with multiple hardware threads (logical CPUs)
  - Load balancing and processor affinity

Exercises, problems and projects

- Exercises
  - 5.4, 5.6, 5.7, 5.8
- Problems
  - 5.13, 5.21

5.13 One technique for implementing lottery scheduling works by assigning processes lottery tickets, which are used for allocating CPU time. Whenever a scheduling decision has to be made, a lottery ticket is chosen at random, and the process holding that ticket gets the CPU. The BTV operating system implements lottery scheduling by holding a lottery 50 times each second, with each lottery winner getting 20 milliseconds of CPU time (20 milliseconds × 50 = 1 second). Describe how the BTV scheduler can ensure that higher-priority threads receive more attention from the CPU than lower-priority threads.

5.21 Consider a variant of the RR scheduling algorithm in which the entries in the ready queue are pointers to the PCBs.

a. What would be the effect of putting two pointers to the same process in the ready queue?

b. What would be two major advantages and two disadvantages of this scheme?

c. How would you modify the basic RR algorithm to achieve the same effect without the duplicate pointers?