Exercise 1

Considering process state transitions, which of the following migration is impossible?_____.

A. running → ready √ √

B. ready → running √ √

C. waiting → ready √ √

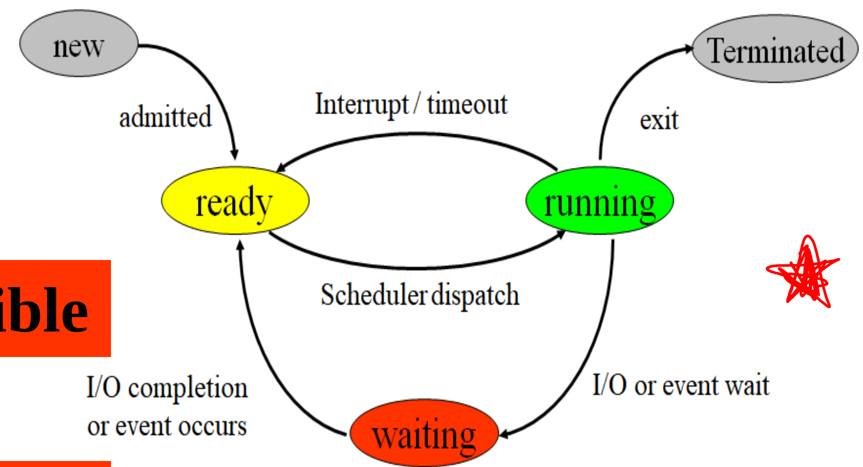
D. ready → waiting ? impossible

E. running → waiting √ √

E. waiting → running ? impossible

B. ready → terminated impossible

E. new → running impossible



term scheduler

Wensheng Li BUPT

Exercise 2

- There are 10 processes executing concurrently in a single-processor system.
  - The number of processes in running state: max:______1______ 1 min:______0______ 0
  - The number of processes in ready state: max:______9______ 9 min:______0______ 0
  - The number of processes in waiting state: max:______10______ 10 min:______0______ 0
- _CPU scheduler_ selects from among the processes that are ***ready*** to execute and allocates the CPU to one of them.
- The job / long term scheduler controls the degree of multiprogramming (the number of processes in memory).
- The process is swapped out, and is later swapped in, by the _medium term_ scheduler. medium-term scheduler

CPU scheduler CPU调度程序 Job scheduler

Wensheng Li BUPT

Exercise 3

mode

I am the parent process, my pid is …
统计结果是: 1

I am the child process, my pid is …
统计结果是: 1

```
#include <unistd.h>
#include <stdio.h>
int main ()
{
  pid_t fpid;   //fpid表示fork函数返回的值
  int count=0;
  fpid=fork();
  if (fpid < 0) { printf("error in fork!"); exit(-1);  }
  else if (fpid == 0) {
        printf("I am the child process, my pid is %d\n", getpid());
        count++;
       }
       else { printf("I am the parent process, my pid is %d\n", getpid());
              count++;
            }
  printf("统计结果是: %d\n", count);
  return 0;
}
```

wait(null);

Exercise 4 CPU Scheduler long term scheduler

- How many processes are created?

```c
#include <stdio.h>
#include <unistd.h>

int main()
{
    /* fork a child process */
    fork();

    /* fork another child process */
    fork();

    /* and fork another */
    fork();

    return 0;
}
```

```c
#include <stdio.h>
#include <unistd.h>

int main()
{
    int i;

    for (i = 0; i < 4; i++)
      fork();

    return 0;
}
```

16

8

Exercise 5

i	son/pa	ppid	pid	fpid
0	parent	2040	3224	3225
0	child	3224	3225	0
1	parent	2040	3224	3226
1	parent	3224	3225	3227
1	child	1	3227	0
1	child	1	3226	0

```
#include <unistd.h>
#include <stdio.h>
int main(void)
{
  int i=0;
  printf("i   son/pa   ppid   pid   fpid \n");
  for (i=0; i<2; i++)
 {
    pid_t  fpid=fork();
    if ( fpid==0 )
        printf("%d child %4d %4d %4d\n", i, getppid(), getpid(), fpid);
    else
        printf("%d parent %4d %4d %4d\n", i, getppid(), getpid(), fpid);
    }
  return 0;
}
```

ppid 当前进程的父进程的 **pid**
pid 当前进程的 **pid**
fpid **fork返回给当前进程的值**

Exercise 6

m-n n n

假设系统中有m个并发进程, m>n

假设 m<n

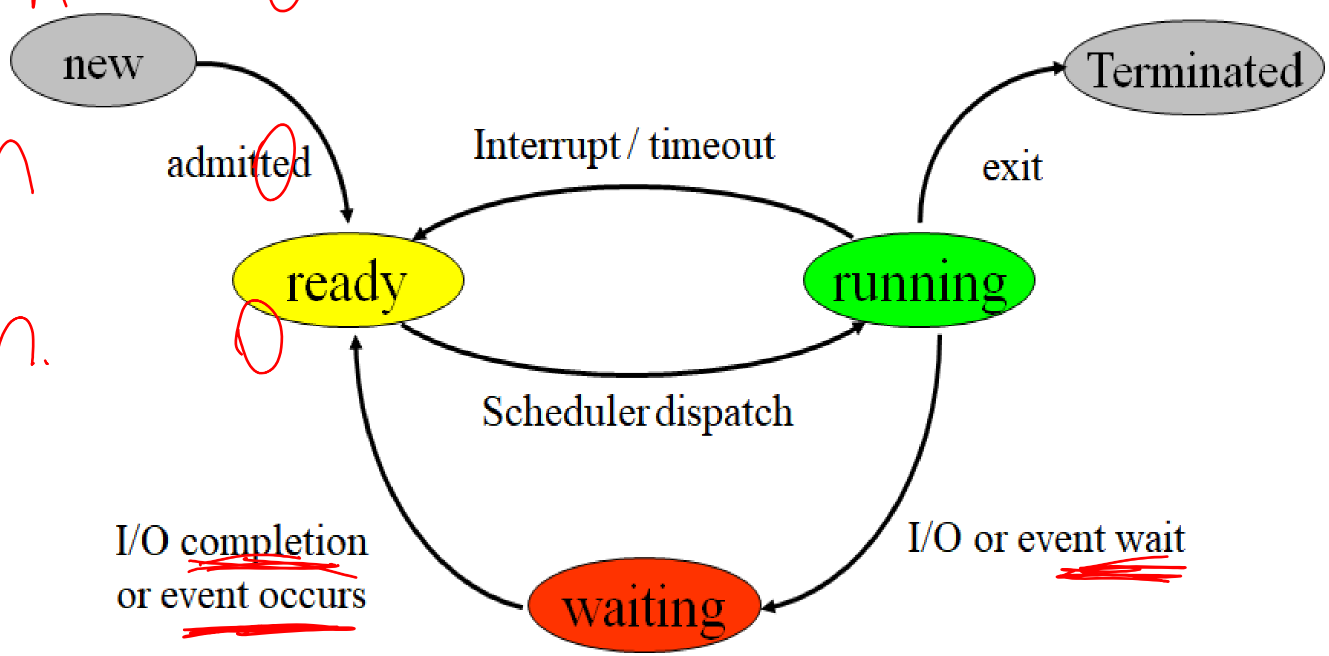
On a system with n processors,

1. what is the maximum number of processes that can be in the ready, running, and waiting states? m-n, n, m 0, m, m
2. what is the minimum number of processes that can be in the ready, running, and waiting states? 0, 0, 0 0, 0, 0

ready m-n 0

running n

waiting m. 0



Wensheng Li BUPT

Exercise 7

- **For each of the following transitions between process states, indicate whether the transition is possible.**
  If it is possible, give an example of one thing that would cause it.

(1) Running → ready √ Time out 超时、

(2) Running → waiting √ System call – open a file

(3) Running → swapped waiting

(4) Waiting → running ×

(5) Running → terminated √ exit()

Exercise 8

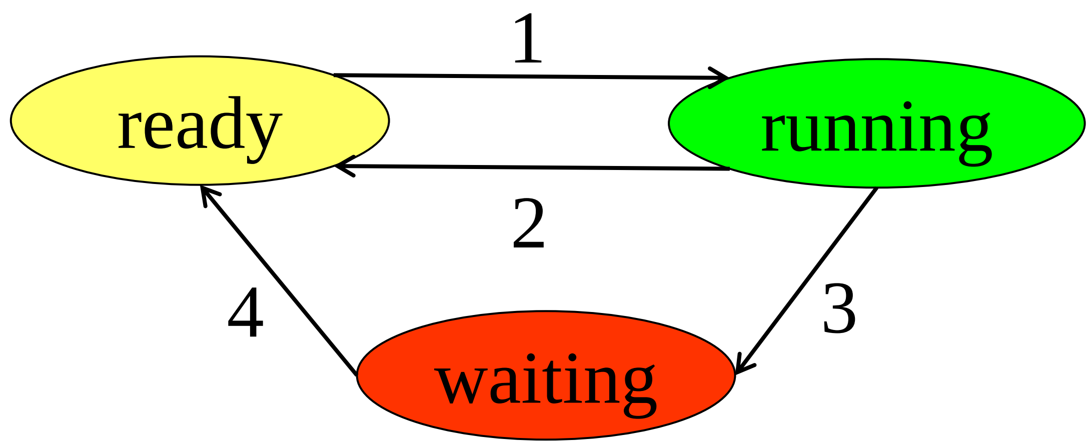
See the following diagram of process state transition. Can the following cause and effect transitions be possible? If yes, give an example.

(a) $2 \rightarrow 1$

(b) $3 \rightarrow 2$

(c) $4 \rightarrow 1$

(d) $1 \rightarrow 3$



a) Yes. If the process is Ready, when CPU is free, the process can be running.

c） Yes. If the process is waiting for some event to occur, such as an I/O completion, after I/O completion, the process is waked up and Ready, this may cause CPU scheduling, when CPU is free, or preemptive scheduler is used, the process can be running.

Exercise 9

- **identify the values of pid at lines A, B, C, and D. (Assume that the actual pids of the parent and child are 2600 and 2603, respectively.)**

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
int main()
{
    pid t pid, pid1;
    /* fork a child process */
    pid = fork();
    if (pid < 0) { // error occurred
    fprintf(stderr, "Fork Failed");
    return 1;
    }
```

```
  else if (pid == 0) { // child process
      pid1 = getpid();
      printf("child: pid = %d",  pid);     //A   0
      printf("child: pid1 = %d", pid1);  //B   2603
  }
  else { /* parent process */
      pid1 = getpid();
      printf("parent: pid = %d",   pid);     //C   2603
      printf("parent: pid1 = %d",  pid1);  //D   2600
      wait(NULL);
   }
   return 0;
}
```

Exercise 1

```
#include <pthread.h>    #include <stdio.h>    #include <types.h>
int value = 0;
void *runner(void *param)  {  /* the thread */
    value = 5;        pthread exit(0);
}
int main (int argc, char *argv[]) {
    pid t pid;              pthread t tid;      pthread attr t attr;
    pid = fork();
    if (pid == 0) {     /* child process */
        pthread attr init (&attr);
        pthread create (&tid, &attr, runner, NULL);
        pthread join (tid, NULL);
        printf("CHILD: value = %d", value); /* LINE C */        5
    }
    else if (pid > 0)   {   /* parent process */
        wait(NULL);
        printf("PARENT: value = %d", value); /* LINE P */        0
    }
}
```

Wensheng Li   BUPT

Scheduling-Algorithms Summary

尊老	FCFS	non-preemptive
爱幼	SJF	non-preemptive Preemptive -- SRTF
照顾优先	Priority Scheduling	non-preemptive preemptive
平等	RR	non-preemptive
等级秩序	Multilevel Queue Scheduling	preemptive
等级秩序	Multilevel Feedback Queue Scheduling	preemptive
公平	HRRN	non-preemptive

Exercise 1

process	Arrival time	Burst time
P1	0.0	8
P2	0.4	4
P3	1.0	1

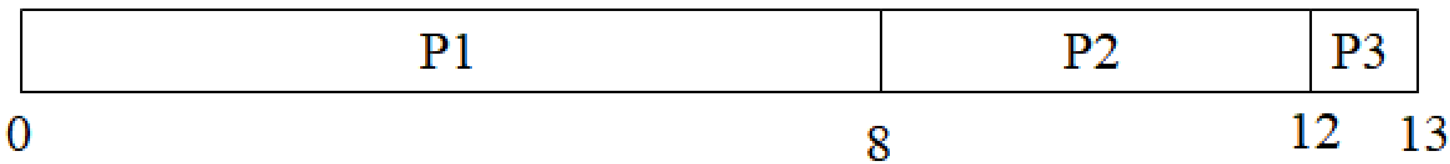
a. What is the **average turnaround time** for these processes with the **FCFS** scheduling algorithm?

b. What is the **average turnaround time** for these processes with the **SJF** scheduling algorithm?

c. The SJF algorithm is supposed to improve performance, but notice that we chose to run process *P*1 at time 0 because we did not know that two shorter processes would arrive soon. Compute what the average turnaround time will be if the CPU is left idle for the first 1 unit and then SJF scheduling is used. Remember that processes *P*1 and *P*2 are waiting during this idle time, so their waiting time may increase. This algorithm could be known as future-knowledge scheduling.

Answer a: FCFS scheduling algorithm

process	Arrival time	Burst time
P1	0.0	8
P2	0.4	4
P3	1.0	1



- **turnaround time:**
  T1=8, T2=12-0.4=11.6, T3=13-1=12
- **average turnaround time:**
  AT=(T1+T2+T3)/3=(8+11.6+12)/3=31.6/3 ≈10.53

Answer b: SJF scheduling algorithm

process	Arrival time	Burst time
P1	0.0	8
P2	0.4	4
P3	1.0	1

P1	P3	P2

0 8 9 13

- **turnaround time:**
  T1=8, T2=13-0.4=12.6, T3=9-1=8
- **average turnaround time:**
  AT=(T1+T2+T3)/3=(8+12.6+8)/3=28.6/3 ≈9.53

Answer c: SJF

可预知未来??

future-knowledge scheduling algorithm

process	Arrival time	Burst time
P1	0.0	8
P2	0.4	4
P3	1.0	1

	P3	P2	P1
0	1	2	6	14

- **turnaround time:**
  T1=14, T2=6-0.4=5.6, T3=2-1=1
- **average turnaround time:**
  AT=(T1+T2+T3)/3=(14+5.6+1)/3=20.6/3 ≈6.87

Exercise

In a computer system with only one processor, one input device and one printer. Processes A and B enter the system sequentially at time 0, and A is scheduled by the CPU scheduler at first. The execution tracks of A and B are as follows:

A: CPU burst lasting 20ms, then I/O burst of 100ms on the input device, and then CPU burst lasting 50ms, exiting.

B: CPU burst lasting 40ms, then I/O burst of 70ms on the printer, and then CPU burst lasting 30ms, exiting.

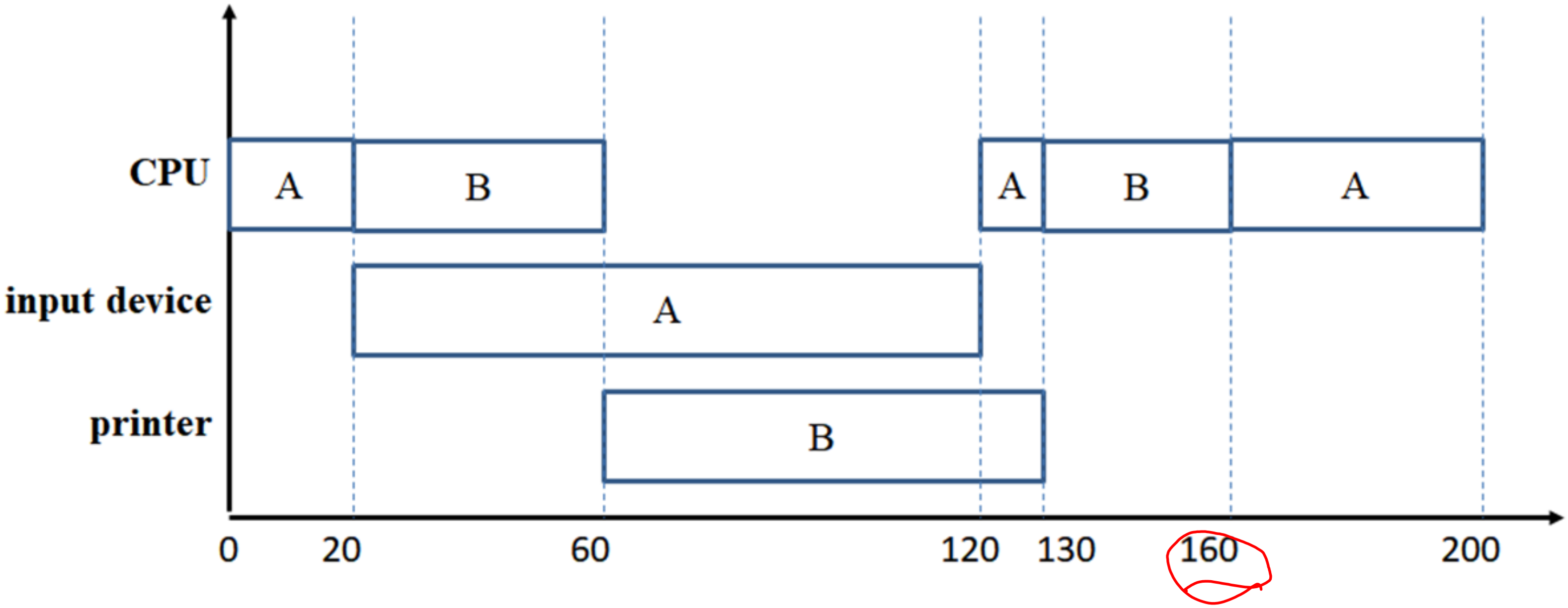
(a) Suppose that preemptive SJF scheduling algorithm (SRTF) is employed, draw the Gantt chart to describe the resource usage of A and B on the CPU, the input device and the printer.

(b) Calculate the waiting time and turnaround time for process A and B respectively.

answer

甘特图.

(1)



(2)

	waiting time(ms)	turnaround time(ms)
for process A	30	200
for process B	20	160

Exercise 2

- 某系统采用基于动态优先级的抢占式调度算法，并且优先数越大的进程其优先级越高。

  系统为所有新建进程赋予优先数 **0**;

  当一个进程在就绪队列中等待**CPU**时，其优先数的变化速率为 α;

  进程获得**CPU**后开始执行，执行过程中，其优先数的变化速率为β。

  为参数 α 和 β 设置不同的值，则导致不同的调度算法。

问题：

a. 如果 β>α>0，则调度原则是什么？

b. 如果 α<β<0，则调度原则是什么？

解答a：如果 β>α>0，调度原则是FCFS

分析：

- 由于新建进程的优先数是 0， 具有最低优先级。
- 当它在就绪队列中等待时，进程的优先数以速率 α 增加，即优先级不断提高，所以最早进入就绪队列的进程，其优先级也最高。
- 优先级最高的队首进程获得调度。
- 当进程在CPU上执行时，它的优先数以速率 β 增加，β>α，任何ready状态的进程不可能具有比正在 **running** 的进程更高的优先级，所以，**running**进程将一直占有CPU，直到它执行结束。
- 然后，在就绪队列中等待时间最长的进程被调度到CPU上执行。

所以，调度原则是 **first-come first-served**。

Ready queue: FIFO queue.

解答b：如果 α<β<0，调度原则是LCFS

分析：

- 新建进程的优先数是 0，具有最高优先级。
  由于新建进程的优先数是0， 一旦它进入就绪队列，它的优先数就开始以速率 α 减少，就绪时间越长，其优先数越小，即优先级越低。
- 新建进程优先级最高，被调度到CPU上执行。
- **running** 状态的进程，其优先数以速率 β 减少，α<β<0，故任何**ready** 进程不可能具有比 **running** 进程更高的优先级。
- 新建进程抢占CPU。原来的执行进程进入就绪队列。
- 如果进程执行过程中，没有新建进程，则将一直占有CPU，直到它执行结束。然后，在就绪队列中等待时间最短的进程被调度到CPU上执行。

所以，调度原则是 **Last-Come First-Served**。

ready queue: stack

Wensheng Li BUPT

Exercise 3

- **Consider the following set of processes P1, P2, P3 and P4. For 1≤ i ≤4, the following table shows their arrival time, CPU-burst time, and their priority number. Here, a smaller priority number implies a higher priority.**

Process	Arrival Time	Burst Time	Priority Number
P1	0.00	4.001	2
P2	3.00	3.001	1
P3	4.00	3.001	4
P4	5.00	5.001	3

(1) Suppose that HRRN scheduling is employed,

Draw a Gantt chart illustrating the execution of these processes.

What are the average waiting time and the average turnaround time.

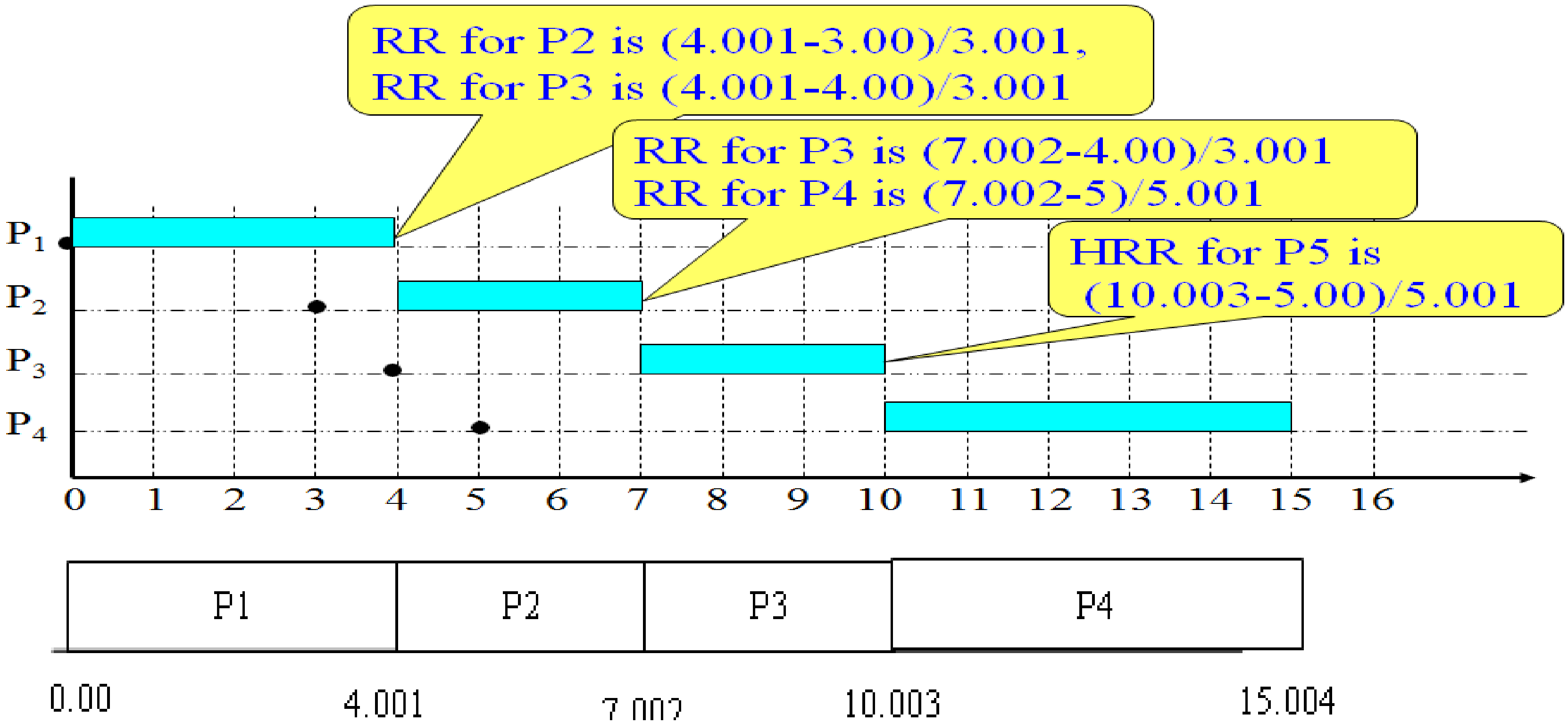
(2) Suppose that nonpreemptive priority scheduling is employed,

Draw a Gantt chart illustrating the execution of these processes.

What are the average waiting time and the average turnaround time.

Answer (1) HRRN

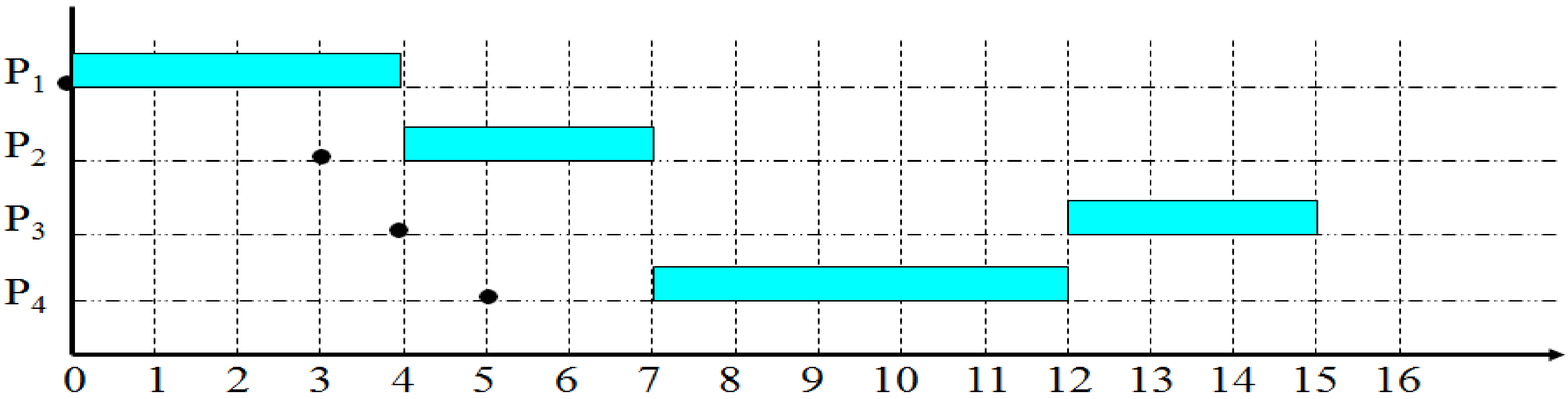
Process	Arrival Time	Burst Time	Priority Number
P1	0.00	4.001	2
P2	3.00	3.001	1
P3	4.00	3.001	4
P4	5.00	5.001	3



	Waiting time	Turnaround time
P1	0.00-0.00 = 0.00	4.001-0.00 = 4.001
P2	4.001-3.00 = 1.001	7.002-3.00 = 4.002
P3	7.002-4.00 = 3.002	10.003-4.00 = 6.003
P4	10.003-5.00 = 5.003	15.004-5.00 = 10.004
Average time	2.2515	6.0025

Answer (2) Priority

Process	Arrival Time	Burst Time	Priority Number
P1	0.00	4.001	2
P2	3.00	3.001	1
P3	4.00	3.001	4
P4	5.00	5.001	3



P1	P2	P4	P3

0.00 4.001 7.002 12.003 15.004

	Waiting time	Turnaround time
P1	0.00-0.00 = 0.00	4.001-0.00 = 4.001
P2	4.001-3.00 = 1.001	7.002-3.00 = 4.002
P3	12.003-4.00 = 8.003	15.004-4.00 = 11.004
P4	7.002-5.00 = 2.002	125.003-5.00 = 7.003
Average time	2.7515	6.5025

Supplement 1

- **Considering a real-time system, in which there are 4 real-time processes P1, P2, P3 and P4 that are aimed to react to 4 critical environmental events e1, e2, e3 and e4 in time respectively.**

  The arrival time of each event ei, 1≤i≤4, (that is, the arrival time of the process Pi), the length of the CPU burst time of each process Pi, and the deadline for each event ei are given below. Here, the deadline for ei is defined as the absolute time point before which the process Pi must be completed.

  The priority for each event ei (also for Pi) is also given, and a smaller priority number implies a higher priority.

Supplement 1(Cont.)

Events	Process	Arrival Time	Burst Time	Priorities	Deadline
e1	P1	0.00	4.00	3	7.00
e2	P2	3.00	2.00	1	5.50
e3	P3	4.00	2.00	4	12.01
e4	P4	6.00	4.00	2	11.00

① **Draw a Gantt chart illustrating the execution of these processes using the following scheduling algorithms: preemptive priority and FCFS.**

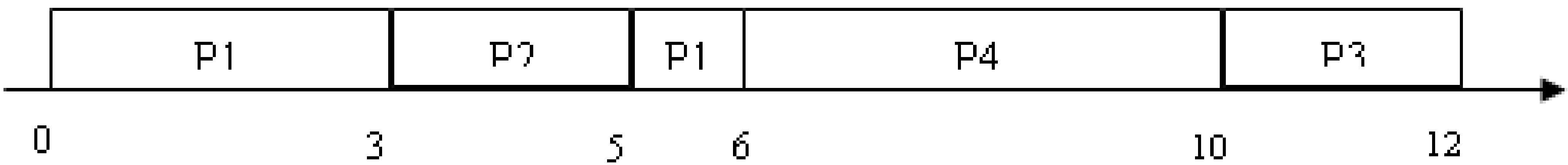
② **What is the average waiting time for each of the scheduling algorithms?**

③ **What is the average turnaround time each of the scheduling algorithms?**

④ **Which event will be treated with in time for each of the scheduling algorithms? (that is, the process reacting to this event will be completed before its deadline?)**

Answer for supplement 1

- **preemptive priority**



平均等待时间:

[(5—3) + (3—3) + (10—4) + (6—6)] /4 = 2

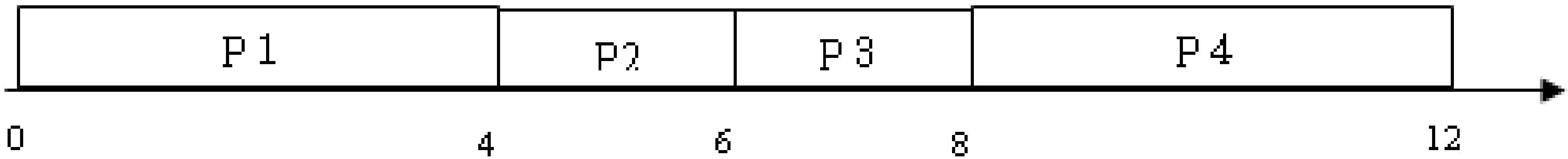
平均周转时间:

[(6—0) + (5—3) + (12—4) + (10—6)] /4 = 5

根据各进程的完成时间点和所对应的事件的deadline可知，全部 4 个事件均可得到及时响应.

Answer for supplement 1(Cont.)

- **FCFS**



平均等待时间:

[(0—0) + (4—3) + (6—4) + (8—6)] /4 = 1.25

平均周转时间:

[(4—0) + (6—3) + (8—4) + (12—6)] /4 = 4.25

根据各进程的完成时间点和所对应的事件的deadline可知，事件e1和e3可得到及时响应

Wensheng Li BUPT

Supplement 2

- **The following processes are being scheduled using a preemptive, round-robin scheduling algorithm. Each process is assigned a numerical priority, with a higher number indicating a higher relative priority. In addition to the processes listed below, the system also has an *idle task* (which consumes no CPU resources and is identified as *Pidle*). This task has priority 0 and is scheduled whenever the system has no other available processes to run. The length of a time quantum is 10 units. If a process is preempted by a higher-priority process, the preempted process is placed at the end of the queue.**

Supplement 2(Cont.)

Thread	Priority	Burst	Arrival
P_1	40	20	0
P_2	30	25	25
P_3	30	25	30
P_4	35	15	60
P_5	5	10	100
P_6	10	10	105

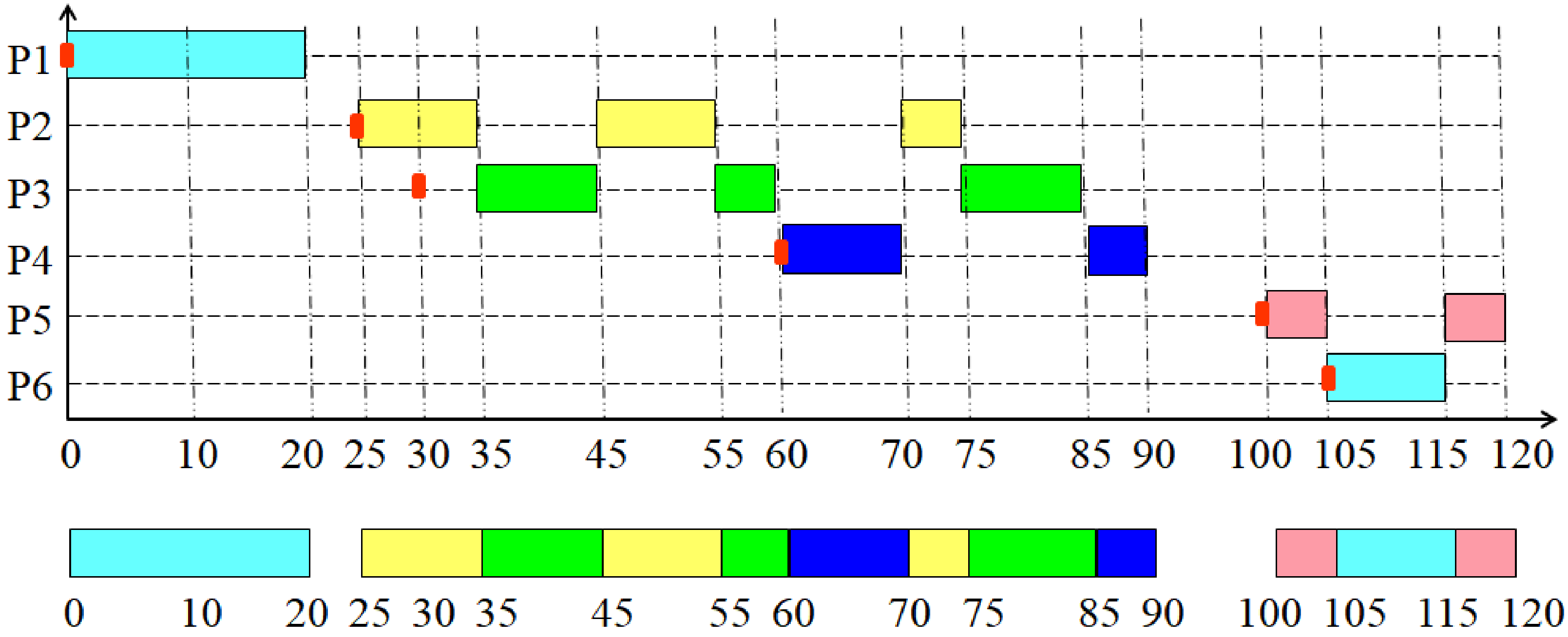
① **Show the scheduling order of the processes using a Gantt chart.**

② **What is the turnaround time for each process?**

③ **What is the waiting time for each process?**

④ **What is the CPU utilization rate?**

Answer for supplement 2



	P1	P2	P3	P4	P5	P6
Turnaround time	20	50	55	30	20	10
Waiting time	0	25	30	15	10	0

CPU utilization rate: 105/120=87.5%

Answer 1 to 6.11

```
Semaphore
    customers=0; // 等待的顾客数，不包括正在理发的顾客
    barber=1;       // 理发师状态，1-闲，
                      用于理发师进程和顾客进程之间的同步
    mutex=1;        // 控制对waiting的互斥访问
int waiting=0;    // 等待的顾客数，包括正在理发的顾客
```

```
void barber ( ) {
    while (1)  {
        wait (customers);
        givehaircut ();
        signal (barber);
    }
}
```

Answer 1 to 6.11 (cont.)

```
void customer () {
    wait (mutex);
    if (waiting==n+1) {  signal (mutex);  exit();  }
    waiting++;
    signal (customers);
    signal (mutex);
    wait (barber);
    receiveHaircut();
    wait (mutex);
    waiting--;
    signal (mutex);
}
```

Answer 2 to 6.11

Semaphore

customers=0; // 用于理发师进程和顾客进程之间的同步

barber=0; // 用于顾客进程之间的同步

mutex=1; // 控制对waiting的互斥访问

int waiting=0; // 坐在椅子上等待的顾客数

```
void barber ( ) {
   while (1)  {
      wait (customers);
      givehaircut ();
   }
}
```

Answer 2 to 6.11 (cont.)

```
void customer () {
    wait (mutex);
    if (waiting==n+1) { signal (mutex); exit(); }
    waiting++;
    if (waiting>1) { take a chair; signal(customers);
                    signal (mutex); wait(barber); }
    else { signal (mutex);
           signal (customers); } // 唤醒理发师
    receiveHaircut();
    wait (mutex);
    waiting--;
    if (waiting>0) { signal(barber); }
    signal (mutex);
}
```

Exercise

- 某理发店有3位理发师，3把理发椅，n把供等候理发的顾客坐的椅子。如果没有顾客到来，理发师们便在理发椅上睡觉。当顾客到来时，必须先唤醒一个理发师，如果理发师们正在理发时又有顾客到来，如果有空椅子，则顾客坐下休息排队等待，如果没有空椅子了，则顾客离开。
- 请使用信号量机制解决理发师进程和顾客进程之间的同步/互斥问题。

wait(customer)

wait(mutex)

Answer to Exercise

- **Semaphores**
  - **customers=0; // 正在等待的顾客数，包括正在理发的**
  - **barbers=3;   // 空闲的理发师数**
  - **mutex=1;   // 控制对C_waiting的互斥访问**
- **Variable**
  - **int C_waiting=0; // 坐在椅子上等待的顾客数**

customers = 0

babers = 3

mutex = 1

int C_waiting = 0

Answer to Exercise (Cont.)

c_waiting--;
signal(mutex))

```
void custumer (){
    wait(mutex);
    if (C_waiting<n) {
        C_waiting++;
        signal(mutex);
        signal(customers);
        wait(barbers);
        gethaircut();  }
    else signal(mutex);
}
```

```
void barber() {
  while(1){
     wait(customers);
     wait(mutex);
     C_waiting--;
     signal(mutex);
     cuthair();
     signal(barbers);
  }
}
```

wait(cm)

M: wait(cm)

取.

but

Wensheng Li BUPT

Exercise 5

- 某应用有三个进程，C、M、和P。
  - 数据采集进程 C 把采集到的数据放到buf1中。
  - 数据处理进程 M 从buf1中取数据进行处理，并把结果放入buf2中。
  - 数据输出进程 P 从buf2中取出结果打印输出。

考虑以下两种情况，用信号量机制实现进程C、M、和P之间的同步，分别给出进程C、M、和P的代码结构

(1) Buf1、buf2都只能保存一个数据

(2) Buf1可以保存m个数据、buf2可以保存n个数据

（假设各进程对缓冲区的操作需要互斥）

(1) Semaphore.

Wensheng Li BUPT

Answer: (1) buf1、buf2都只能存放一个数据

signal(cm)

Semaphore： cm=1; mc=0; mp=1; pm=0; cm. mc		

Process C
{
While (1) {
采集数据;
wait(cm);
数据存入 buf1;
signal(mc)
}
}
	Process M
	{
	While (1) {
	wait(mc);
	从buf1中取出数据;
	signal(cm);
	对数据进行处理;
	wait(mp);
	处理结果存入buf2;
	signal(pm)
	}
	}
		Process P
		{
		While (1) {
		wait(pm);
		从buf2中取出结果;
		signal(mp);
		打印结果
		}
		}

同步信号量，后wait() 前signal()

Wensheng Li BUPT

empty1=m

Answer: (2) buf1可以存放m个,buf2能存放n个

Semaphore： empty1=m; full1=0; empty2=n; full2=0; mutex1=1; mutex2=1

Process C	Process M	Process P
{	{	{
While (1) {	While (1) {	While (1) {
采集数据;	wait(full1);	wait(full2);
wait(empty1);	wait(mutex1);	wait(mutex2);
wait(mutex1);	从Buf1中取出数据;	从Buf2中取出结果;
把数据存入Buf1;	signal(mutex1);	signal(mutex2);
signal(mutex1);	signal(empty1);	signal(empty2);
signal(full1)	对数据进行处理;	打印结果
}	wait(empty2);	}
}	wait(mutex2);	}
	把处理结果存入Buf2;	
	signal(mutex2);	
	signal(full2)	
	}	
	}	

这、、没啥好的

Wensheng Li BUPT

Exercise 6

- 某工厂有一条生产线，其上可以放置10个零件。有三个工人小张、小王和小李。
  - 小张每次生产1个零件A，并放置到生产线上；
  - 小王每次生产1个零件B，并放置到生产线上；
  - 小李每次从生产线上取2个A和3个B，组装产品C。

  工人们不能同时使用生产线取放零件。

请用信号量机制实现三个工人进程，确保流水线能够正常工作。

（1）定义信号量及变量，给出其初值，说明其作用

（2）写出三个工人进程的代码结构

empty=10

full=0

A: roomA=7

Answer:

wait(roomA)
wait(empty)
wait(mutex)

put

signal(mutex)

signal(partA)

- Semaphore
  mutex=1;
  partA=0; // A的数量
  partB=0; // B的数量
  roomA=7; // A可用空间
  roomB=8; // B可用空间

- Zhang:
  while(1){
    produce A;
    wait(roomA);
    wait(mutex);
    put A on line;
    signal(mutex);
    signal(partA);
  }

- Wang:
  while(1){
    produce B;
    wait(roomB);
    wait(mutex);
    put B on line;
    signal(mutex);
    signal(partB);
  }

- Li:
  while(1){
    wait(partA);
    wait(partA);
    wait(partB);
    wait(partB);
    wait(partB);
    wait(mutex);
    get two A and three B from line;
    signal(mutex);
    signal(roomA);
    signal(roomA);
    signal(roomB);
    signal(roomB);
    signal(roomB);
    produce C;
  }

signal(empty) 5次

Wensheng Li BUPT

Exercise 7 B：room=8.

- 某系统中有m个进程并发执行，分为A、B两组，他们共享文件F，A组进程对文件F进行写操作，B组进程对文件F进行只读操作。写操作需要互斥进行，读操作可以同时发生，即当有B组的某进程正在对文件F进行读操作时，若没有A组的进程提出写请求，则B组的其他进程可以同时对文件F进行读操作，如果有A组的进程提出写请求，则阻止B组的其他进程对文件F进行读操作的后续请求，等当前正在读文件的B组进程全部执行完毕，则立即让A组提出写请求的进程执行对文件F的写操作。

（1）定义信号量及变量，给出其初值，说明其作用

（2）写出A组和B组进程的代码结构

Wensheng Li BUPT

Answer 1:

Writer相对优先

```
int reader_count=0;
Semaphore
    mutex=1;
    RW_mutex=1;
    W_first=1;

writers(){
    wait(W_first);
    wait(RW_mutex);
    writing file F;
    signal(RW_mutex);
    signal(W_first);
}
```

W_first=1

wait(W_first)

```
Readers(){
    wait(W_first);
    wait(mutex);
    reader_count++;
    if (reader_count==1
        wait(RW_mutex);
    signal(mutex);
    signal(W_first);
    reading file F;
    wait(mutex);
    reader_count--;
    if (reader_count==0
        signal(RW_mutex);
    signal(mutex);
}
```

Answer 2:

Writer绝对优先

Int R-num=0; W_num=0;

Semaphore wrt=1; W_first=1; mutex_w=1; mutex_R=1;

```
Writer(){
    Wait(mutex_w);
    W_num++;
    If (W_num==1)  wait(W_first);
    signal(mutex_w);
    Wait(wrt);
    写文件;
    signal(wrt);
    Wait(mutex_w);
    W_num--;
    If (W-num==0) signal(W_first);
    signal(mutex_w);
}
```

```
Reader(){
    Wait(W_first);
    Wait(mutex_R);
     R_num++;
     if (R_num==1)  wait(wrt);
     signal(mutex_R);
     signal(W_first);
     读文件;
     Wait(mutex_R);
     R_num--;
     If (R_num==0)  signal(wrt);
     signal(mutex_R);
}
```

只要有写的
就饿了?

Wensheng Li BUPT