操作系统作业 3

PB19051183 吴承泽

1.

多线程的优点：

①响应性：采用多线程，即使出现部分阻塞或是执行冗长操作，它人可以继续执行，从而增加对用户的响应程度。

②资源共享更方便：允许一个应用程序在同一地址空间内有多个不同的活动线程

③经济性：由于线程能够共享它们所属进程的资源，所以创建和切换线程更佳经济

④具有可伸缩性：对于多处理器结构，多线程的优点更大，多线程可以再多处理核上并行运行。

选项：b、c

2.

a. 6

b. 8

3.

LINE C:CHILD: value = 5

LINE P:PARENT: value = 0

4.

普通管道和命名管道的区别：

①：对于 UNIX 和 Windows 系统，一旦进程完成通信并且终止，

普通管道不存在了，命名管道在通信进程完成后，

②：普通管道是单向的，命名管道是双向的

③：对普通管道而言，父进程创建一个管道，使用它来和子进程进行通信，对于命名管道而言父子关系不是必须的。

5.

①没有任何两个进程可以同时进入临界区

②每个进程执行速度不为零

③若没有进程在其临界区内执行且有进程需要进入临界区，那么只有那些不在剩余区内执行的进程可以参加选择，以便确定下次谁能进入临界区。

④一个进程作出进入临界区的请求直到这个请求允许为止，其它进程允许进入临界区的次数具有上限

严格轮转模式可以满足所有要求：

首先，假定每个进程执行速度不为 0，设置一个共享的信息量时，在其中一个进程准备进入临界区时，本进程不满足 while 循环的同时，其他进程能够满足不满足 while 循环跳出的条件，保证了互斥。又由于该进程若跳出了临界区，在跳出后立刻改变了共享信息量的值，可以使其它进程中的一个通过 while 循环，进入临界区，保证了进步的要求。又若一进程准备进入临界区时，其它进程轮完后总会有一次让该信息量使这个进程能够进入临界区并且临界区仅有该进程在跑。

6.

死锁是在等待队列中可能会出现一种这样的情况：两个或多个进程分别获取了一些信号量，导致他们都在等待对方释放该信号量的情况，这会产生无限等待的可能。

死锁的四个要求：

①互斥：有且仅有一个进程在同一时间可以使用该资源

②持有资源并等待：一个进程必须持有至少一项资源，并等待获取另一项资源，且该资源在被另一个进程持有

③不可抢占：资源释放仅可通过结束该进程来释放

④循环等待：进程中每一个进程都在等待另一个进程结束，每个进程的等待队列产生了闭环。

7.

信号量是一个整型变量，通过不可分割的增加减少来修改，并表征所空闲的资源的数量。

信号量的功能：

设该信号量为 S，当 S 不为 0 的时候，当一个进程需要用到该信号量表征的资源时，取出该信号量（即 S--），进入临界区，在临界区中的操作结束后，释放该信号量（即 S++）。当取出该信号量时发现 S==0，则阻塞自己，知道信号量 S 被其他进程返回回来时，再进入临界区，达到互斥的实现。

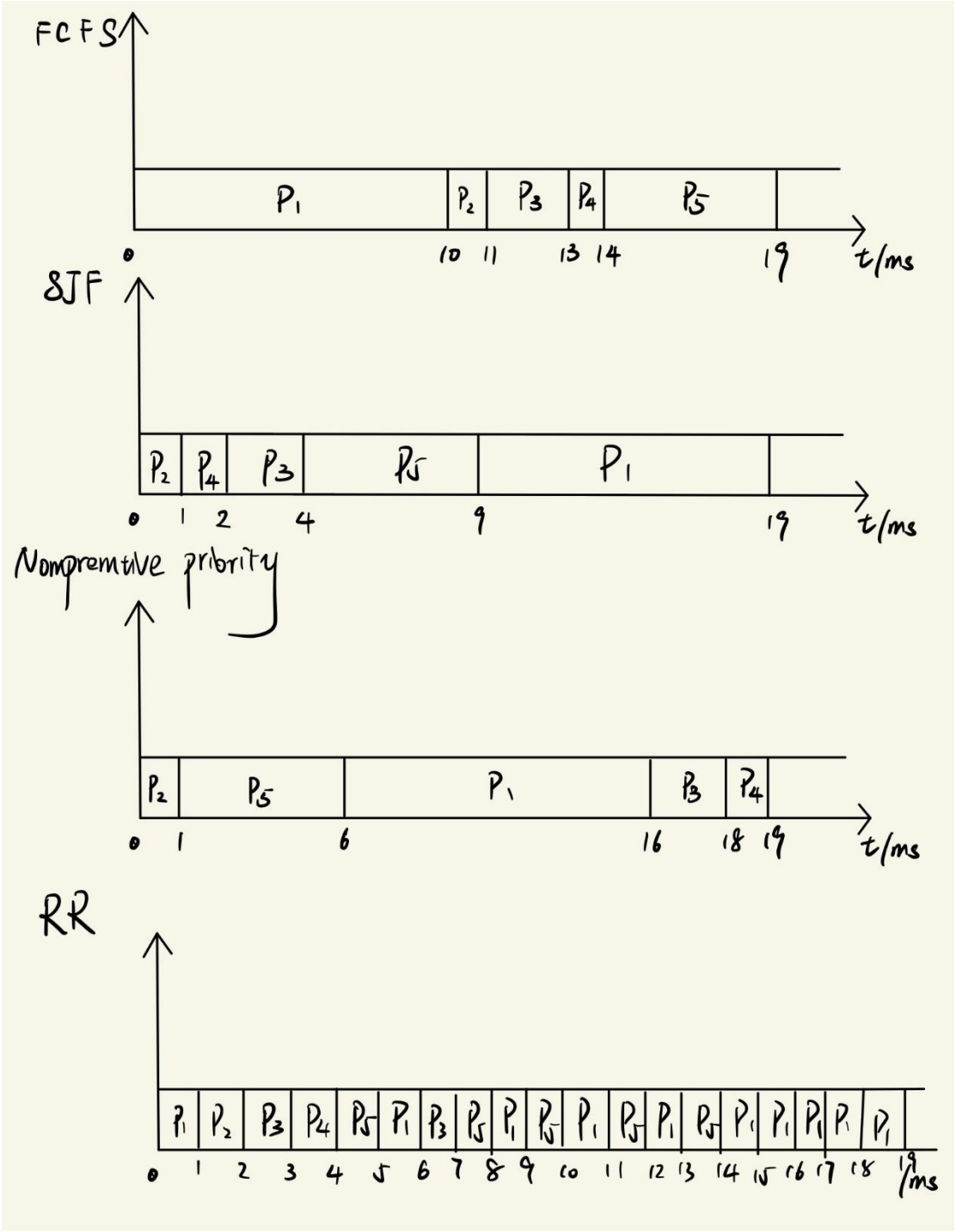
8.

代码如下

```
int chopsticks[5] = {1,1,1,1,1};//分别表示第 i 个哲学家左边的筷子

void * Dining_Philosophers_Problems(int i) //表示第 i 个哲学家的状态
{
    do
    {
        wait(chopsticks[i] , chopstick[(i + 1) % 5]);
        /*eat for awhile*/

        signal(chopsticks[i] , chopstick[(i + 1) % 5]);

        /*think for awhile*/
    }while(true);
}


void wait(int *chopsticks1, int *chopsticks2)//up
{
    while(*chopsticks1 == 0 || *chopsticks2 == 0)
    {
        /*think for awhile*/; //阻塞自己
    }
    *chopsticks1--;
    *chopsticks2--;
}

void signal(int *chopsticks1, int *chopsticks2)//down
{
    *chopsticks1++;
    *chopsticks2++;
}
```

9.

a)



b)

t_i 对应 Process i 的周转时间

FCFS:

t_1 = 10ms

t_2 = 11ms

t_3 = 14ms

t_4 = 14ms

t_5 = 19ms

SJF:

t_1 =19 ms

t_2 = 1ms

t_3 = 4ms

t_4 = 2ms

t_5 = 9ms

Nonpreemptive priority:

t_1 = 16ms

t_2 = 1ms

t_3 = 18ms

t_4 = 19ms

t_5 = 5ms

RR:

t_1 = 19ms

t_2 = 2ms

t_3 = 7ms

t_4 = 4ms

t_5 = 14ms

c)

t_i 对应 Process i 的等待时间

FCFS:

t_1 = 0ms

t_2 = 10ms

t_3 = 11ms

t_4 = 13ms

t_5 = 14ms

SJF:

t_1 = 9ms

t_2 = 0ms

t_3 = 2ms

t_4 = 1ms

t_5 = 4ms

Nonpreempitive priority:

t_1 = 6ms

t_2 = 0ms

t_3 = 16ms

t_4 = 18ms

t_5 = 1ms

RR

t_1 = 9ms

t_2 = 1ms

t_3 = 5ms

t_4 = 3ms

t_5 = 9ms

d)

设四种调度的平均时间分别为 T_F,T_S,T_N,T_R

T_F = (0 + 10 + 11 + 13 + 14)/5 = 9.6ms

T_S = (9 + 0 + 2 + 1 + 4)/5 = 3.2ms

T_N = (6 + 0 + 16 + 18 + 1)/5 = 8.2ms

T_R = (9 + 1 + 5 + 3 + 9)/5 = 5.6ms

经过比较，SJF 调度平均等待时间最短

10.

b,d

11.

比如说，当有两个进程时 p1 = 50ms,p2 = 80ms,t1=25ms,t2=35ms，由于 p1 周期较短，p1 优先级较高，因此在 0~25ms 中，进程 1 完成，在 25~50ms 中，进程 2 完成了前 25ms，此时进程 1 又进入就绪队列中，在 50~75ms 的时间中，CPU 执行进程 1，执行完进程 1 后，进程 2 还需执行 10ms 结束，但是第 80ms 为进程 2 的第二个周期，显然不够第一次的进程 2 结束自己的运行，因此该例子单调速率调度证明了有可能会错过截止期限。