CSS225: Operating System
Midterm Mock Exam

curated by The Peanuts

Name. Nonprawich I. ...ID. 6622772422 ...Section.............Seat No.............

Conditions: Semi-Closed Book

Directions:

1. This exam has 33 pages (including this page).

2. No multitasking during taking the exam.

3. Write your username or process ID at the top.

4. Reading the problem is optional but highly recommended.

5. Solutions can be written in English or Thai.

6. Students may not escape through Windows or Linux.

Part A: Multiple-Choice Questions (15 marks)

1. Which of the following best describes the basic instruction cycle in the Von Neumann model?

a) Decode and execute the instruction only

b) Fetch the instruction from memory only

c) Fetch the instruction, then decode and execute it

d) Execute and store the result only

2. What is the primary purpose of hardware interrupts in a computer system?

a) To reduce the speed of the CPU

b) To increase CPU utilization by preventing idle time

c) To completely stop program execution when an error occurs

d) To allow the user to manually control I/O devices

3. In the storage hierarchy of a computer system, which of the following is correctly ordered from fastest to slowest?

a) Main memory, registers, cache, secondary storage

b) Registers, cache, main memory, secondary storage

c) Cache, registers, main memory, secondary storage

d) Secondary storage, main memory, cache, registers

4. What is the main advantage of Direct Memory Access (DMA) in a computer system?

a) It eliminates the need for secondary storage

b) It allows data transfer between I/O devices and memory without CPU intervention

c) It provides additional security for system programs

d) It increases the speed of the bootstrap program

5. What is the primary function of a bootstrap program in a computer system?

a) To manage file systems and memory allocation

b) To load the operating system kernel into main memory

c) To execute user applications

d) To handle hardware interrupts

6. From a system view perspective, what is the primary role of an operating system?

a) To provide a user interface

b) To run application programs

c) To manage hardware resources

d) To store data permanently

✳✳ 7. What is the difference between multiprogramming and multi-tasking?

a) Multiprogramming loads multiple processes into memory; multitasking executes them using time-sharing

b) Multiprogramming is for single-user systems; multitasking is for multi-user systems

c) Multiprogramming requires multiple CPUs; multitasking works with a single CPU

d) Multiprogramming is for batch systems; multitasking is only for real-time systems

8. In dual-mode operation, what happens when a user program needs to perform a privileged operation?

a) It can directly access hardware in user mode

b) It must terminate and restart in kernel mode

c) It issues a system call that triggers a mode switch to kernel mode

d) It must wait until the OS automatically switches to kernel mode

9. What causes a context switch in an operating system?

a) Only hardware interrupts

b) Only timer interrupts

c) Only software interrupts (traps)

d) Hardware interrupts, software interrupts, or timer interrupts

10. Which of the following is a characteristic of MS-DOS as a simple structured operating system?

a) It supports multitasking with multiple applications running simultaneously

b) It strictly separates user applications from hardware access

c) It allows only one program to run at a time (single-tasking)

d) It has a microkernel structure with most services in user space

11. What is a key disadvantage of a monolithic operating system structure?

a) System calls are executed too slowly

b) Updating one part may require reinstalling the whole OS

c) It cannot support modern applications

d) It uses too much disk space

12. In a layered operating system structure, what is the relationship between layers?

a) Each layer interacts with all other layers

b) Each layer interacts only with its neighboring layers

c) Each layer is completely independent of other layers

d) Each layer can only interact with hardware directly

13. What is a key feature of the microkernel operating system structure?

a) All OS services run in kernel mode for better performance

b) Only essential components stay in the kernel while other OS services run in user space

c) It does not support module loading after system boot

d) It has a purely layered design with strict hierarchical access

14. What are Loadable Kernel Modules (LKMs) used for in module-structured operating systems?

a) To provide a graphical user interface for the operating system

b) To connect the operating system with application programs

c) To allow dynamic loading and unloading of OS components as needed

d) To replace the bootstrap program with a more efficient one

15. Which of the following best describes modern operating systems like Windows and Linux?

a) They use a pure microkernel structure

b) They use a pure monolithic structure

c) They use a pure layered structure

d) They use hybrid structures combining elements from different approaches

16. What is the definition of a process in an operating system?

a) A program stored on a hard drive

b) An application program in execution by the CPU

c) A section of main memory

d) A hardware device controlled by the OS

17. In the memory structure of a process, which section contains the executable code?

a) Stack section

b) Heap section

c) Data section

d) Text section

18. What is the purpose of the "free space" in a process memory structure?

a) To store system calls

b) To reserve space for additional global variables

c) To allow heap expansion during execution

d) To store process control information

19. Which process state indicates that the process is waiting for CPU execution?

a) New

b) Ready

c) Running

d) Waiting

20. On a system with a single CPU core, which of the following statements is true?

a) Multiple processes can be in the running state simultaneously

b) Only one process can be in the running state, but multiple processes can be in the ready state

c) Only one process can exist in the system at any time

d) All processes must cycle through all possible states

21. What information is NOT stored in a Process Control Block (PCB)?

a) Process state

b) Process number (PID)

c) Program counter

d) The actual program code

22. What happens during a context switch?

a) The OS terminates one process and creates another

b) The OS saves the state of the running process to its PCB and loads the state of another process

c) The OS moves a process from the ready queue to a wait queue

d) The OS creates a duplicate copy of the current process

23. How many threads does a newly created process have by default?

a) Zero

b) One

c) Two

d) Depends on the operating system

24. What is the primary difference between a kernel process and a user process?

a) Kernel processes are faster than user processes

b) Kernel processes operate in kernel mode with full control over system resources, while user processes operate in user mode

c) Kernel processes can only communicate with hardware, not with user processes

d) User processes can directly access physical memory, while kernel processes cannot

25. Which queue contains processes that are waiting for events such as I/O completion?

a) Ready queue

b) Running queue

c) Wait queue

d) New process queue

26. What is the relationship between a parent process and a child process?

a) The parent process is stored inside the child process

b) The child process is stored inside the parent process

c) Both processes share the same memory space

d) Both processes have separate address spaces in memory

27. What happens to child processes in modern operating systems if their parent process terminates?

a) They continue running normally

b) They are terminated automatically

c) They become orphaned processes with a new parent

d) They become zombie processes

28. When using the fork() system call, what value is returned to the child process?

a) -1

b) 0

c) The PID of the parent process

d) The PID of the child process

29. After a successful fork() operation, what happens to the execution flow?

a) The parent process continues execution, while the child process waits

b) The child process continues execution, while the parent process waits

c) Both parent and child processes continue execution at the instruction following the fork

d) Both parent and child processes restart execution from the beginning of the program

30. Which of the following statements is true about data in parent and child processes after a fork?

a) Data is shared between parent and child processes

b) Data is copied, not shared, giving each process its own independent copy

c) Only the stack is copied, but heap is shared

d) Only global variables are copied, local variables are shared

31. What is a thread in an operating system?

a) A program stored on a hard drive

b) The smallest entity executed by a CPU core

c) A separate address space in memory

d) A type of process that runs in kernel mode

32. Which of the following is NOT an element that a thread has of its own?

a) Program counter

b) Stack

c) Heap

d) Registers

33. What is the relationship between a process and threads?

a) A process can contain multiple threads, but a thread can only belong to one process

b) A thread can contain multiple processes, but a process can only belong to one thread

c) A process and a thread are different terms for the same concept

d) A process runs in user mode, while threads always run in kernel mode

34. What happens when a process is initially created?

a) It has zero threads until the programmer adds them

b) It automatically has one thread called the main thread

c) It automatically creates multiple worker threads

d) It has exactly two threads: a main thread and a worker thread

35. Which of the following is shared among all threads in a process?

a) Stack

b) Program counter

c) Registers

d) Code section

36. What happens if one thread in a multi-threaded process modifies a global variable?

a) Only that thread sees the change

b) The change affects all threads in the process

c) The change only affects the main thread

d) The operating system blocks the modification

37. Which statement about user threads and kernel threads is correct?

a) User threads exist in user space and are created by applications

b) User threads exist in kernel space and are created by the OS

c) Kernel threads exist in user space and are created by applications

d) Both user threads and kernel threads are types of hardware threads

38. On a single-core CPU system, how do multiple threads appear to run simultaneously?

a) They truly run in parallel through hardware support

b) They take turns running by rapidly switching among them

c) Each thread uses a separate virtual CPU

d) The operating system creates temporary additional cores

39. What is synchronous threading?

a) The main thread creates worker threads but does not wait for them to finish

b) The main thread waits for all worker threads to complete before continuing

c) All threads must start execution at exactly the same time

d) Threads that can only communicate with each other, not with processes

40. Consider the following Python code:

```python
import threading

def task():
    print("Worker thread task")

T1 = threading.Thread(target=task)
T1.start()
T1.join()
print("Main thread continues")
```

What does the join() method do in this code?

a) It combines T1 with the main thread

b) It makes the main thread wait until T1 finishes

c) It terminates T1 immediately

d) It assigns T1 to the same CPU core as the main thread

41. Which of the following best describes a hardware thread?

a) A thread created by a hardware device driver

b) A virtual/logical CPU core provided by the processor

c) A thread that runs exclusively on one physical CPU core

d) A thread that can only access hardware directly

42. If a multi-threaded process has three threads (one main thread and two worker threads), how many stacks will exist in the process's address space?

a) One stack shared by all threads

b) Two stacks (one for the main thread, one shared by worker threads)

c) Three stacks (one for each thread)

d) Four stacks (one for the process and one for each thread)

43. When does the CPU scheduler (short-term scheduler) select a new process to run?

a) Only when a process terminates

b) Only when a process requests I/O

c) Only when the CPU becomes idle

d) All of the above

44. Which of the following scenarios represents non-preemptive scheduling?

a) A process with higher priority arrives and takes the CPU from the current process

b) A process completes its time quantum and is moved to the back of the ready queue

c) A process terminates and releases the CPU voluntarily

d) A process is interrupted by a timer

45. What is the main difference between preemptive and non-preemptive scheduling?

a) In preemptive scheduling, the CPU can be taken away from a running process

b) In preemptive scheduling, processes have higher priority

c) In non-preemptive scheduling, processes execute faster

d) In non-preemptive scheduling, there is no ready queue

46. What does the dispatcher module in an operating system do?

a) Selects which process from the ready queue will be executed next

b) Loads processes from disk into main memory

c) Performs context switching and gives control of the CPU to the selected process

d) Creates new processes when requested by user applications

⁎⁎ 47. What is dispatch latency?

a) The time it takes to move a process from disk to memory

b) The time it takes to stop one process and start another

c) The time a process spends in the ready queue

d) The time between process creation and termination

48. Which scheduling metric represents the total time from process submission to completion?

a) Waiting time

b) Response time

c) Throughput

d) Turnaround time

49. What scheduling algorithm assigns the CPU to the process that has the shortest expected processing time?

a) First-Come, First-Served (FCFS)

b) Round Robin (RR)

c) Shortest Job First (SJF)

d) Priority Scheduling

50. Which scheduling algorithm can lead to the starvation problem? ~~అ~~ ఇ ఆ'e priority

a) First-Come, First-Served (FCFS)

b) Round Robin (RR)   *this is possible*

c) Shortest Job First (SJF)

d) All of the above

51. What technique can be used to prevent starvation in priority scheduling?

a) Using larger time quantum

b) Implementing multilevel queues

c) Using aging to gradually increase the priority of waiting processes

d) Converting to non-preemptive scheduling

52. If the time quantum in Round Robin scheduling is set very small, what undesirable effect occurs?

a) Processes may starve

b) The algorithm effectively becomes FCFS

c) Too many context switches occur, wasting CPU time

d) The algorithm effectively becomes SJF

53. If the time quantum in Round Robin scheduling is set extremely large, what does the algorithm become equivalent to?

a) First-Come, First-Served (FCFS)

b) Shortest Job First (SJF)

c) Priority Scheduling

d) Multilevel Queue Scheduling

54. In Multilevel Queue Scheduling, what determines which queue a process is placed in?

a) The process's remaining CPU burst time

b) The process's previous CPU usage

c) The process's properties like memory size, priority, or type

d) Random assignment to balance the load

55. What is the key difference between Multilevel Queue Scheduling and Multilevel Feedback Queue Scheduling?

a) Multilevel Feedback Queue allows processes to move between queues

b) Multilevel Feedback Queue uses only one scheduling algorithm for all queues

c) Multilevel Queue supports more priority levels

d) Multilevel Queue allows processes to move between queues

56. Consider these processes with their burst times and arrival times:

Process	Burst Time	Arrival Time
P1	~~10~~ 9	0
P2	~~5~~ 4	1
P3	~~3~~ 0	2

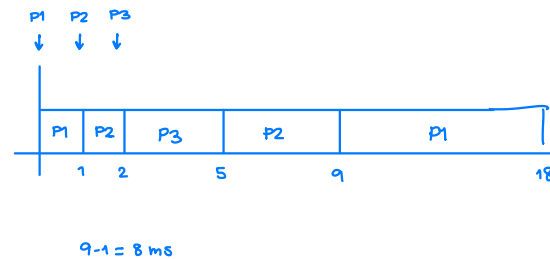Using the preemptive Shortest Job First algorithm, what is the waiting time for process P1?

a) 0 ms

b) 8 ms

c) 9 ms

d) 10 ms

P1 P2 P3
↓   ↓   ↓

| P1 | P2 | P3 | P2 | P1 |
  1   2   5   9   18

9-1 = 8 ms

57. Consider these processes with their burst times and arrival times:

Process	Burst Time	Arrival Time
P1	6̷2̷0	0
P2	3̷0	0
P3	8̷4	0

Using Round Robin scheduling with a time quantum of 4, what is the turnaround time for process P3?

$P_3, P_{2,9} P_1$ ↓

a) 8 ms

b) 13 ms

c) 15 ms

d) 17 ms

17 - 0 = 17 ms #
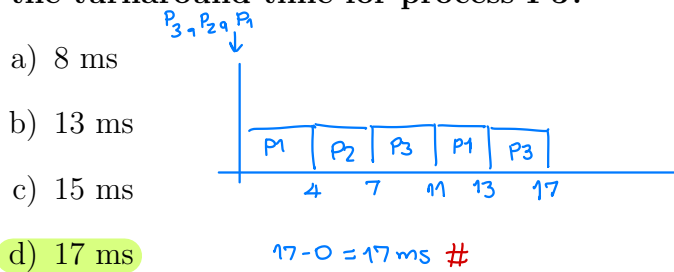
58. What is a race condition in the context of operating systems?

a) When two processes compete to use the CPU

b) When the outcome of execution depends on the particular order in which instructions are executed

c) When one process runs faster than another process

d) When a process tries to access a terminated process's memory

59. What is the critical section in a program?

a) The section of code that performs I/O operations

b) The section of code that only one process can execute at a time

c) The section of code that accesses shared resources among processes

d) The section of code that contains the main algorithm of a program

60. Which of the following is NOT an approach to prevent concurrency issues?

a) Mutex locks

b) Atomic instructions

c) Semaphores

d) Multi-core processing with increased CPU speed

61. What is a spin lock in the context of process synchronization?

a) A lock that automatically releases after a fixed time

b) A situation where a process continuously checks a condition in a loop, waiting for it to change

c) A lock that can only be used by the kernel

d) A lock that prevents multiple processes from accessing any part of memory

62. What are the two main functions used in mutex lock implementation?

a) lock() and unlock()

b) wait() and signal()

c) acquire() and release()

d) enter() and exit()

63. What is the main disadvantage of using spin locks?

a) They consume memory

b) They waste CPU time

c) They can only be used with single-core processors

d) They require hardware support

64. Which of the following correctly describes a semaphore?

a) A synchronization method that allows only one process in the critical section

b) A synchronization method that uses a boolean variable to control access

c) A synchronization method that can allow multiple processes in the critical section

d) A hardware instruction that cannot be interrupted

65. What are the two atomic operations used with classic semaphores?

a) lock() and unlock()

b) acquire() and release()

c) wait(S) and signal(S)

d) enter() and exit()

66. Consider a classic semaphore with an initial value S = 5. After several wait() and signal() operations, S = 2. How many processes are currently in the critical section?

a) 2 processes

b) 3 processes

c) 5 processes

d) Cannot be determined from the information given

67. What is the key difference between classic semaphores and revised semaphores?

a) Classic semaphores use busy waiting while revised semaphores block waiting processes

b) Classic semaphores are faster but revised semaphores are more accurate

c) Classic semaphores work with single-core systems while revised semaphores work with multi-core systems

d) Classic semaphores use integers while revised semaphores use boolean values

68. In the context of Python programming, what does the following code create?

```
lock = multiprocessing.Lock()
```

a) A semaphore with initial value 1

b) A mutex lock

c) A critical section

d) A new process

69. Consider a revised semaphore with initial value S = 8. After executing several processes, S = -4. How many processes are waiting in the semaphore's list?

a) 4 processes

b) 8 processes

c) 12 processes

d) 16 processes

70. What problem does the following Python code address by using a lock?

```python
def TaskAdd(A, lock):
    for _ in range(A):
        lock.acquire()
        Num.value = Num.value + 1
        lock.release()
```

a) Deadlock

b) Race condition

c) Resource starvation

d) Priority inversion

Part B: Short Answer Questions (60 marks)

1. Consider the python code and the tree of processes as shown below. Answer the following questions. (Assume that there is no error running this program.)

```
rc1 = os.fork()
if (rc1 > 0):
    print("PID: %d and rc = %d\n" %(os.getpid(), rc1))  # Line A
elif (rc1 == 0):
    print("PID: %d and rc = %d\n" %(os.getpid(), rc1))  # Line B
    rc2 = os.fork()
    if (rc2 > 0):
        print("PID: %d and rc = %d\n" %(os.getpid(), rc2))  # Line C
        rc3 = os.fork()
        if (rc3 > 0):
            print("PID: %d and rc = %d\n" %(os.getpid(), rc3))  # Line D
        elif (rc3 == 0):
            print("PID: %d and rc = %d\n" %(os.getpid(), rc3))  # Line E
    elif (rc2 == 0):
        print("PID: %d and rc = %d\n" %(os.getpid(), rc2))  # Line F
```

(Handwritten annotations: rc1 = os.fork() → 187, 145; 145, 187. rc2 = os.fork() → 187, 229. rc3 = os.fork() → 187, 230. Brace labels: 145, 187, 187, 229, 230.)

Tree of Processes

```
    145
     |
    187
   /    \
 230    229
```
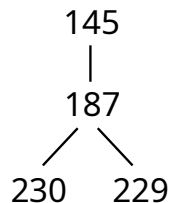
a) Write down the output according to Line A.

PID: 145 and rc = 187   #

b) Write down the output according to Line B.

PID: 187 and rc = 0   #

c) Write down the output according to Line C.

PID: 187 and rc = 229   #

d) Write down the output according to Line D.

PID: 187 and rc = 230   #

e) Write down the output according to Line E.

PID: 230 and rc = 0   #

f) Write down the output according to Line F.

PID: 229 and rc = 0   #

2. Consider the following Python code. Assume that there is no error. How many processes and threads will be created in total? Show your work by tracing through the execution and identifying each process and thread creation point.

```python
def ThreadTask():
    print("Worker thread running")

def ProcessTask():
    t1 = threading.Thread(target=ThreadTask)
    return

a = os.fork()
if (a > 0):
    b = os.fork()
    if (b > 0):
        t1 = threading.Thread(target=ThreadTask)
        t2 = threading.Thread(target=ThreadTask)
    else:
        c = os.fork()
        if (c == 0):
            p1 = multiprocessing.Process(target=ProcessTask)
        # Other Python commands are omitted.
elif (a == 0):
    d = os.fork()
    if (d > 0):
        t3 = threading.Thread(target=ThreadTask)
    else:
        t4 = threading.Thread(target=ThreadTask)
        t5 = threading.Thread(target=ThreadTask)
    # Other Python commands are omitted.
```

```
                        ┌──────┐
                        │  1M  │
                        └──────┘
                            │
                            ▼
                      A = os.fork()
            A>0      ╱              ╲      A=0
                   ╱                  ╲
                  ▼                    ▼
            ┌──────┐              ┌──────┐
            │  1M  │              │  1M  │
            └──────┘              └──────┘
                │                     │
                ▼                     ▼
          B = os.fork()         D = os.fork()
       B>0  ╱      ╲  B=0     D>0  ╱      ╲  D=0
           ▼        ▼             ▼        ▼
      ┌──────┐  ┌──────┐     ┌──────┐  ┌──────┐
      │  1M  │  │  1M  │     │  1M  │  │  1M  │
      └──────┘  └──────┘     └──────┘  └──────┘
          │         │            │         │
          ▼         ▼            ▼         ▼
 threading.Thread×2  C = os.fork()  threading.Thread  threading.Thread×2
                 C>0 ╱   ╲ C=0
     ┌──────┐       ▼     ▼      ┌──────┐       ┌──────┐
     │  1M  │   ┌──────┐ ┌──────┐│  1M  │       │  1M  │
     │  2W  │   │  1M  │ │  1M  ││  1W  │       │  2W  │
     └──────┘   └──────┘ └──────┘└──────┘       └──────┘
                            │
                            ▼
                         process
                       ╱        ╲
                      ▼          ▼
                 ┌──────┐    ┌──────┐
                 │  1M  │    │  1M  │
                 └──────┘    └──────┘
                      │          │
                      ▼          ▼
          threading.Thread  threading.Thread
                      │          │
                      ▼          ▼
                 ┌──────┐    ┌──────┐
                 │  1M  │    │  1M  │
                 │  1W  │    │  1W  │
                 └──────┘    └──────┘
```
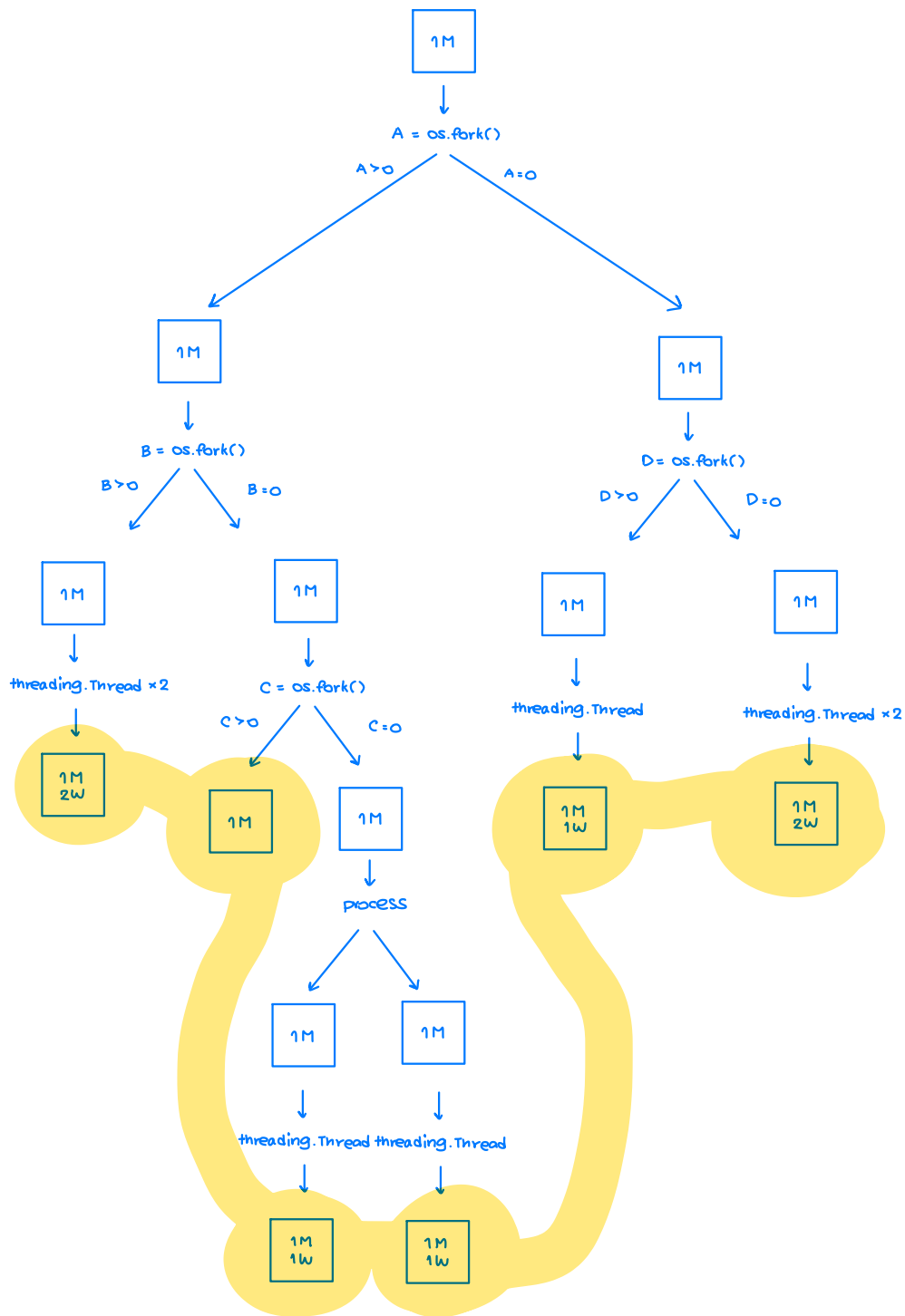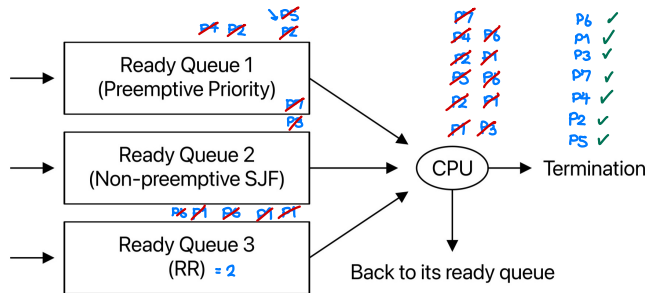
∴ 6 processes   #

   6 main threads

   7 worker threads

3. Consider a multiple-level queue scheduling (using only one CPU) with **3** queues as shown below. The priorities of these queues from the highest to the lowest are arranged in this order: Ready Queue 1, Ready Queue 2, and Ready Queue 3. The Ready Queue 1 applies the priority scheduling with preemption (higher priority numbers indicate higher priority). The Ready Queue 2 applies the shortest job first (SJF) non-preemptive scheduling. The Ready Queue 3 applies the round robin (RR) scheduling with the quantum time equal to **2** milliseconds.
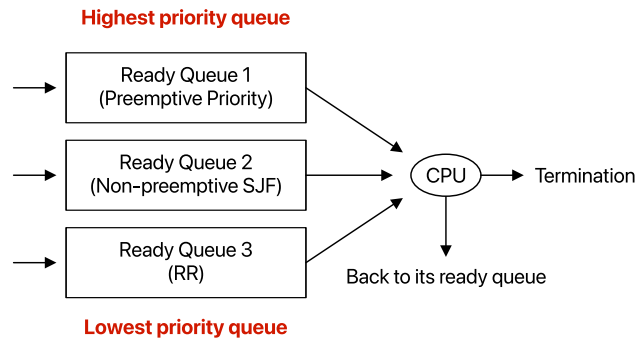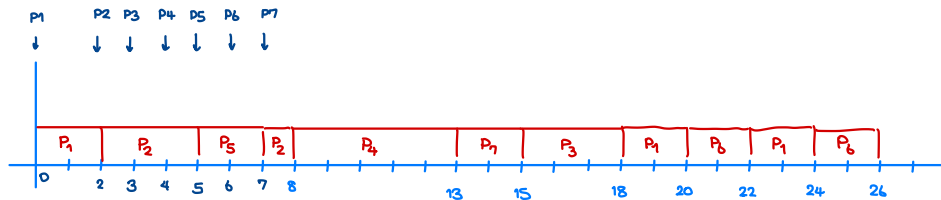

Suppose that the following processes arrive for execution (arrival time) at the times indicated and to these queues as specified below. Each process will run for the amount of time (one CPU burst) listed. Assume that the unit of time is **millisecond**.

Process	Entering queue	Burst time	Arrival time	Priority
P1	Ready Queue 3	6 4 2 0	0	-
P2	Ready Queue 1	4 1 0	2	3
P3	Ready Queue 2	3 0	3	-
P4	Ready Queue 1	5 0	4	1
P5	Ready Queue 1	2 0	5	4
P6	Ready Queue 3	4 2 0	6	-
P7	Ready Queue 2	2 0	7	-

Ready Queue 1
(Preemptive Priority)

Ready Queue 2
(Non-preemptive SJF)

Ready Queue 3
(RR) = 2

P4 P2 ↘P5
       ↗

       P7
       P6
       P8

CPU → Termination

P7 ✓
P4 P6
P2 P1
P5 P6
P2 P1
P1 P3

P6 ✓
P1 ✓
P3 ✓
P7 ✓
P4 ✓
P2 ✓
P5 ✓

P6 P1 P6 P1 P7

Back to its ready queue

Note: For Ready Queue 1, <u>higher priority number means higher priority</u>. When processes have the same priority, use FCFS.

a) Draw a Gantt chart to illustrate this multiple-level queue scheduling.

P1   P2 P3 P4 P5 P6 P7
↓   ↓ ↓   ↓ ↓   ↓   ↓

| P1 | P2 | P5 | P2 | P4 | P7 | P3 | P1 | P6 | P1 | P6 |

0   2   3 4 5 6 7 8   13   15   18   20   22   24   26

b) Calculate the average waiting time and average turnaround time for all processes.

$W_1 = 0 + (18-2) + (22-20) = 18$ ms

$W_2 = 0 + (7-5) = 2$ ms

$W_3 = (15-3) = 12$ ms

$W_4 = (8-4) = 4$ ms

$W_5 = 0$ ms

$W_6 = (20-6) + (24-22) = 16$ ms

$W_7 = (13-7) = 6$ ms

$W_{Avg} = \dfrac{18 + 2 + 12 + 4 + 0 + 16 + 6}{7}$

$= \dfrac{58}{7}$

$= 8.28$ ms #

$T_1 = 24 - 0 = 24$ ms

$T_2 = 8 - 2 = 6$ ms

$T_3 = 18 - 3 = 15$ ms

$T_4 = 13 - 4 = 9$ ms

$T_5 = 7 - 5 = 2$ ms

$T_6 = 26 - 6 = 20$ ms

$T_7 = 15 - 7 = 8$ ms

$T_{Avg} = \dfrac{24 + 6 + 15 + 9 + 2 + 20 + 8}{7}$

$= \dfrac{84}{7}$

$= 12$ ms #

4. Consider 60 processes sharing 5 critical sections. Let us call these critical sections as Critical Sections 1 to 5. Assume that, on each process, we write and use the classic semaphores and mutex locks to enclose these critical sections in this way:

```
wait(P);                          15
Critical Section 1;   12
wait(Q);              8
Critical Section 2;   5
signal(Q);
signal(P);
acquire(R);
Critical Section 3;
wait(S);              12  7
Critical Section 4;   → 3
signal(S);
release(R);
wait(T);              20
Critical Section 5;
signal(T);
```

Initially, we set the following variables as follows: $P = 15$, $Q = 8$, $S = 12$, and $T = 20$. From the given pseudocode above, answer the following 5 sub-questions. Briefly explain to get the full score; otherwise, your score will be deducted.

1. What is the maximum number of processes that can be simultaneously in Critical Section 3? Why?

   Because it is a mutex-lock, only one can be inside at a time. #

2. If Critical Section 3 currently has the maximum possible number of processes, what is the maximum number of processes that can be in Critical Section 4? Why?

12, Because it is a semaphore, with the value of 12   #

3. Assume there are currently 5 processes in Critical Section 2. What is the maximum number of processes that can be in Critical Section 1 at this time? Why?

15-3 = 12 processes   #

4. After the system has been running for some time, the current value of S is 7. There are 3 processes in Critical Section 4. What is the maximum number of processes that could be in Critical Section 3 at this moment? Why?

Still only one! Because it's a mutex-lock! #

5. What is the maximum number of processes that can be simultaneously in Critical Section 5? Why?

20 processes # เลือกตามได้แบบสัง