1.

1.10

An interrupt is a hardware-generated change-of-flow within the system. An interrupt handler is summoned to deal with the cause of the interrupt; control is then returned to the interrupted context and instruction.

A trap is a software-generated interrupt.

An interrupt can be used to signal the completion of an I/O to obviate the need for device polling.

A trap can be used to call operating system routines or to catch arithmetic errors.

1.11

a) Host writes a DMA command block into memory, CPU writes the address of this command block to the DMA controller, DMA controller operates the memory bus directly, placing address on the bus to perform transfers

b) DMA interrupts CPU to signal transfer completion

c) yes. Because cycle stealing（周期挪用） is used to transfer data on the system bus, so the instruction cycle is suspended so data can be transferred, the execution of the user programs is slowdown.

1.13

Caches are useful when two or more components need to exchange data, and the components perform transfers at differing speeds. Caches solve the transfer problem by providing a buffer of intermediate speed between the components. If the fast device finds the data it needs in the cache, it need not wait for the slower device.

The data in the cache must be kept consistent with the data in the components. If a component has a data value change, and the datum is also in the cache, the cache must also be updated.

This is especially a problem on multiprocessor systems where more than one process may be accessing a datum.

A component may be eliminated by an equal-sized cache, but only if:

(a) the cache and the component have equivalent state-saving capacity (that is, if the component retains its data when electricity is removed, the cache must retain data as well), and

(b) the cache is affordable, because faster storage tends to be more expensive.

1.17

a. Batch: Jobs with similar needs are batched together and run through the computer as a group by an operator or automatic job sequencer. Performance is increased by attempting to keep CPU and I/O devices busy at all times through buffering, off-line operation, spooling, and multiprogramming. Batch is good for executing large jobs that need little interaction; it can be submitted and picked up later.

b. Interactive: This system is composed of many short transactions where the results of the next transaction may be unpredictable. Response time needs to be short (seconds) since the user submits and wait for the results.

c. Time sharing: This systems uses CPU scheduling and multiprogramming to provide economical interactive use of a system. The CPU switches rapidly from one user to another. Instead of having a job defined by spooled card images, each program reads its next control card from the terminal, and output is normally printed immediately to the screen.

d. Real time: Often used in a dedicated application, this system reads information from sensors and must respond within a fixed amount of time to ensure correct performance.

e. Network: In the simplest terms, is a communication path between two or more systems. Networks vary by the protocols used, the distances between nodes, and the transport media.

f. Parallel: Such systems have more than one processor in close communication, sharing the computer bus, the clock, and sometimes memory and peripheral devices.

g. Distributed: This system distributes computation among several physical processors. The processors do not share memory or a clock. Instead, each processor has its own local memory. They communicate with each other through various communication lines, such as a high-speed bus or telephone line.

h. Clustered: Clustered systems gather together multiple CPUs to accomplish computational work. Clustered systems coupled together.

i. Handheld: Handheld systems include personal digital assistants (POAs), such as Palm-Pilots or cellular telephones with connectivity to a network such as the Internet. Most handheld devices have a small amount of memory, include slow Processors, and feature small display screen.

2

2.1

Helpful to the user

Helpful to ensuring the efficient operation of the system

2.3

Register

Parameters are stored in a block in memory, and the address of the block is passed as a parameter in a register

stack

2.8

Message passing

- **Useful for exchanging smaller amount of data**
- **Processes can be running in different computer**

shared memory

- **Allows maximum speed and convenience of communication**
- **Processes must have shared memory**

3

3.1

Short-term (CPU scheduler)—selects from processes in memory those processes that are ready

to execute and allocates the CPU to them.
Medium-term—used especially with time-sharing systems as an intermediate scheduling level. A swapping scheme is implemented to remove partially run programs from memory and reinstate them later to continue where they left off.
Long-term (job scheduler)—determines which jobs are brought into memory for processing.
The primary difference is in the frequency of their execution. The short-term must select a new process quite often. Long-term is used much less often since it handles placing jobs in the system and may wait a while for a job to finish before it admits another one.

3.2
In general, the operating system must save the state of the currently running process and restore the state of the process scheduled to be run next. Saving the state of a process typically includes the values of all the CPU registers in addition to memory allocation. Context switches must also perform many architecture-specific operations, including flushing data and instruction caches.

3.4
5

4

4.2
Context switching between user threads is quite similar to switching between kernel threads, although it is dependent on the threads library and how it maps user threads to kernel threads. In general, context switching between user threads involves taking a user threads of its LWP and replacing it with another threads. This act typically involves saving and restoring the state of the registers.

补充 1
a. **User-level threads are unknown by the kernel, whereas the kernel is aware of kernel threads.**
b. **User threads are scheduled by the thread library and the kernel schedules kernel threads.**
c. **Kernel threads need not be associated with a process whereas every user thread belongs to a process.**

补充 2
Because a thread is smaller than a process, thread creation typically uses fewer resources than process creation.
Creating a process requires allocating a process control block (PCB), a rather large data structure. The PCB includes a memory map, list of open files, and environment variables. Allocating and managing the memory map is typically the most time-consuming activity.
Creating either a user or kernel thread involves allocating a small data structure to hold a register set, stack, and priority.

5

5.4

a The four Gantt charts are:

FCFS	1	1	1	1	1	1	1	1	1	2	3	3	4	5	5	5	5	5	5
RR	1	2	3	4	5	1	3	5	1	5	1	5	1	5	1	1	1	1	1
SJF	2	4	3	3	5	5	5	5	1	1	1	1	1	1	1	1	1	1	1
Priority	2	5	5	5	5	5	1	1	1	1	1	1	1	1	1	1	3	3	4

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

b Turnaround time:

	FCFS	RR	SJF	Priority
P1	10	19	19	16
P2	11	2	1	1
P3	13	7	4	18
P4	14	4	2	19
P5	19	14	9	6

c Waiting time (turnaround time minus burst time):

	FCFS	RR	SJF	Priority
P1	0	9	9	6
P2	10	1	0	0
P3	11	5	2	16
P4	13	3	1	18
P5	14	9	4	1

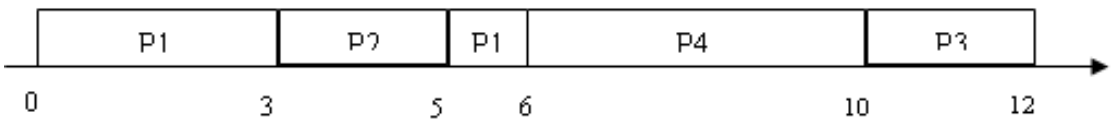
d Short Job First.

5.5

- **b**
- **d**

补充 1

- **preemptive priority：**



平均等待时间:

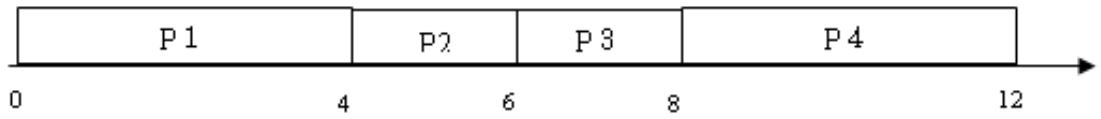
[(5–3) + (3–3) + (10–4) + (6–6)] /4 = 2

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.

- **FCFS**



平均等待时间:

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

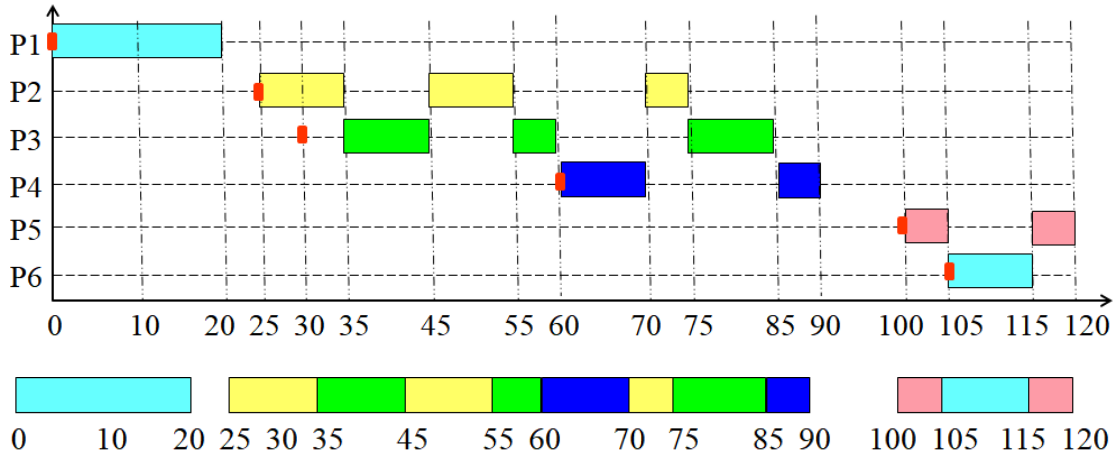
平均周转时间:

[(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

补充 2

（1）



(2) **turnaround time for each process**

P1: 20, P2: 50, P3: 55, P4: 30, P5: 20, P6: 10

(3) **waiting time for each process**

P1: 0, P2: 25, P3: 30, P4: 15, P5: 10, P6: 0

(4) CPU utilization rate: 105/120=87.5%

6

6.1

互斥：**turn=i, Pi enter into its critical section**

Progress: i 不在临界区，则

1) flag[i]=false 或

2) Flag[i]=true , but turn=j

Pj 请求，**flag[j]=true,**

1) 则 **Pj** 进入

2) Pj 可进入，**Pi** 等待 **Pj** 退出后，将 **turn=i** 后，进入

Bounded waiting：i 在临界区中，flag[i]=true turn=I
Pj 等待，Pi 退出后，设置 turn=j，则 Pj 进入。

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 wait 的互斥访问
int waiing=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);
   }
}

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

6.16

In monitor, The x.signal() operation resumes exactly one suspended process. If no process is suspended, then the signal operation has no effect.
For semaphore, signal() operation removes one process from the list of waiting processes and awakens that process.
If there is no process in the list of waiting processes, the value of the semaphore increase 1.

7

7.5

a. Increase *Available* (new resources added). -- safely
b. Decrease, *Available* (resource permanently removed from system).

若减少的资源不是进程所需要的，或者资源减少后，剩余资源还可以满足进程的需要，则是安全的。

c. Increase *Max* for one process (the process needs more resources than allowed; it may want more).

若Max增加后，系统仍可满足其需要，存在安全序列，则安全。

d. Decrease *Max* for one process (the process decides it does not need that many resources). --safely

e. Increase the number of processes.

若新增加进程所需要的资源小于当前的的可用资源，且存在安全序列，则安全。

f. Decrease the number of processes. -- safely

7.6

Suppose the system is deadlocked. This implies that each process is holding one resource and is waiting for one more.

Since there are three processes and four resources, one process must be able to obtain two resources. This process requires no more resources and, therefore it will return its resources when done.

7.11

(a)

	Allocation A B C D	Max A B C D	Need A B C D
P0	0 0 1 2	0 0 1 2	0 0 0 0
P1	1 0 0 0	1 7 5 0	0 7 5 0
P2	1 3 5 4	2 3 5 6	1 0 0 2
P3	0 6 3 2	0 6 5 2	0 0 2 0
p4	0 0 1 4	0 6 5 6	0 6 4 2

(b) Exist a execute sequence p0,p2,p1,p3,p4,so the system is safe.

(c) Exist a sequence p0,p2,p1,p3,p4,which can make the system in security. So the requirement can be fulfilled in time.

7.14

```
Semaphores
   Bridge=1;
   Mutex_n=1;
   Mutex_s=1;
Int counter_nfarmer=0;
     Counter_sfarmer=0;

North_farmer(){
   Wait(Mutex_n);
   counter_nfarmer++;
   if (counter_nfarmer==1) wait(bridge);
   signal(Mutex_n);
   walking through the bridge;
```

```
    Wait(Mutex_n);
    counter_nfarmer--;
    if (counter_nfarmer==0) signal(bridge);
    signal(Mutex_n);
}

South_farmer(){
    Wait(Mutex_s);
    counter_sfarmer++;
    if (counter_sfarmer==1) wait(bridge);
    signal(Mutex_s);
    walking through the bridge;
    Wait(Mutex_s);
    counter_sfarmer--;
    if (counter_sfarmer==0) signal(bridge);
    signal(Mutex_s);
}
```