## 操作系统

## 第四次作业

**调度**

1 Why is it important for the scheduler to distinguish I/O-bound programs from CPU-bound programs?

**That’s because I/O-bound programs are different from CPU-bound programs in the use of CPU. A I/O-bound program often has lots of short CPU executions with several I/O waiting periods, but a CPU-bound program often has some long CPU executions. The CPU scheduler must distinguish I/O-bound programs from CPU-bound programs to make better use of CPU resources and improve operating efficiency.**

**这是因为I/O密集型程序与CPU密集型程序在CPU的使用上有所不同。I/O密集型程序通常具有大量短CPU执行，并有多个I/O等待，但CPU密集型程序通常有仅有少量长CPU执行。CPU调度程序必须区分I/O密集型程序和CPU密集型程序，以更好地利用CPU资源、提高运行效率。**

2 Consider the following set of processes, with the length of the CPU burst given in milliseconds:

|  |  |  |
| --- | --- | --- |
| Process | Burst Time | Priority |
| P1 | 10 | 3 |
| P2 | 1 | 1 |
| P3 | 2 | 3 |
| P4 | 1 | 4 |
| P5 | 5 | 2 |

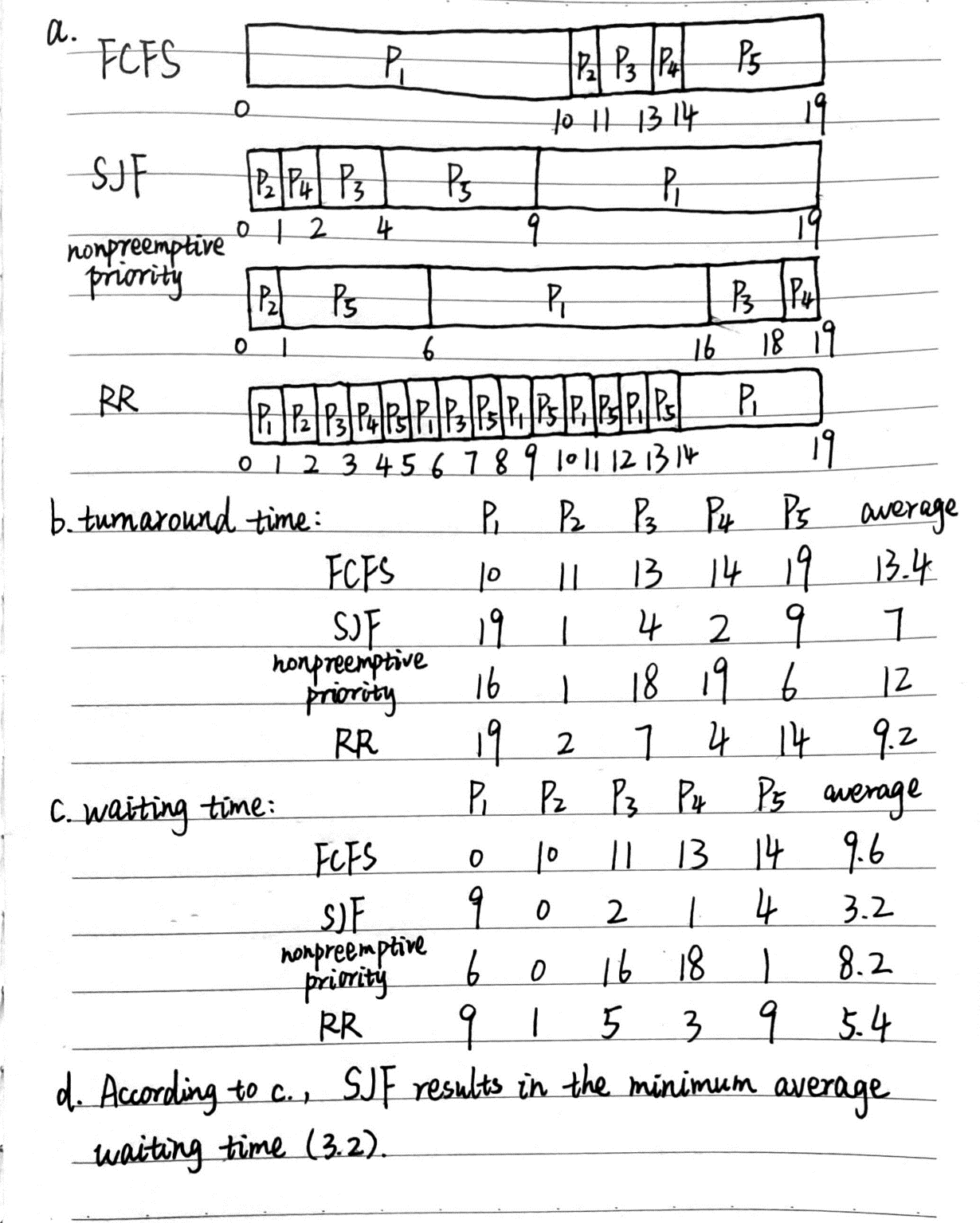
The processes are assumed to have arrived in the order P1, P2, P3, P4, P5, all at time 0.

a. Draw four Gantt charts that illustrate the execution of these processes using the following scheduling algorithms: FCFS, SJF, nonpreemptive priority (a smaller priority number implies a higher priority), and RR (quantum = 1).

b. What is the turnaround time of each process for each of the scheduling algorithms in part a?

c. What is the waiting time of each process for each of the scheduling algorithms in part a?

d. Which of the algorithms in part a results in the minimum average waiting time (over all processes)?



3 Which of the following scheduling algorithms could result in starvation?

a. First-come, first-served

b. Shortest job first

c. Round robin

d. Priority

**bd**

**SJF and priority scheduling may result in starvation, which means some progresses may wait for CPU forever. FCFS and RR couldn’t cause starvation.**

4 The traditional UNIX scheduler enforces an inverse relationship between priority numbers and priorities: The higher the number, the lower the priority. The scheduler recalculates process priorities once per second using the following function:

Priority = (Recent CPU usage / 2) + Base

where base = 60 and recent CPU usage refers to a value indicating how often a process has used the CPU since priorities were last recalculated.

Assume that recent CPU usage for process P1 is 40, process P2 is 18, and process P3 is 10. What will be the new priorities for these three processes when priorities are recalculated? Based on this information, does the traditional UNIX scheduler raise or lower the relative priority of a CPU-bound process?

**Priority of P1: 40/2+60=80 Priority of P2: 18/2+60=69**

**Priority of P3: 10/2+60=65**

**The traditional UNIX scheduler lowers the relative priority of a CPU-bound process, because a CPU-bound process often has longer CPU usage, which causes bigger priority number and lower priority.**