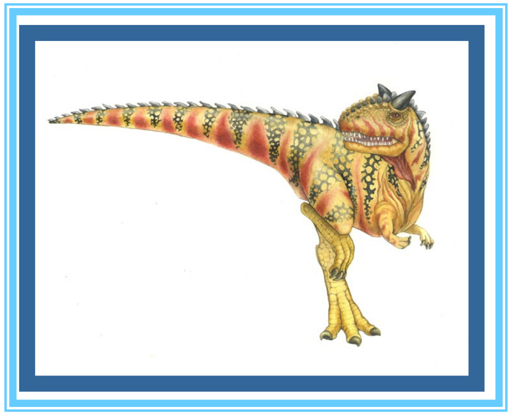
Review & Exercise



1. 操作系统的定义及特征

操作系统是管理和控制计算机系统中软硬件资源，合理组织计算机工作流程，方便用户操作使用机器的程序的集合。

基本特征:

- 执行的并发性
- 资源的共享性
- 操作的异步性

2. 操作系统的功能

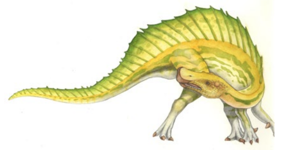
(1). 处理机管理(包括：进程管理)

(2). 存储管理

(3). 设备管理

(4). 文件系统管理

(5). 用户接口（作业管理）



3. 操作系统的三种基本类型及特点

(1). 批处理系统

主要特点：

① 脱机操作； ② 成批处理； ③ 多道程序运行； ④ 无交互性。

(2). 分时系统

主要特点：

① 交互性； ② 同时性； ③ 独立性； ④ 及时性。

(3). 实时系统

主要特点：

① 实时时钟管理； ② 连续人机对话； ③过载防护，安全可靠； ④ 资源利用率低

4. 多道程序运行及特点

多道程序设计：允许多作业同时进入内存轮流交替占用CPU运行的技术。

特点：（1）多道性

（2）宏观上并行

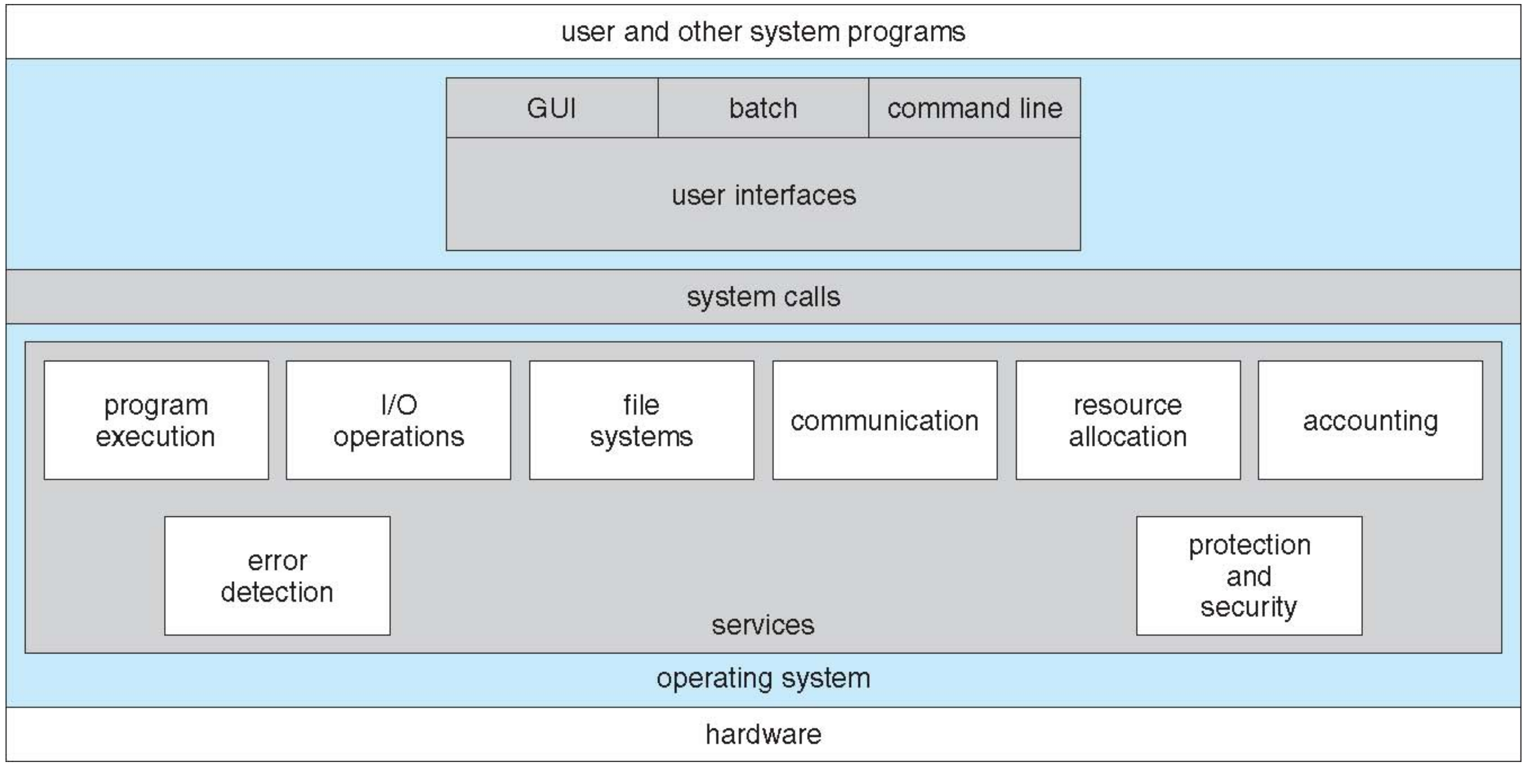
（3）微观上串行

Operating System Services

We can view OS from several points:

- The services that the OS provides
- The interface provided to users and programmers
- Its components and interconnections

A View of Operating System Services



Services to the user

User interface - Almost all operating systems have a user interface (UI)

Varies between

Command-Line (CLI) 命令行

Graphics User Interface (GUI) 用户界面

Batch 批处理

Program execution

The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)

Services to the user (Cont)

I/O operations

A running program may require I/O, which may involve a file or an I/O device

File-system manipulation

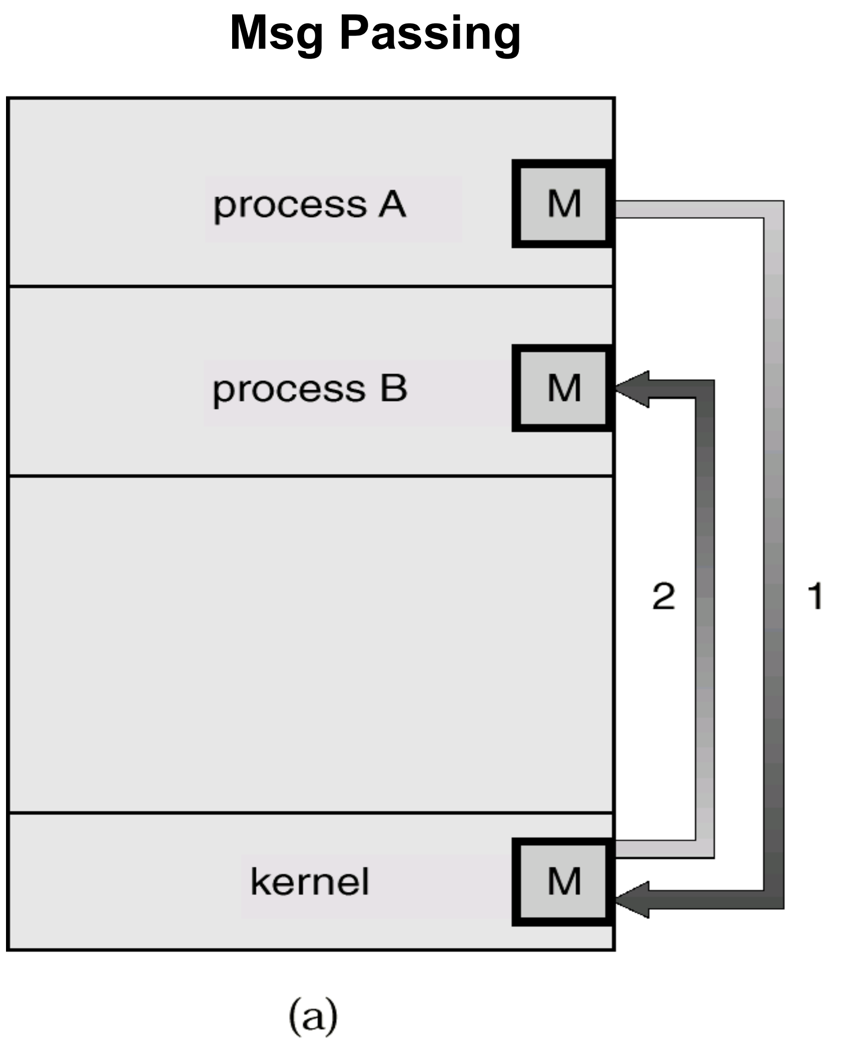
Obviously, programs need to read and write files and directories, create and delete them, search them, list file Information, permission management.

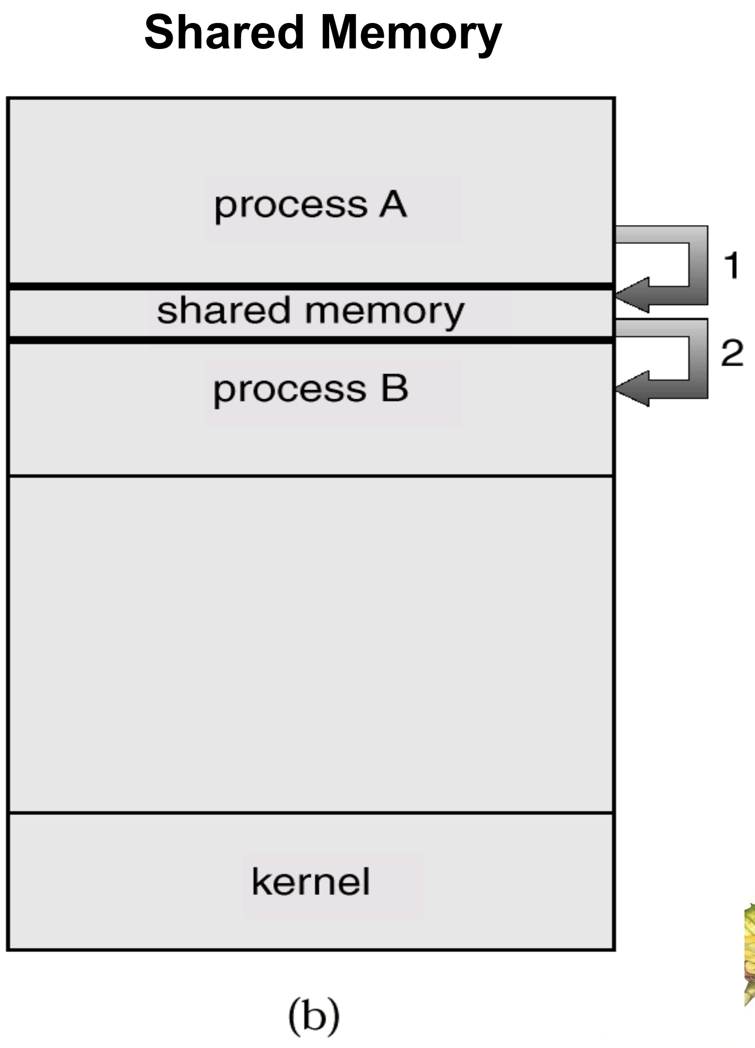
Services to the user (Cont)

Communications

- Processes may exchange information, on the same computer or between computers over a network
  - Communications may be via shared memory or through message passing (packets moved by the OS)

Communication Models





Services for the System Itself

Resource allocation - When multiple users or multiple jobs running concurrently, resources must be allocated to each of them

Many types of resources - Some (such as CPU cycles, main memory, and file storage) may have special allocation code, others (such as I/O devices) may have general request and release code

Accounting（统计）- To keep track of which users use how much and what kinds of computer resources

Services for the System Itself (Cont)

Protection and security - The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other

- **Protection** involves ensuring that all access to system resources is controlled
- **Security** of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts

User Operating System Interface

- Command interface
  - Command Line Interface (CLI) or command interpreter
  - GUI interface
- Program interface
  - System call

Operating System Structure

Simple Structure

Layered Approach

Microkernel System Structure

Modular Approach

Virtual Machines

A virtual machine takes the layered approach to its logical conclusion. It treats hardware and the operating system kernel as though they were all hardware

A virtual machine provides an interface *identical* to the underlying bare hardware

The operating system host creates the illusion that a process has its own processor and (virtual memory)

Each guest provided with a (virtual) copy of underlying computer

本章疑难点

1、并行性与并发性的区别和联系

并行性和并发性是既相似又有区别的两个概念。并行性是指两个或多个事件在同一时刻发生，并发性是指两个或多个事件在同一时间间隔内发生。

在多道程序环境下，并发性是指在一段时间内，宏观上有多个程序同时运行，但在单处理器系统中每个时刻却仅能有一道程序执行，因此微观上这些程序只能分时地交替执行。若在计算机系统中有多个处理器，则这些可以并发执行的程序便被分配到多个处理器上，实现并行执行，即利用每个处理器来处理一个可并发执行的程序。

本章疑难点

2、特权指令与非特权指令

谓特权指令，是指有特殊权限的指令，由于这类指令的权限最大，使用不当将导致整个系统崩溃，如清内存、置时钟、分配系统资源、修改虚存的段表或页表、修改用户的访问权限等。若所有程序都能使用这些指令，则系统一天死机n次就不足为奇。

为保证系统安全，这类指令只能用于操作系统或其他系统软件，不直接提供给用户使用。因此，特权指令必须在核心态执行。实际上，CPU在核心态下可以执行指令系统的全集。形象地说，特权指令是那些儿童不宜的东西，而非特权指令是老少皆宜的东西。

本章疑难点

3、访管指令与访管中断

访管指令是一条可以在用户态执行的指令。在用户程序中，因要求操作系统提供服务有意识地使用访管指令，从而产生一个中断事件(自愿中断)，将操作系统转换为核心态，称为访管中断。访管中断由访管指令产生，程序员使用访管指令向操作系统请求服务。

为什么要在程序中引入访管指令呢?这是因为用户程序只能在用户态下运行。若用户程序想要完成在用户态下无法完成的工作，该怎么办?解决这个问题要靠访管指令。访管指令本身不是特权指令，其基本功能是让程序拥有“自愿进管”的手段，从而引起访管中断。

处于用户态的用户程序使用访管指令对，系统根据访管指令的操作数执行访管中断处理程序，访管中断处理程序将按系统调用的操作数和参数转到相应的例行子程序。完成服务功能后，退出中断，返回到用户程序断点继续执行。

1. 程序顺序执行及特点

程序在处理机上的执行是严格按序的。

特点：

① 顺序性

② 封闭性

③ 可再现性

2．进程并发执行及特点

进程在处理机上的执行时间是交叉重叠的，是提高CPU利用率而采取的一种同步操作技术。

特点：

① 独立性

② 随机性

③ 资源共享性

3.　进程的定义及引入目的

定义 一个具有独立的功能的程序关于某个数据集在处理机上的一次执行过程及分配资源的基本单位。

引入目的

① 为了控制和协调并发程序对软硬件资源的共享和竞争。

② 为了描述程序动态执行的过程和有个分配资源的基本单位。

Process Concept

进程是动态的，程序是静态的：程序是有序代码的集合；进程是程序的执行。

进程是暂时的，程序是永久的：进程是一个状态变化的过程，程序可长久保存。

进程与程序的组成不同：进程的组成包括程序、数据和进程控制块（即进程状态信息）。

进程与程序的对应关系：通过多次执行，一个程序可对应多个进程；通过调用关系，一个进程可对应多个程序。

Process Concept

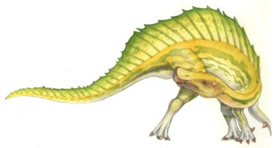
结构特征：进程实体=程序段+相关的数据段+PCB。

动态性：进程的实质是进程实体的一次执行过程，因此动态性是进程的最基本的特征。

并发性：多个进程实体同存在于内存中，且能在一段时间内同时运行。是最重要的特征。

独立性：进程实体是一个能独立运行、独立分配资源和独立接受调度的基本单位。

异步性：进程按各自独立的、不可预知的速度向前推进。



5. 进程的描述

进程的描述包括三部分：

① 程序

② 数据结构集

③ 进程控制块（PCB）

Process Concept

Two types process

- System process
- User process

Process Concept

系统进程与用户进程的区别:

资源的使用:系统进程被分配一个初始的资源集合，这些资源可以为它独占，也能以最高优先权的资格使用。用户进程通过系统服务请求的手段竞争使用系统资源;

I/O操作:用户进程不能直接做I/O操作，而系统进程可以做显式的、直接的I/O操作。

CPU工作状态:系统进程在管态下活动，而用户进程则在用户态下活动。

6. 进程的状态及转换

① 运行状态

一个进程正占用CPU执行。

② 等待状态

进程因等待某事件不能享用CPU.

③ 就绪状态

进程已具备运行条件尚未占用CPU。

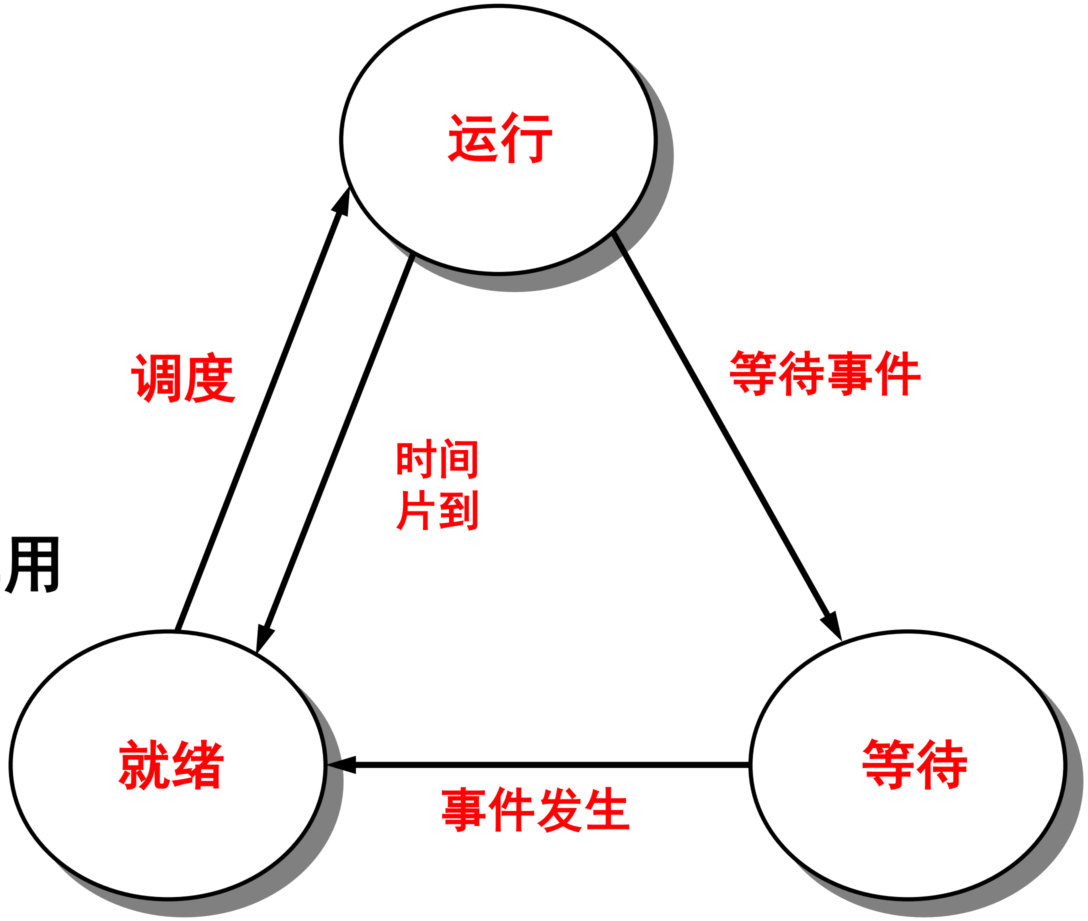
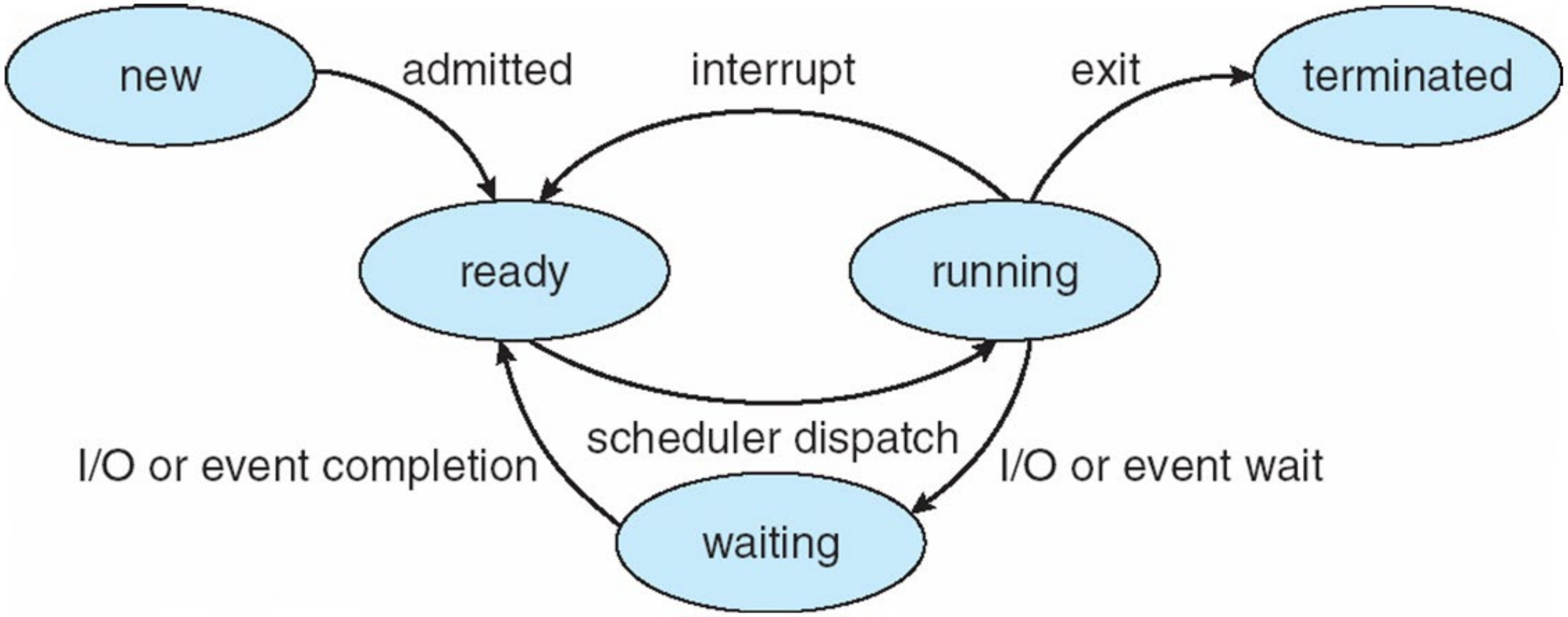


Diagram of Process State

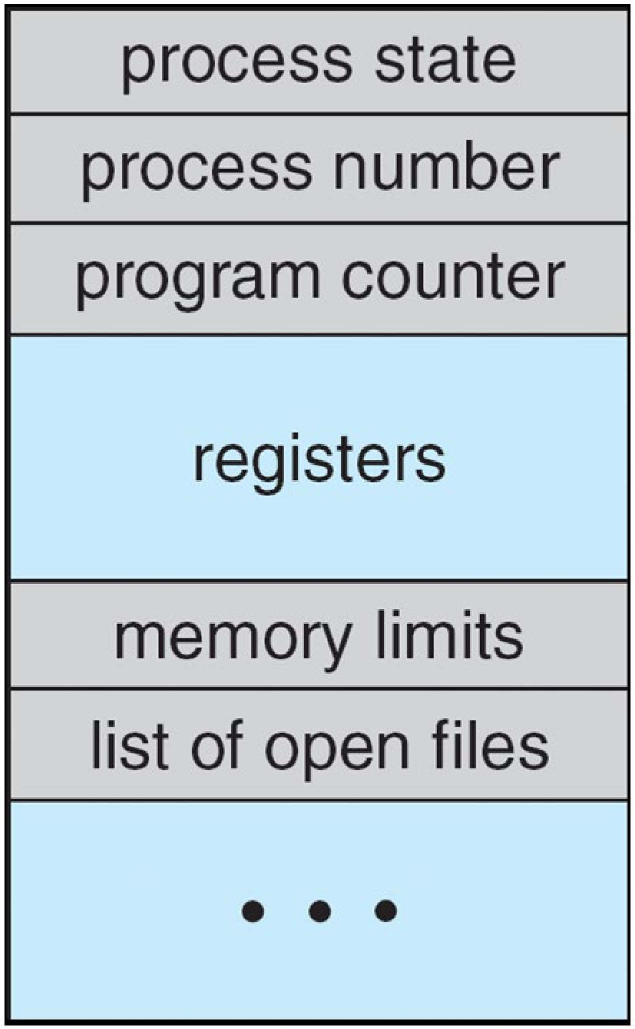


Process Control Block (PCB)

Information associated with each process

- Process state
- Program counter
- CPU registers
- CPU scheduling information
- Memory-management information
- Accounting information
- I/O status information

Process Control Block (PCB)



Process Scheduling Queues

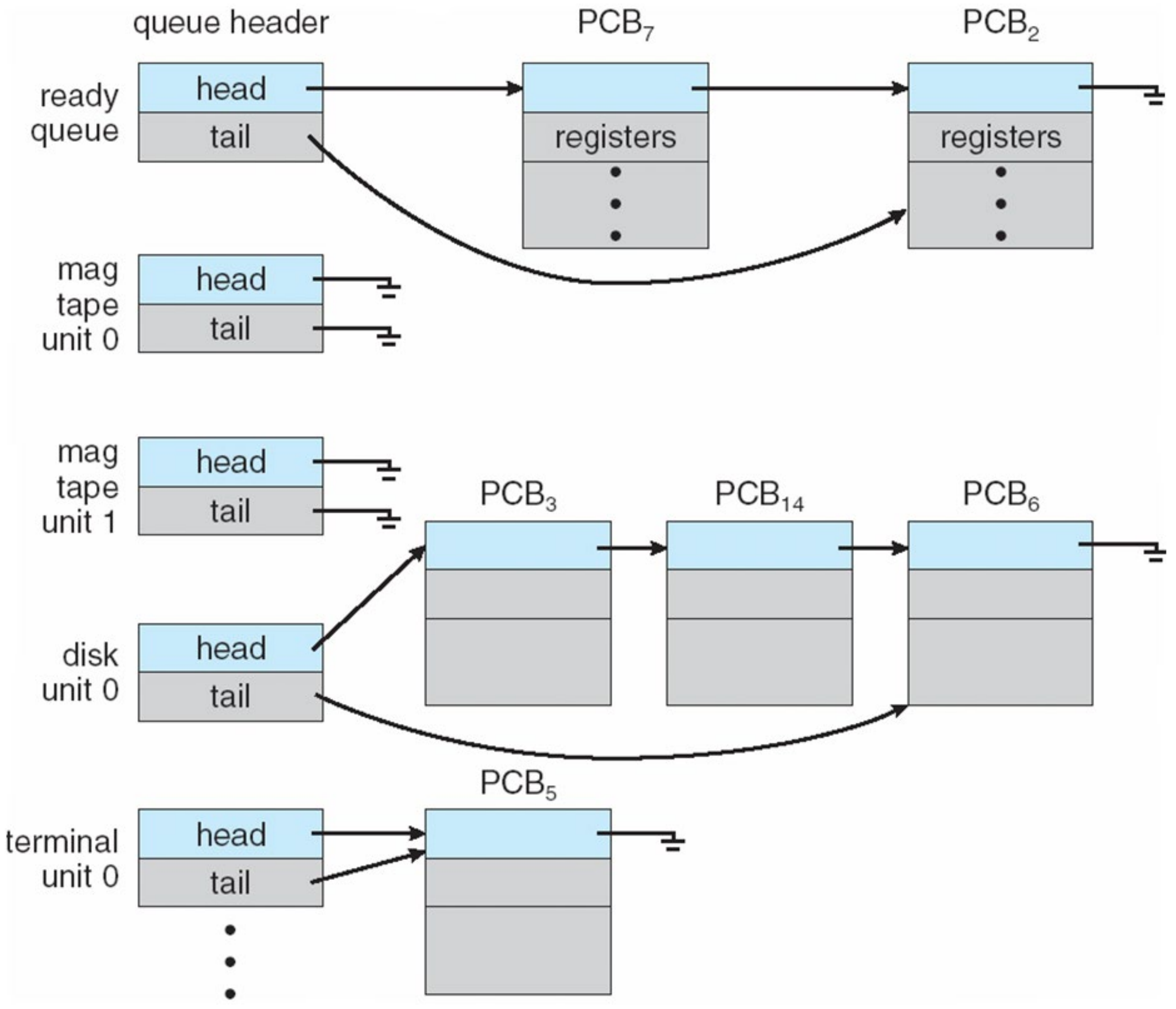
Job queue – set of all processes in the system

Ready queue – set of all processes residing in main memory, ready and waiting to execute

Device queues – set of processes waiting for an I/O device

Processes migrate among the various queues

Ready Queue And Various I/O Device Queues



Representation of Process Scheduling

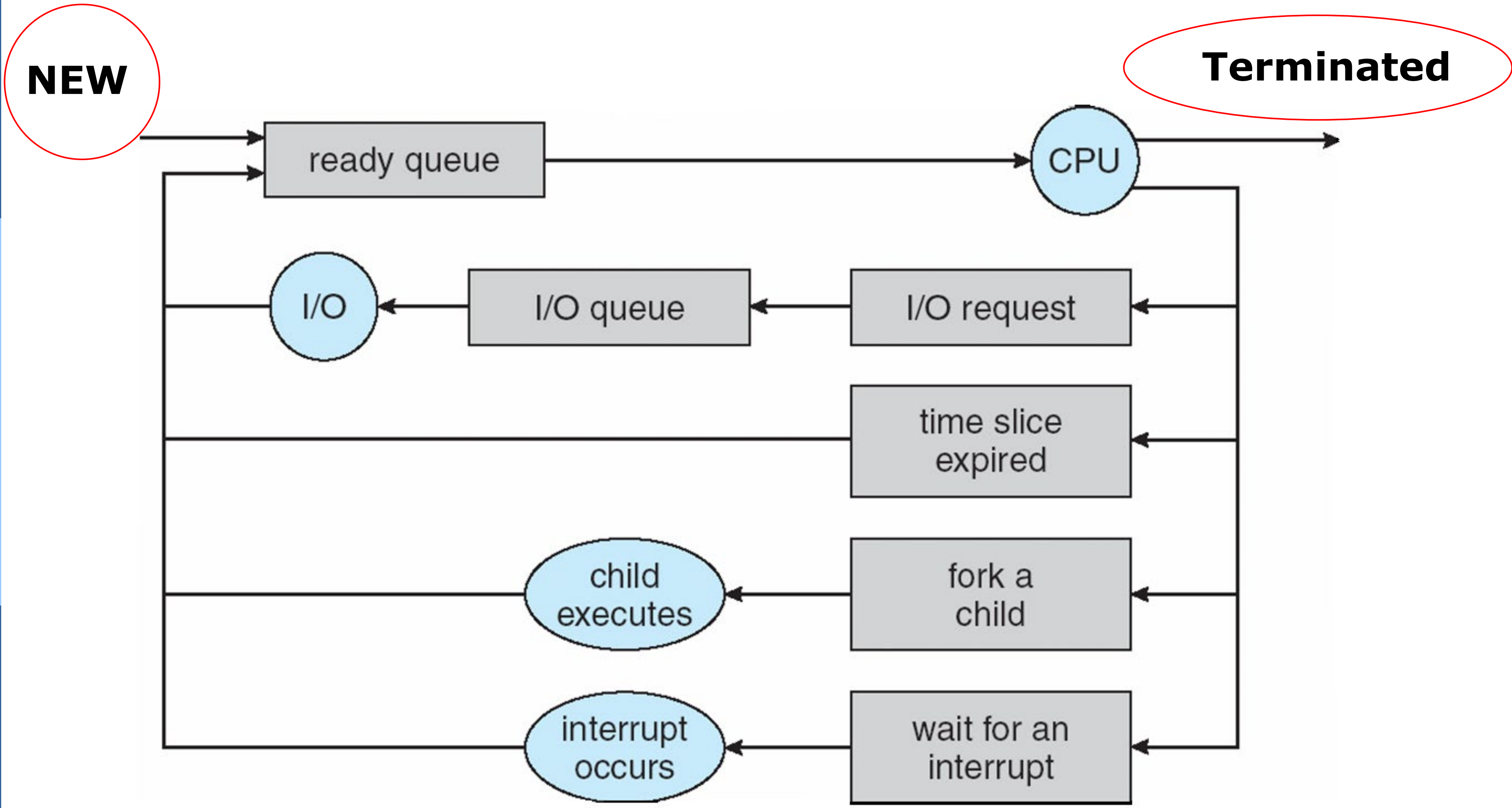
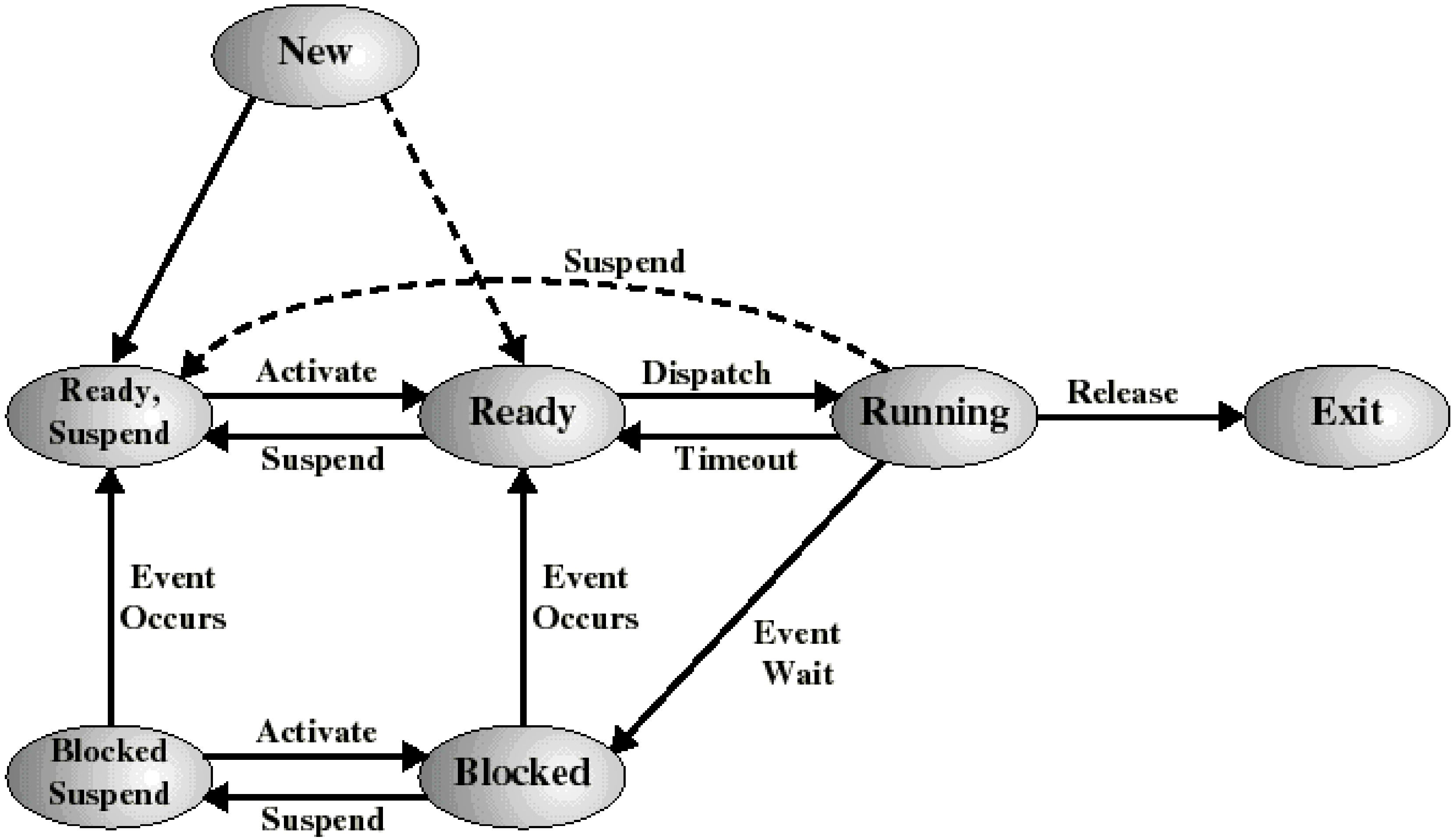


Diagram of Process State



Processes Overheads

A full process includes numerous things:

- an address space (defining all the code and data pages)
- OS resources and accounting information
- a “thread of control”,
  - defines where the process is currently executing
  - That is the PC and registers

- Creating a new process is costly
  - all of the structures (e.g., page tables) that must be allocated
- Communicating between processes is costly
  - most communication goes through the OS

Overview

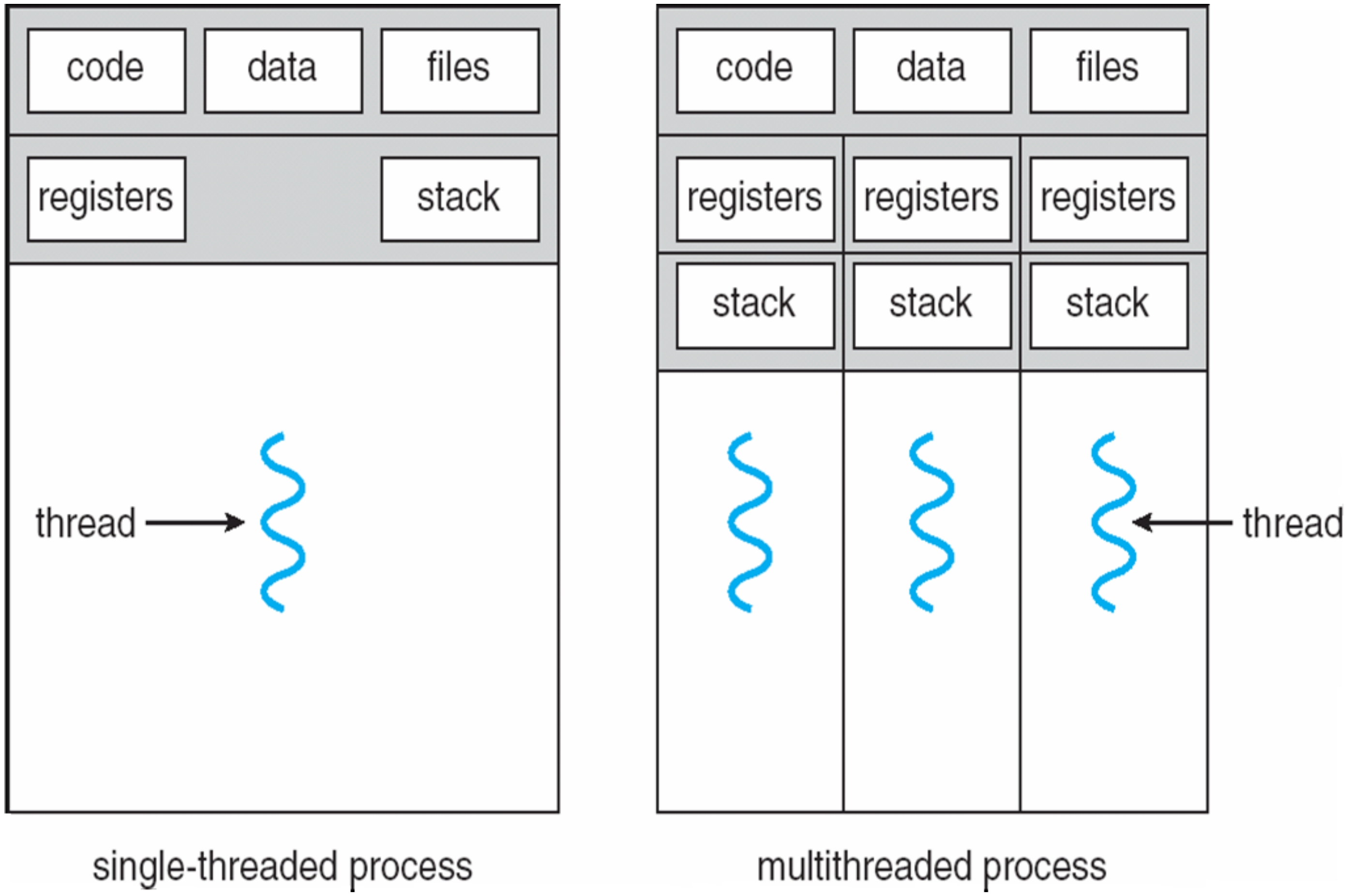
A *thread* (or *lightweight process*) is a basic unit of CPU utilization

- program counter
- register set
- stack space

A thread shares with its peer threads its:

- code section
- data section
- operating-system resources, such as open files and signals
- collectively known as a task.

Single and Multithreaded Processes



Implementation of thread

Kernel-supported threads

Supported directly by the operating system

User-level threads

supported above the kernel, via a set of library calls at the user level

Multithreading Models

Many-to-One

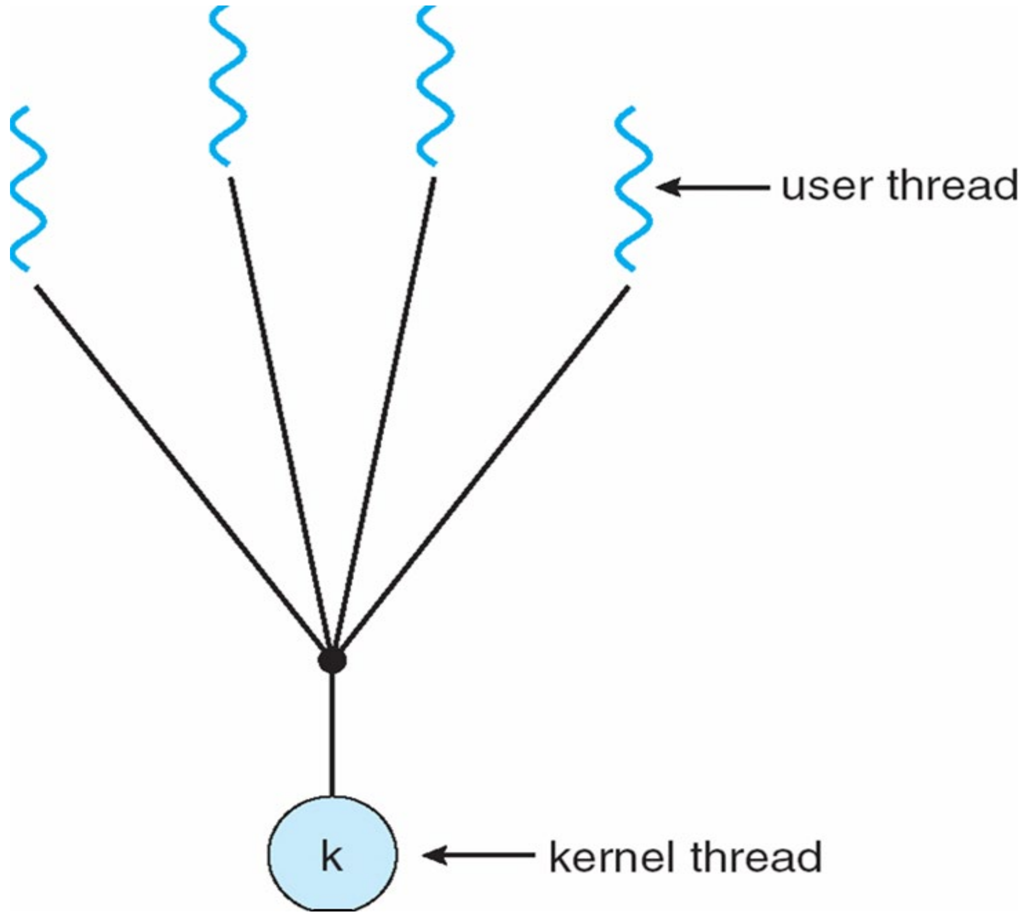
One-to-One

Many-to-Many

Two level model

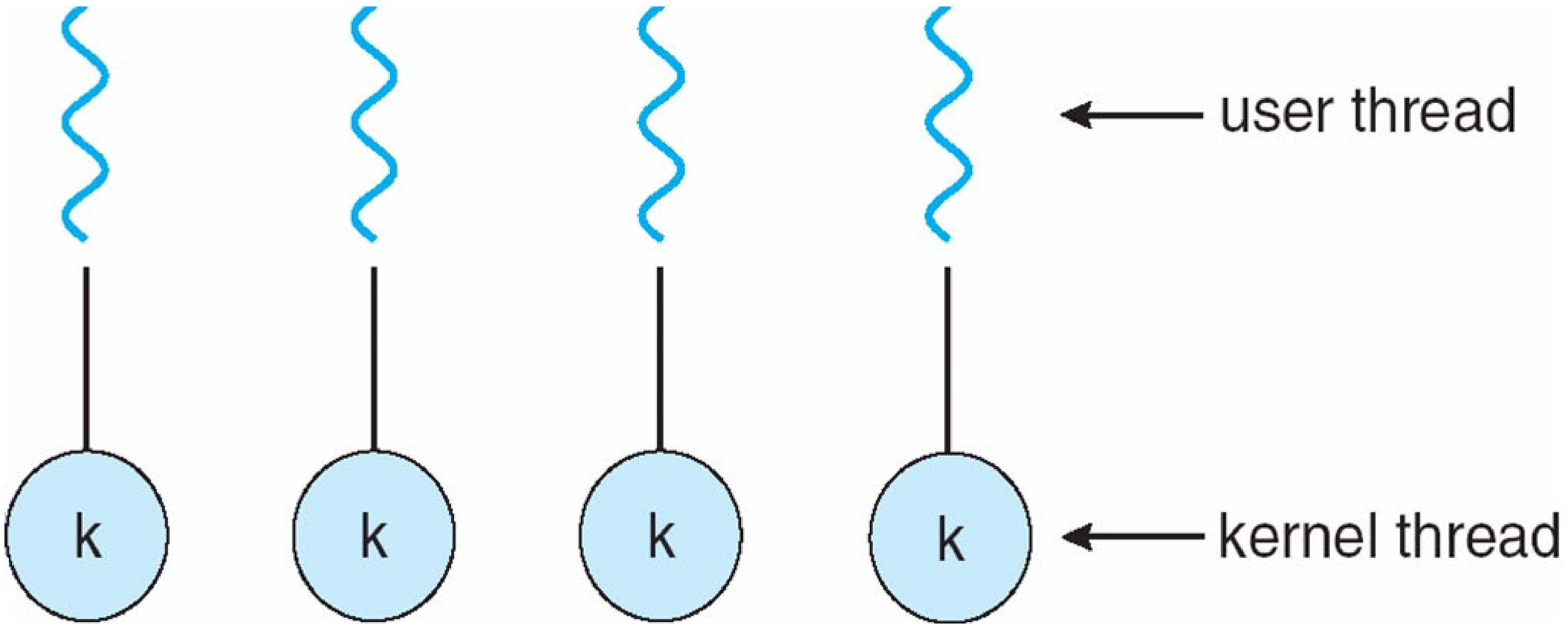
Many-to-One Model

Many user-level threads mapped to single kernel thread



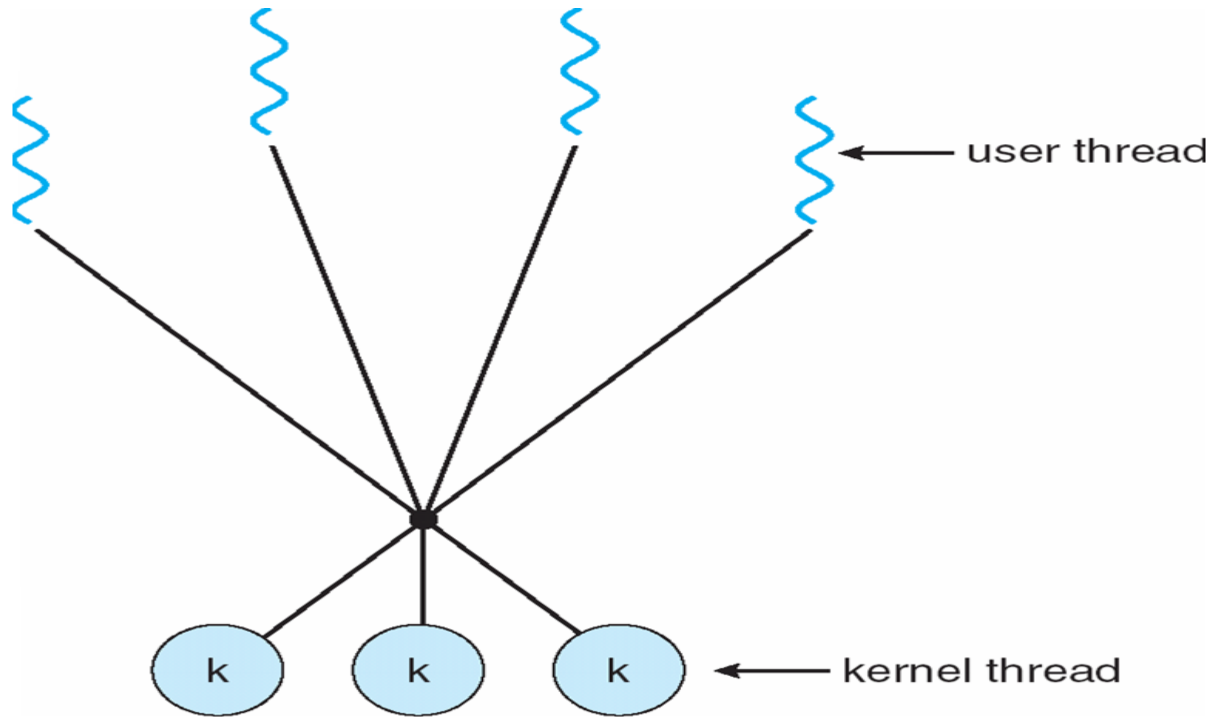
- No parallel execution of threads - can't exploit multiple CPUs
- All threads block when one uses synchronous I/O

One-to-one Model



- More concurrency
- Better multiprocessor performance
- Each user thread requires creation of kernel thread
- Each thread requires kernel resources; limits number of total threads

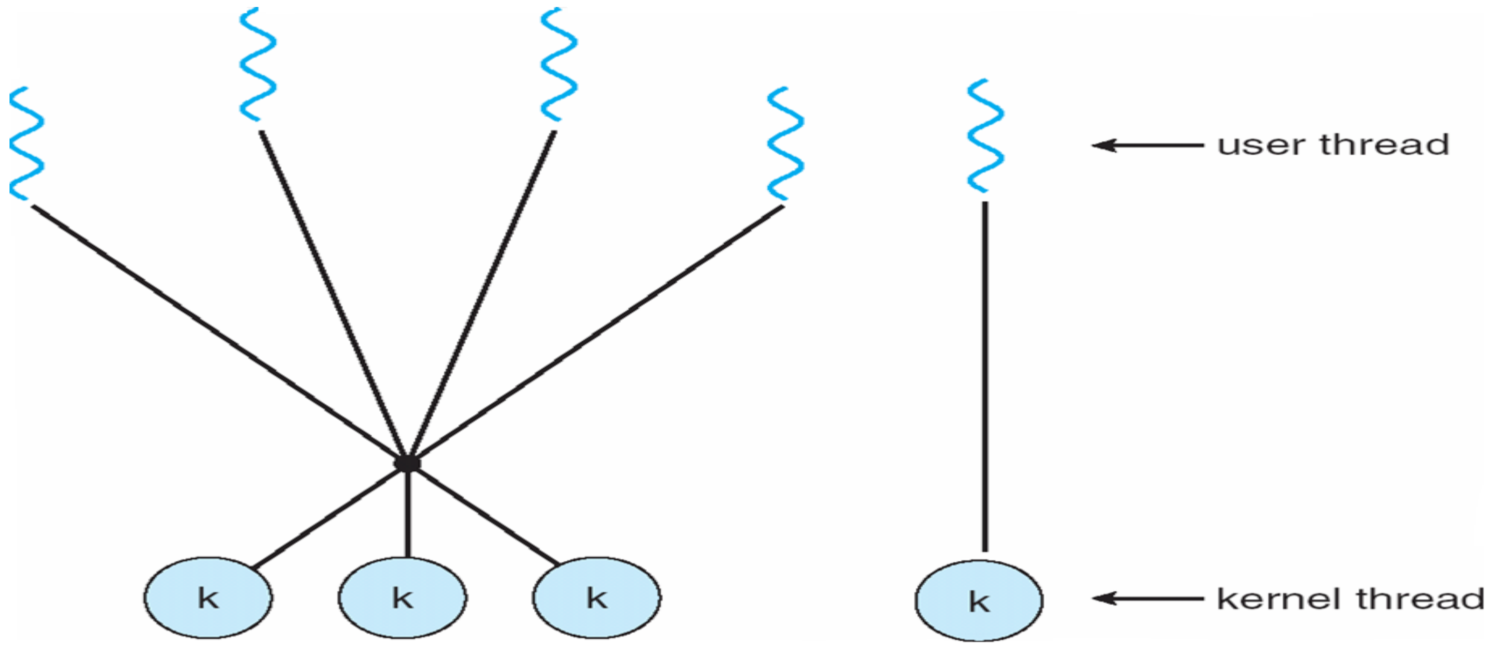
Many-to-Many Model



If U<k? No benefits of multithreading
If U>k, some threads may have to wait for an Kthread to run
- Active thread - executing on an Kthread
- Runnable thread - waiting for an Kthread

A thread gives up control of Kthread under the following:
– synchronization, lower priority, yielding, time slicing

Two-level Model

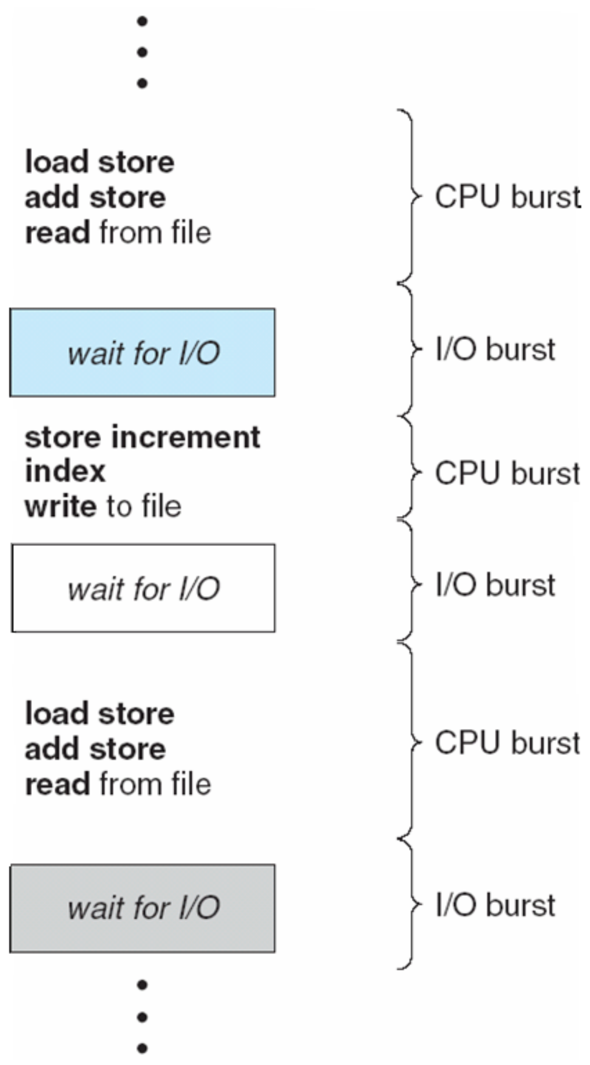


- Supports both bound and unbound threads
  - Bound threads - permanently mapped to a single, dedicated kthread
  - Unbound threads - may move among kthreads in set
- Thread creation, scheduling, synchronization done in user space

CPU-I/O Burst Cycle

- CPU–I/O Burst Cycle – Process execution consists of a cycle of CPU execution and I/O wait.
- CPU burst distribution
  - Usually, a large number of short CPU bursts and a small number of long CPU bursts
  - An I/O-bound program typically has many short CPU bursts
  - A CPU-bound program might have a few long CPU bursts.

Alternating Sequence of CPU And I/O Bursts



2. 进程调度的功能

① 记录进程状态;

② 从就绪进程选取一个进程;

③ 实施进程上下文切换。

3. 引起进程调度的时机

（1）正在执行的进程执行完毕。这时，如果不选择新的就绪进程执行，将浪费处理机资源。

（2）执行中进程自己调用阻塞原语将自己阻塞起来进入睡眠等待状态。

（3）执行中进程调用了P原语操作，从而因资源不足而被阻塞；或调用了V原语操作激活了等待资源的进程队列。

（4）执行中的进程提出I/O请求后被阻塞。

（5）在分时系统中时间片已经用完。

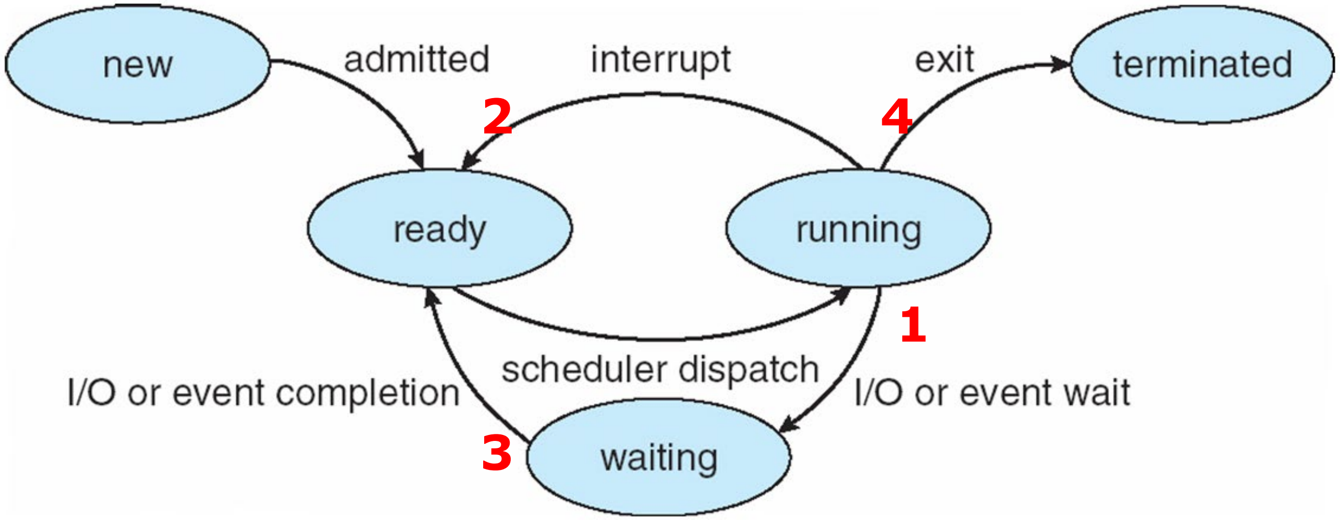
（6）在执行完系统调用，在系统程序返回用户进程时，可认为系统进程执行完毕，从而可调度选择一新的用户进程执行。

（7）就绪队列中的某进程的优先级变得高于当前执行进程的优先级，从而也将引发进程调度。

CPU Scheduler

CPU scheduling decisions may take place when a process:

1. Switches from running to waiting state
2. Switches from running to ready state
3. Switches from waiting to ready
4. Terminates



Scheduling under 1 and 4 is **nonpreemptive**

All other scheduling is **preemptive**

CPU Scheduling scheme

Nonpreemptive Scheduling

- Once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state.

Preemptive scheduling

- According to some rule, the CPU utilization of a process can be preempted .
  - Timer
  - Priority

Scheduling Criteria

CPU utilization – keep the CPU as busy as possible

Throughput – # of processes that complete their execution per time unit

Turnaround time – amount of time to execute a particular process

Waiting time – amount of time a process has been waiting in the ready queue

Response time – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

Scheduling Algorithm Optimization Criteria

Max CPU utilization

Max throughput

Min turnaround time

Min waiting time

Min response time

Fairness: everyone makes progress, no one starves

"The perfect CPU scheduler"

Scheduling Algorithm

FCFS

SJF

Priority scheduling

Round Robin (RR)

Multilevel Queue

Multilevel Feedback Queue

4.　算法性能衡量

平均周转时间 T＝ $\frac{1}{n}\sum_{i=1}^{n} T_i$

其中 $T_i = T_{完成} - T_{提交}$

平均带权周转时间 W＝ $\frac{1}{n}\sum_{i=1}^{n} W_i$

其中 W_i＝周转时间/运行时间

按FCFS算法计算周转时间

作业号	提 交 时 间	执 行 时 间	开 始 时 间	完 成 时 间	周 转 时 间	带 权 周 转
1	10.00	2.00	10.00	12.00	2.00	1.00
2	10.10	1.00	12.00	13.00	2.90	2.90
3	10.25	0.25	13.00	13.25	3.00	12.00

平均周转时间 T=(2.00+2.90+3.00）/3=2.63小时

平均带权周转时间 W=(1.00+2.90+12.00)/3=5.30小时

按SJF算法计算周转时间

作 业 号	提 交 时 间	执 行 时 间	开 始 时 间	完 成 时 间	周 转 时 间	带 权 周 转
1	10. 00	2. 00	10. 00	12. 00	2. 00	1. 00
2	10. 10	1. 00	12. 25	13. 25	3. 15	3. 15
3	10. 25	0. 25	12. 00	12. 25	2. 00	8. 00

平均周转时间T =（2. 00+3. 15+2. 00）/3=2. 38小时

平均带权周转时间W=(1. 00+3. 15+8. 00)/3=4. 05小时

在一个批处理系统中，有两个作业进程。有一作业序列，其到达时间及估计运行时间如下表

作业	到达时间	估计运行时间（分钟）
1	10：00	35
2	10：10	30
3	10：15	45
4	10：20	20
5	10：30	30

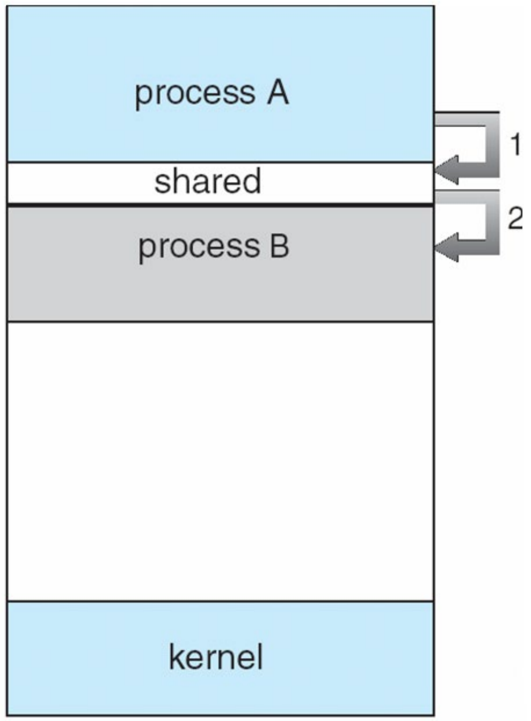
系统采用最高响应比优先的作业调度算法（响应比＝等待时间/估计运行时间）。作业进程的调度采用短作业优先的抢占调度算法。

1、列出各作业的执行时间

2、计算这批作业的平均周转时间

Background

Example： Suppose that we wanted to provide a solution to the consumer-producer problem that fills all the buffers. We can do so by having an integer count that keeps track of the number of full buffers. Initially, count is set to 0. It is incremented by the producer after it produces a new buffer and is decremented by the consumer after it consumes a buffer.

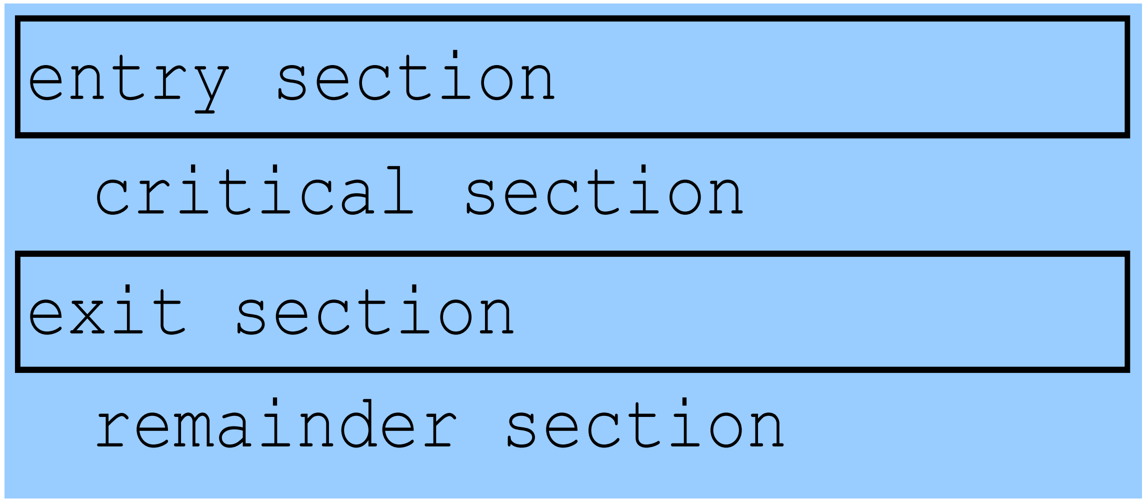


Critical-Section

To ensure the shared data to be modified mutual excluded, no more than one process can execute in its critical section at the same time.

Each process must request permission to enter its critical section----entry section (进入区)

And exit with the exit section (退出区).



7. 临界区与管理原则

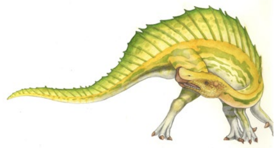
临界区：不允许多个并发进程交叉执行的程序段。

管理原则

① 每次至多一个进程进入临界区；

② 进程不能无限期留在临界区；

③ 进程不能相互阻塞。



Solution to Critical-Section Problem

1. Mutual Exclusion 互斥 (safety) - No more than one process can be in a critical section at any time.
2. Progress 有空让进 (liveness) - If no process is executing in its critical section and there exist some processes wishing to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely
3. Bounded Waiting 有限等待- A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted
   - Assume that each process executes at a nonzero speed
   - No assumption concerning relative speed of the N processes

   Ideally we would like fairness as well

   If two threads are both trying to enter a critical section, they have equal chances of success

   … in practice, fairness is rarely guaranteed

临界区小结

- 锁是解决临界区问题的常用方法：
  - 互斥可以使用锁来实现
  - 通常需要一定等级的硬件支持
- 常用的三种实现方法：
  - 禁用中断（仅限于单处理器）
  - 软件方法（复杂）
  - 硬件方法（原子操作指令，单处理器或多处理器均可）
- 可选的实现内容：
  - 有忙等待
  - 无忙等待

9.　进程的同步与互斥

进程同步　相互合作的并发进程之间在某些点要相互通信，互相协调，共同完成任务的过程。

进程互斥　不允许两个或两个以上的并发进程同时进入临界区。

Semaphore (Cont.)

Counting semaphore – integer value can range over an unrestricted domain

Binary semaphore – integer value can range only between 0 and 1

- Same as a **mutex lock**

Can implement a counting semaphore ***S*** as a binary semaphore

With semaphores we can solve various synchronization problems

10.　信号量与PV原语

信号量（Semaphore）表示系统中资源实体数目或资源使用情况的整型量，其值只能由PV原语操作改变。

n个进程共享m个资源，信号量变化范围

$$(m-n) \leqslant S \leqslant m$$

P(S)：代表申请使用资源的操作

S＝S-1；

若S＜0，则该进程被阻塞后与该信号相对应的对列中，然后转进程调度；

若S≥0 ，则，调用P(S)原语的进程继续运行。

V(S)：代表释放归还资源的操作

S＝S+1；

若S≤0，则唤醒一个等待S的进程后，调用P(S)原语的进程继续运行或转进程调度；

若S＞0，则，调用V(S)原语的进程继续运行。

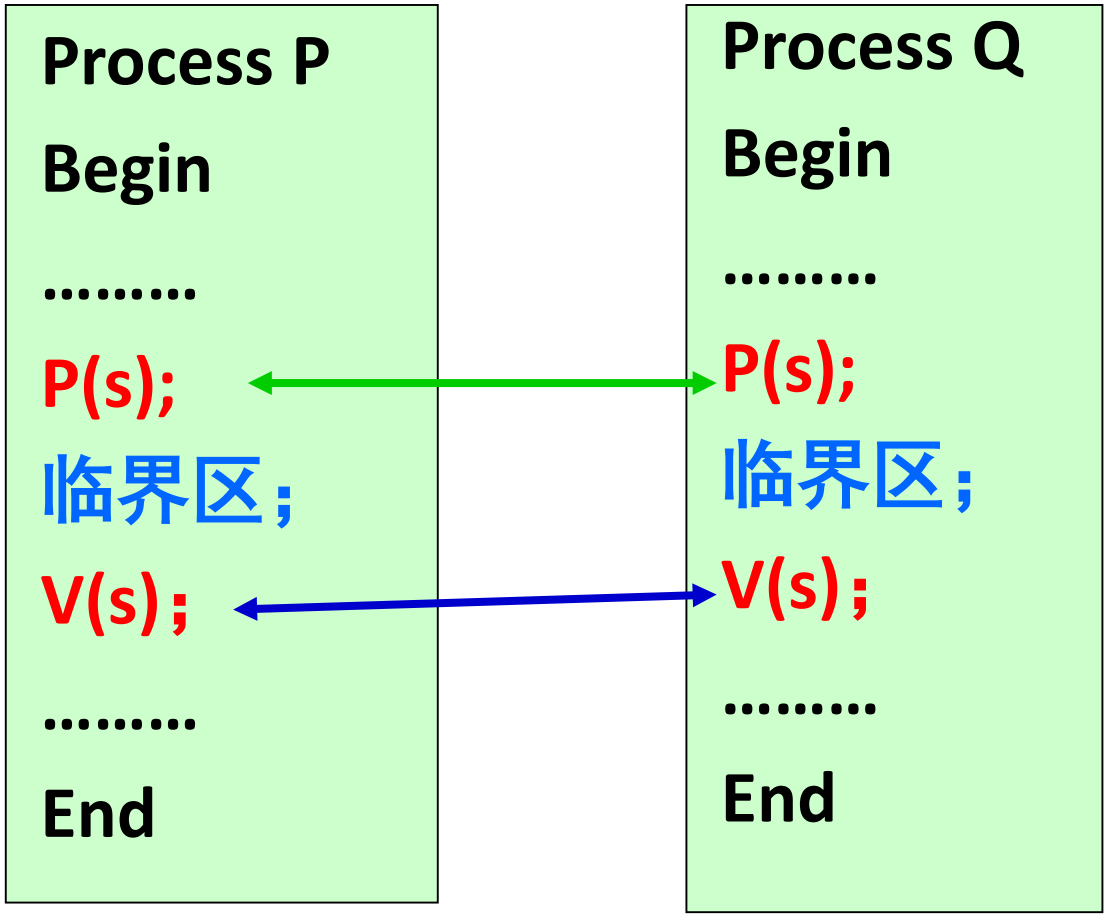
11. 进程并发执行的描述

主程序

```
Begin
… , s: semaphore;    /* 定义信号量 */
…; s=XXX;            /* 赋初值 */
 COBEGIN
     Process P1;      /*并发进程  */
     process p2;
     ……………
  COEND
End
```

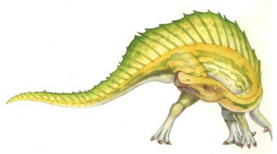
12. PV原语实现进程互斥

设公用信号量s，初值为1（或k）



```
Process P
Begin
………
P(s);
临界区;
V(s);
………
End
```

```
Process Q
Begin
………
P(s);
临界区;
V(s);
………
End
```



PV原语实现进程互斥例子

设进程P、Q共享一台打印机，打印机任何时刻只能被一个进程使用，不能同时使用。

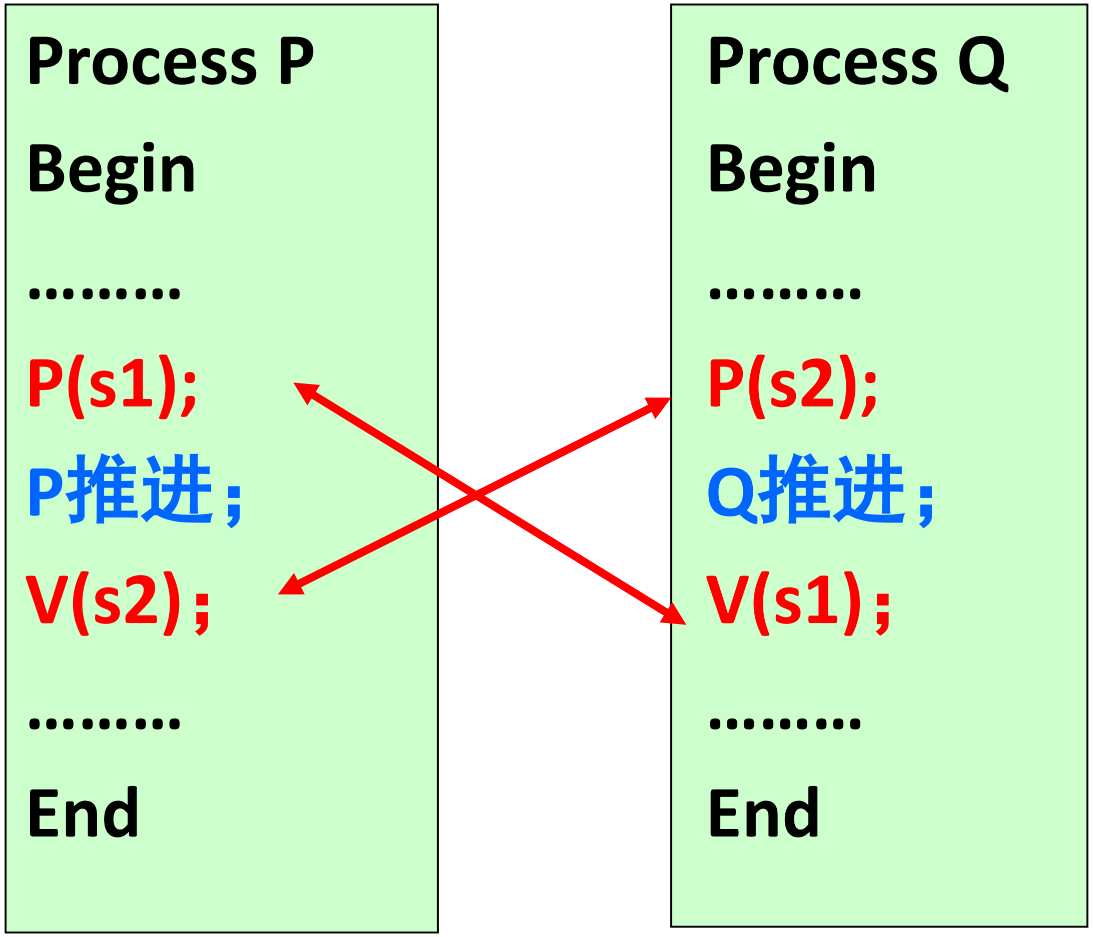
设公用信号量S，初值为1。

```
Process P()
Begin
P(s);
进程P使用打印机;
V(s) ;
End
```

```
Process Q()
Begin
P(s);
进程Q使用打印机;
V(s) ;
End
```

13. PV原语实现进程同步

分别设私用信号量s1，初值为1（或k）； s2，初值为0



```
Process P
Begin
………
P(s1);
P推进；
V(s2)；
………
End
```

```
Process Q
Begin
………
P(s2);
Q推进；
V(s1)；
………
End
```

PV原语实现进程同步例子

现有2个进程R、W，它们共享可以存放一个数的缓冲区Buf。进程R每次读入一个数存放到Buf中，由进程W打印输出。

设私用信号量s1,初值为1， s2,初值为0 。

```
Process R()
Begin
L1:读一个数;
P(s1);
Buf=数；
V(s2)；
Goto L1;
End
```

```
Process W()
Begin
L2:P(s2);
打印Buf中的数；
V(s1)；
Goto L2;
End
```

Monitors

A programming language construct

A collection of procedures, variables, data structures

Access to it is guaranteed to be exclusive.

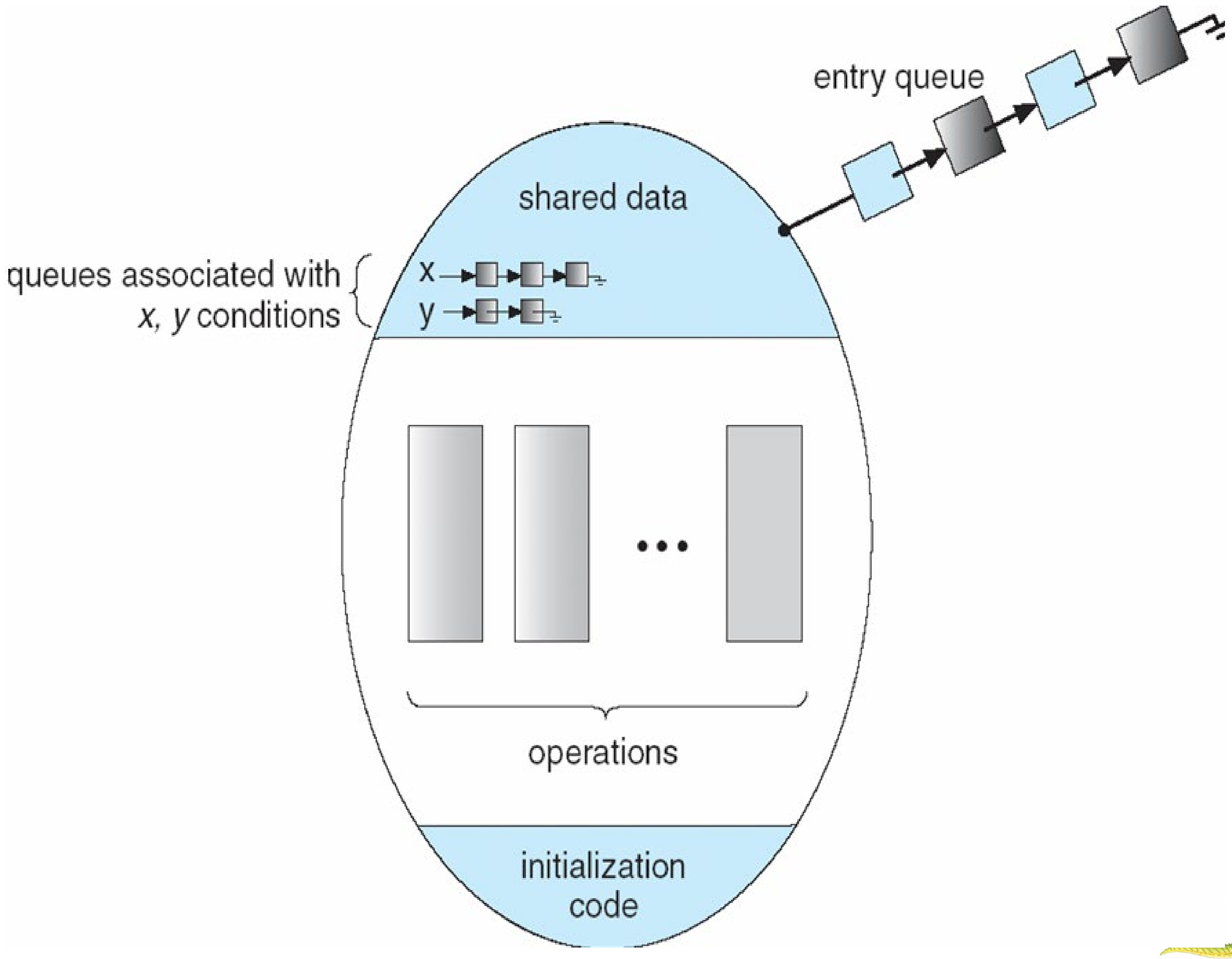
- By the compiler (not programmer)

Monitors use separate mechanisms for the two types of synchronization

- use locks for mutual exclusion
- use condition variables for ordering constraints

monitor = a lock + the condition variables associated with that lock

Monitor with Condition Variables



The Deadlock Problem

A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set.

The resources may be either physical resources or logical resources.

Necessary Conditions for Deadlock

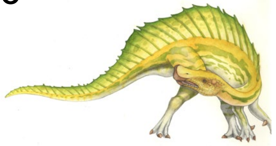
Mutual exclusion: only one process at a time can use a resource

Hold and wait: a process holding at least one resource is waiting to acquire additional resources held by other processes

No preemption: a resource can be released only voluntarily by the process holding it, after that process has completed its task

Circular wait: there exists a set $\{P_0, P_1, \ldots, P_0\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1, P_1 is waiting for a resource that is held by

$P_2, \ldots, P_{n-1}$ is waiting for a resource that is held by P_n, and P_n is waiting for a resource that is held by P_0.



15. 解决死锁的方法

（1） 预防

（2） 避免

（3） 检测与恢复

Deadlock Prevention

Mutual Exclusion

Turn nonsharable to sharable.

- Time-sharing
- Virtual-combination
- Spooling

- 以下两个优先级相同的进程PA和PB在并发执行结束后，x、y和z的值分别为多少（信号量S1和S2的初值均为0）？

PA:	PB:
(1) x = 1;	(1) y = 1;
(2) x = x+1;	(2) y = y+3;
(3) P(S1);	(3) V(S1);
(4) z = x + 1;	(4) z = y + 1;
(5) V(S2);	(5) P(S2);
(6) x = x + z;	(6) y = y + z;

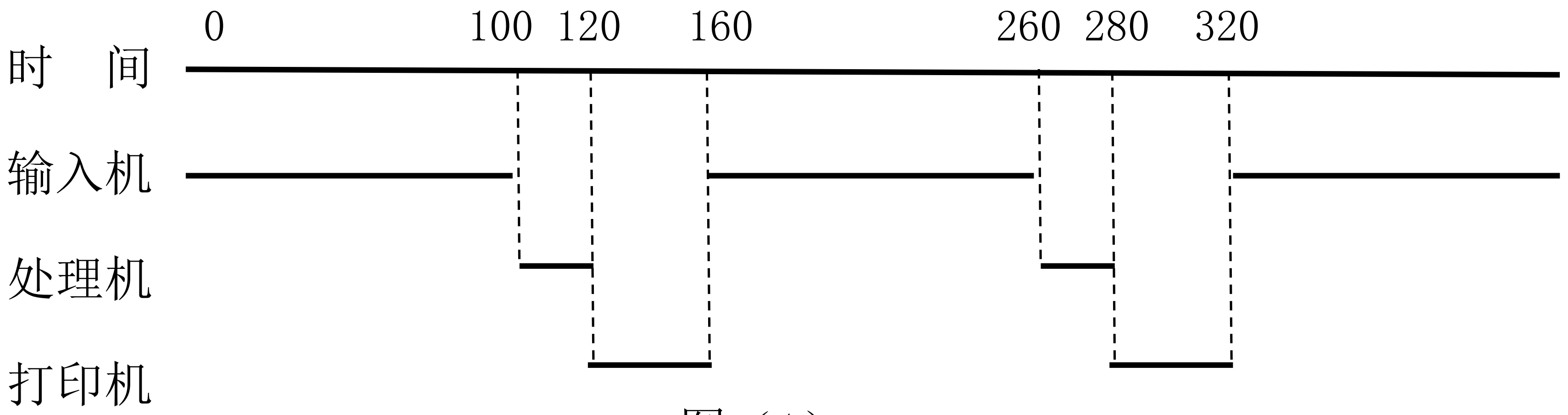
- 以下两个优先级相同的进程PA和PB在并发执行结束后，x、y和z的值分别为多少（信号量S1和S2的初值均为0）？

  - 答：将PA和PB进程分解为以下6个程序段。

```
SA1:  x :  = 1;
      x :  = x + 1;
SA2:  z :  = x + 1;
SA3:  x :  = x + z;
SB1:  y :  = 1;
      y :  = y + 3;
SB2:  z :  = y + 1;
SB3:  y :  = y + z;
```

- SA1与SB1可以并发执行，SA3与SB3可以并发执行。SA2与SB2因变量交集不为空，因而不能并发执行。因此SA2与SB2的执行顺序不同，可能会有不同的结果。
- 先执行SA2，则x=7, y=9, z=5；
- 先执行SB2，则x=5, y=7, z=3。

若某计算问题的执行情况如图（1）所示。



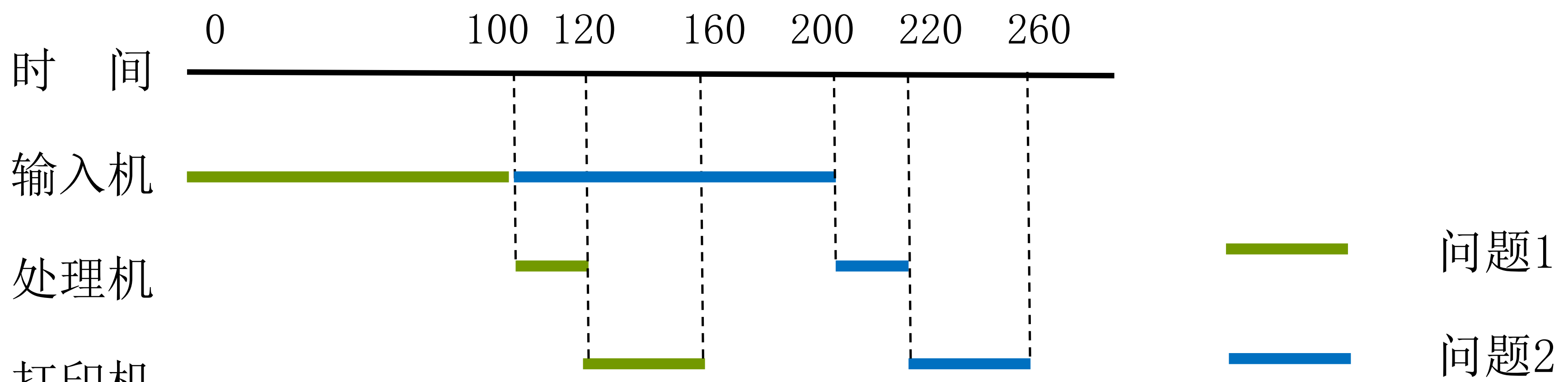
图（1）

则请回答下列问题。

1. 叙述该计算问题中处理机、输入机和打印机是如何协同工作的。
2. 计算在图（1）所示执行情况下处理机的利用率。
3. 简述处理机利用率不高的原因。
4. 请画出能提高处理机利用率的方案。

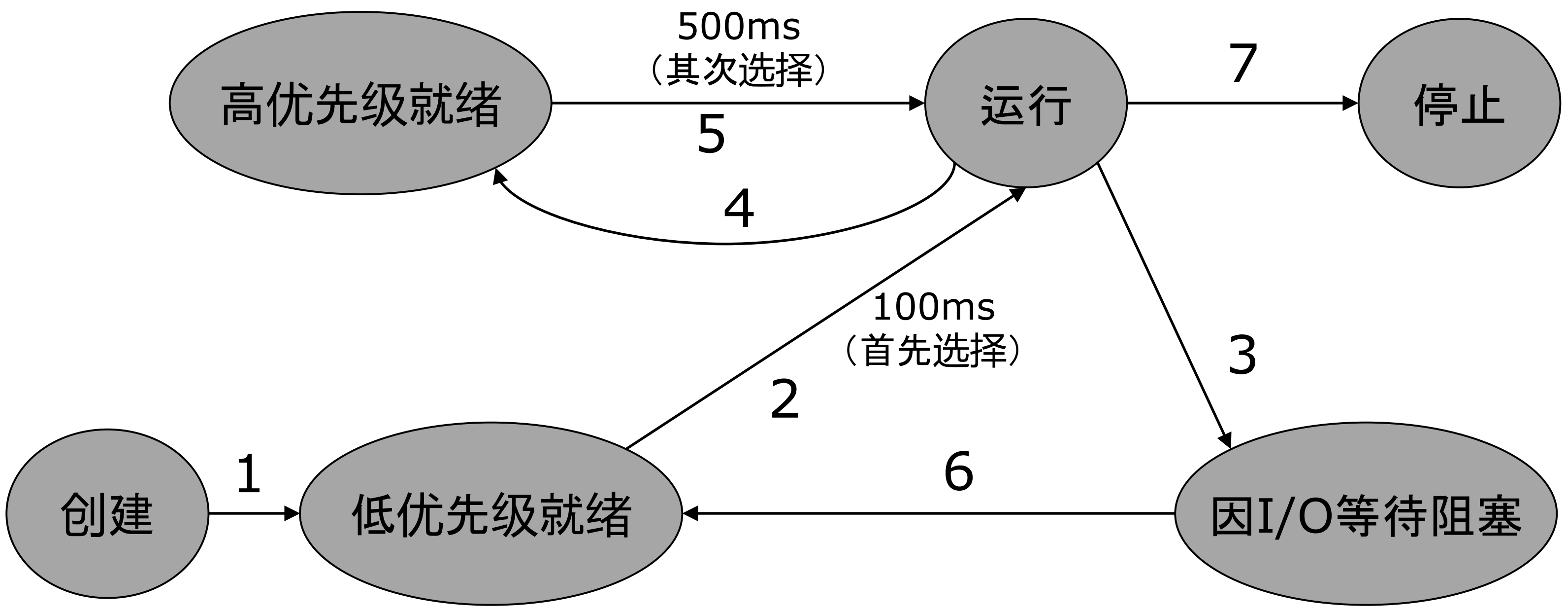
解答：

1. 处理机、输入机和打印机是按照输入→处理→打印的顺序依次执行的，输入机为处理机提供数据，处理机得到数据后进行处理，处理结果通过打印机输入。输入机读取一批数据，花费时间为100；处理机对这批数据进行计算，花费时间为20；打印机打印计算结果，花费时间为40。
2. 处理机的利用率=[20/(100+20+40)]×100%=12.5%
3. 当一道程序在运行中发出I/O请求后，处理机只能处于等待状态，即必须等I/O完成后才能继续运行，因此处理机会长时间处于空闲状态，这会导致其利用率不高。
4. 采用多道程序设计技术使处理机、输入机和打印机并行工作，可以提高处理机的利用率，如图（2）所示。



图（2）

- 某系统的进程状态图如下图所示：
  - （1）说明一个进程发生变迁3、4、6的原因。



➢ 解答：发生变迁3的原因是运行进程提出I/O请求。发生变迁4是运行进程的时间片到。变迁6是阻塞进程因所等待的I/O操作完成而被I/O中断服务程序唤醒。

- 某系统的进程状态图如下图所示：
  - （2）下述因果变迁是否会发生？若会，在什么情况下发生？
    - ①3->5；　②6->4；　③6->7

- 解答：①因果变迁3->5会发生。当某个运行进程由于I/O请求二阻塞时，如果此时系统中不存在高优先级就绪进程而只存在低优先级就绪进程，则系统将选择一低优先级就绪进程投入运行。

- 某系统的进程状态图如下图所示：
  - （2）下述因果变迁是否会发生？若会，在什么情况下发生？
    - ①3->5；　②6->4；　③6->7

➢ 解答：②因果变迁6->4不会发生。发生4的原因只可能是时间片用完。

➢ 解答：③因果变迁6-7不会发生。发生7的原因是进程在时间片用完前运行结束。

- 某系统的进程状态图如下图所示：
  - （3）根据此进程图，说明该系统的CPU调度策略和调度效果。

➢ 解答：该系统的进程调度策略是多队列调度策略，它将系统中的所有进程按照优先级的高低组织成高优先级和低优先级两个队列，两个队列均采用时间片轮转调度算法，但高优先级队列的时间片较短（为100ms），低优先级队列的时间片较长（为500ms），系统总是先调度高优先级队列上的进程，仅当该队列为空时，才调度低优先级队列上的进程。

- 某系统的进程状态图如下图所示：
  - （3）根据此进程图，说明该系统的CPU调度策略和调度效果。

➢ 解答：这种调度策略不仅体现了短进程优先的思想，同时也不会使长进程长期得不到处理，而且它还能较好地满足I/O型作业的需要。对每个进程来说，它总是先进入高优先级就绪队列，如果进程运行时间较短，则它便能以较小的周转时间完成；如果进程运行时间较长，则在运行100ms后它将进入低优先级就系队列，而且当再次调度到该进程时，它的时间片将增大到500ms。

- 某系统的进程状态图如下图所示：
  - （3）根据此进程图，说明该系统的CPU调度策略和调度效果。

- 解答：如果是I/O型的进程，由于每次完成输入输出工作后，它将进入高优先级就绪队列，因此将被优先调度到，这将有效提高系统中I/O设备的利用率和系统的吞吐量。

End of Exercise Lesson