Operating System Review Problems

1. What is a process？ Please draw a process state transition diagram with nine states in Unix SVR4?

   •A program in execution（一个正在执行的程序）

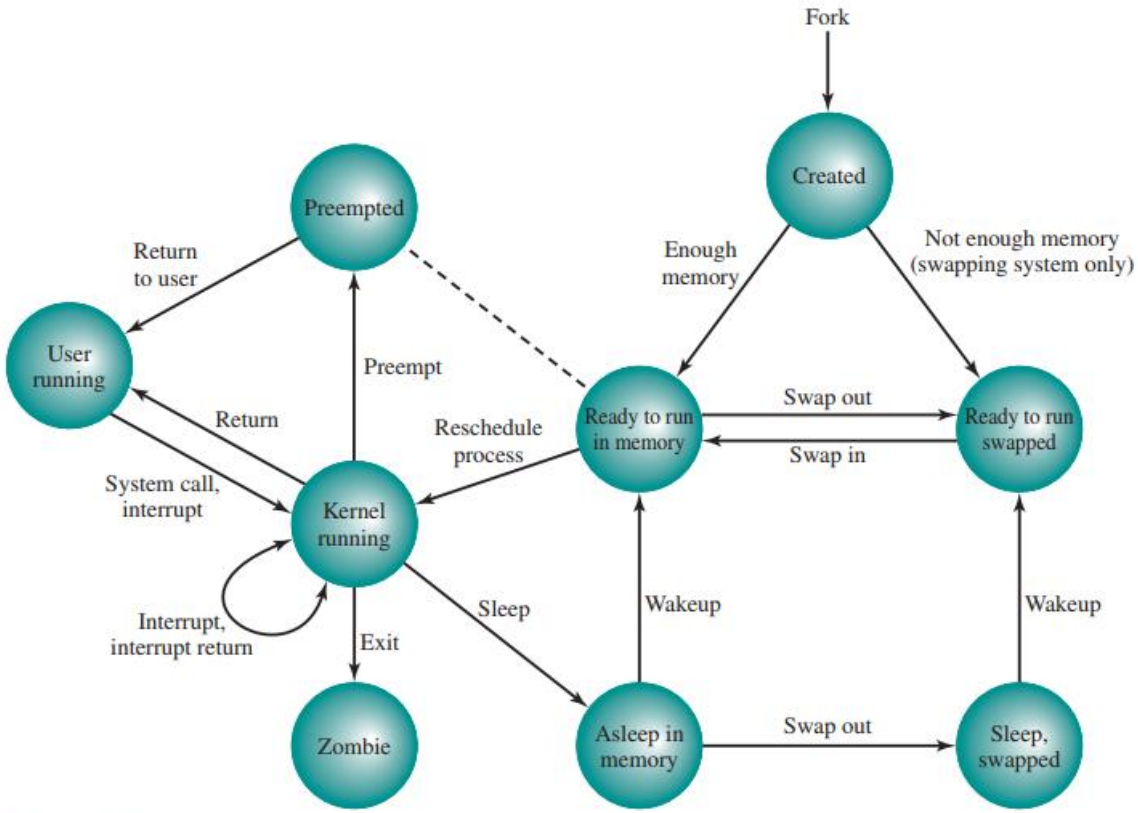


Figure 3.17 **UNIX Process State Transition Diagram**

Table 3.9 UNIX Process States

User Running	Executing in user mode.
Kernel Running	Executing in kernel mode.
Ready to Run, in Memory	Ready to run as soon as the kernel schedules it.
Asleep in Memory	Unable to execute until an event occurs; process is in main memory (a blocked state).
Ready to Run, Swapped	Process is ready to run, but the swapper must swap the process into main memory before the kernel can schedule it to execute.
Sleeping, Swapped	The process is awaiting an event and has been swapped to secondary storage (a blocked state).
Preempted	Process is returning from kernel to user mode, but the kernel preempts it and does a process switch to schedule another process.
Created	Process is newly created and not yet ready to run.
Zombie	Process no longer exists, but it leaves a record for its parent process to collect.

2. Illustrated some scenarios where process switch occur?

   ① Clock interrupt（时钟中断）

Process has executed for the maximum allowable time slice

② ***I/O interrupt（I/O 中断）***

③ ***Memory fault（内存失效）***

Memory address is in virtual memory so it must be brought into main memory

④ ***Trap（陷阱）***

Error or exception occurred

May cause process to be moved to edit state

⑤ ***Supervisor call（系统调用）***

Such as file open

3. Illustrated some scenarios where process should blocked?

① ***The system fails to apply for resources.*** ***（向系统申请资源时失败）***

For example, if A process (A) requests A printer but the printer is in use by another process (B), process A is blocked.

② ***Waiting for an operation.（等待某种操作）***

Process A starts an I/O device. If process A can start an I/O device only after the specified I/O task is complete, process A automatically blocks after starting the I/O device.

③ ***New data has not arrived.（新数据尚未到达）***

For cooperative processes, if a process needs to obtain data from another process before it can process the data, it will enter the blocking state as long as the data has not arrived.

④ ***Waiting for a new task to arrive.（等待新任务的到达）***

Each time the process completes its task

4. Suppose there are three processes running in an operating system with multithreading as following:

Process ID	Threads
A	T11, T12
B	T21, T22
C	T31, T32

If the multithreading is supported by user-level implementation（用户级）and thread T11 is blocked, which thread may be scheduled to run next?

One of T21,T22,T31,T32

If the multithreading is supported by kernel-level implementation（内核级）, which thread may be scheduled to run next?

One of T12,T21,T22,T31,T32 原因：内核级的实现，OS 才知道 A 中有 T12

5. What is race condition? Give a program implementation to demonstrate the mutual exclusion enforced by Test and Set (or exchange) instruction or by

semaphore?

*A race condition occurs when multiple processes or threads **access shared resource** so that the final result depends on **the order of execution of instructions** in the multiple processes.*

*竞争条件是在多个进程或线程**访问共享资源**时，其最终结果依赖于多个进程的**指令执行顺序**。*

用信号量实现互斥：

```
semaphore mutex = 1;
void P(int i = 0)
{
    while (true) {
        semWait(mutex);
        /*临界区critical section*/
        semsignal(mutex);
        /*其余部分remainder*/
    }
}
void main()
{
    parbegin(P(1), P(2)...P(n));
}
```

Wait 用法：

wait(num),num 是目标参数，wait 的作用是使其（信息量）减一。

如果信息量>=0，则该进程继续执行；否则该进程置为等待状态，排入等待队列。

signal用法：

signal(num),num 是目标参数，signal 的作用是使其（信息量）加一。

如果信息量>0，则该进程继续执行；否则释放队列中第一个等待信号量的进程。

用 TS 实现互斥：

```
Lock DB 0
...
entercritical:
    TS REG, Lock
    CMP REG, 0
    JNZ entercritical
    ...

enxitcritical:
        MOV Lock, 0
```

CMP 指令:进行比较两个操作数的大小

JNZ:依据 ZF 标志位（我经常叫做 零标志位，判断是不是 0） 不为 0/不等于时跳转执行

MOV 指令:是数据传送指令,用于将一个数据从源地址传送到目标地址(移动/赋值)

用 exchange 实现互斥：

```
Lock DB 0
...
entercritical:
    MOV reg, !=0
    Xchg Lock,reg
    CMP REG, 0
    JNZ entercritical
    ...

enxitcritical:
        MOV Lock, 0
```

XCHG 指令:交换两个操作数内容

6. Consider the following program:

```
boolean blocked[2];
int turn;
void P(int id)
{
   while (true) {
      blocked[id] = true;
      while (turn != id){
         while (blocked[1-id])   /* do nothing */;
         turn = id; //循环结束，则某一进程进入临界区
      }
      /* critical section */
      blocked[id] = false;
      /* remainder */
   }
}
void main()
{
   blocked[0] = false;
   blocked[1] = false;
   turn = 0;
   parbegin(P(0), P(1));
}
```

This software segment is the solution to the mutual exclusion problem for two processes. Find a counterexample(反例) that demonstrates that this solution is incorrect.

考虑进程 0 执行到 blocked[id] = true 这个语句是发生了进程切换，操作系统调度进程 1 运行。这种情况下会导致进程 0 和进程 1 同时进入临界区。

Consider that P(0) executed to blocked[id] = true, a process switch occurred. OS scheduled P(1) to run. In this case, P(0) and P(1) enter the critical section at the same time.

Consider the case in which turn equals 0 and P(1) sets blocked[1] to true and then finds blocked[0] set to false. P(0) will then set blocked[0] to true, find turn = 0, and enter its critical section. P(1) will then assign 1 to turn and will also enter its critical section. The error was pointed out in [RAYN86]

7. Give a solution to bounded producer-consumer problem with semaphore.

A Solution to the Bounded-Buffer Producer/Consumer Problem Using Semaphores

```
/* program boundedbuffer */
      const int sizeofbuffer = /* buffer size */;
      semaphore s = 1, n = 0, e = sizeofbuffer;
      void producer()
      {
           while (true) {
                produce();
                semWait(e);
                semWait(s);
                append();
                semSignal(s);
                semSignal(n);
           }
      }
      void consumer()
      {
           while (true) {
                semWait(n);
                semWait(s);
                take();
                semSignal(s);
                semSignal(e);
                consume();
           }
      }
      void main()
      {
                 parbegin (producer, consumer);
      }
```

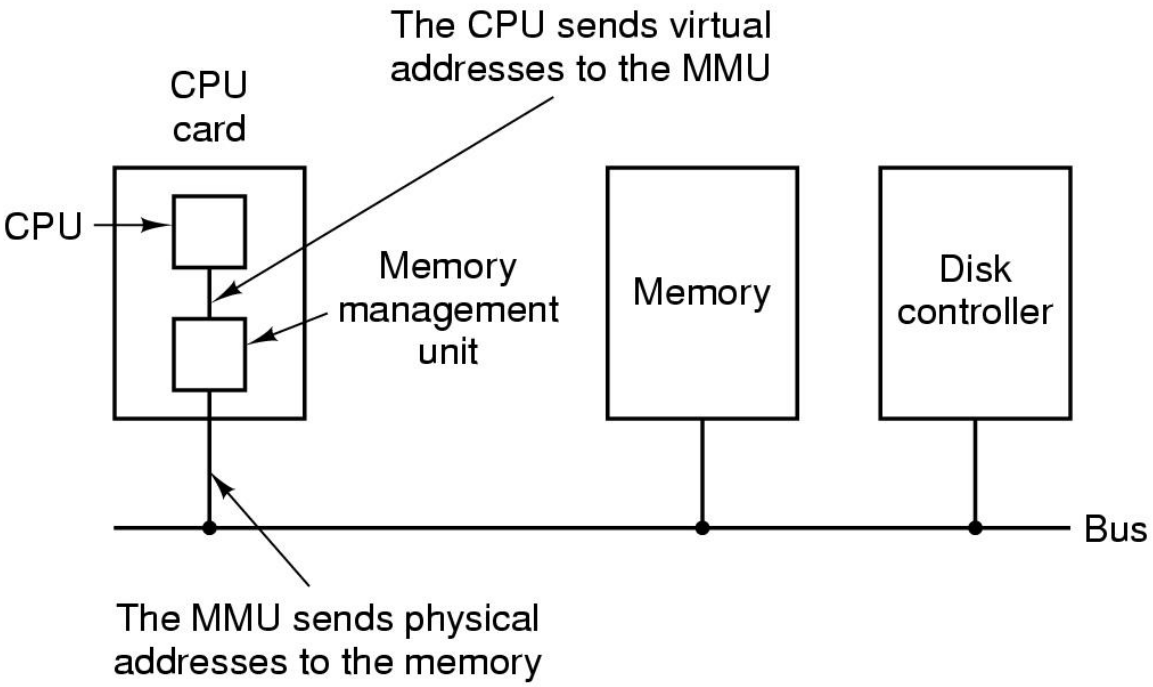
8. What is deadlock?

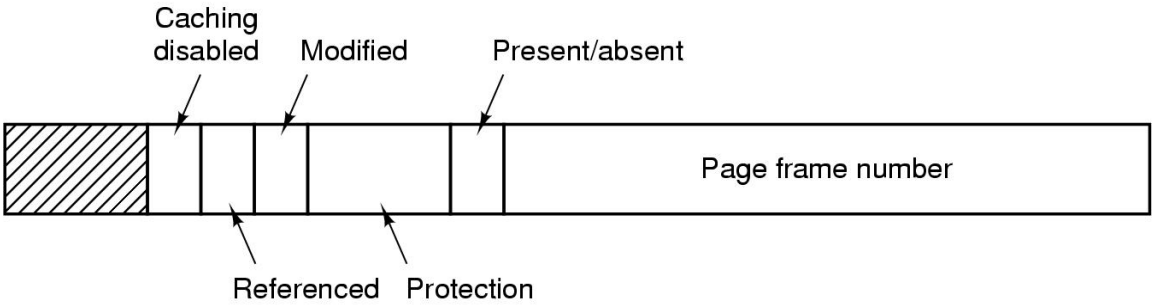
The permanent blocking of a set of processes that either compete for system resources or communicate with each other.
一组竞争系统资源或进行通信的进程间“永久阻塞”
四个条件：*①mutual exclusion（互斥）②hold and wait（占有且等待）③no preemption（非抢占）④circular wait（循环等待）*
四种策略：*①ignorance（忽略）②prevention（预防）③avoidance（避免）④检测与破坏（detection and destroy）*

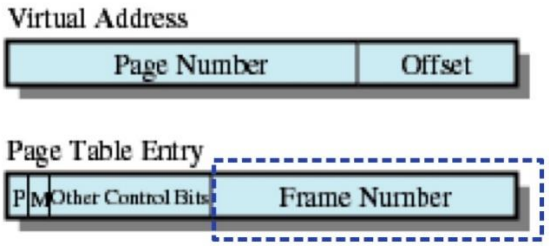
9. Banker algorithms? Safe state/unsafe state

*The strategy of **resource allocation denial,** referred to as the banker's algorithm. Consider a system with a fixed number of processes and a fixed number of resources. At any time a process may have zero or more resources allocated to it. The state of the system reflects the current allocation of resources to processes. Thus, the state consists of the two vectors, Resource and Available, and the two matrices, Claim and Allocation, defined earlier. It ensures that the system of processes and resources is always in a safe state. When a process makes a request for a set of resources, assume that the request is granted, update the system state accordingly, and then determine if the result is a safe state. If so, grant the request and, if not, block the process until it is safe to grant the request.*
***A safe state** is one in which there is at least one sequence of resource allocations to processes that does not result in a deadlock (i.e., all of the processes can be run to completion).*
***An unsafe state** is a state that is not safe (one or more processes cannot be run to completion).*

10. Address mapping in paging system:







例题：一页 4K，页号为 1 的页，其页框号为 6，求 mov AX, [10000]后页实际地址

$Page\ number = \frac{virtual\ address}{page\ size}$=10000/4096=2

$Offset = (virtual\ addnumberress)\ mod\ (page\ size) = 1808$

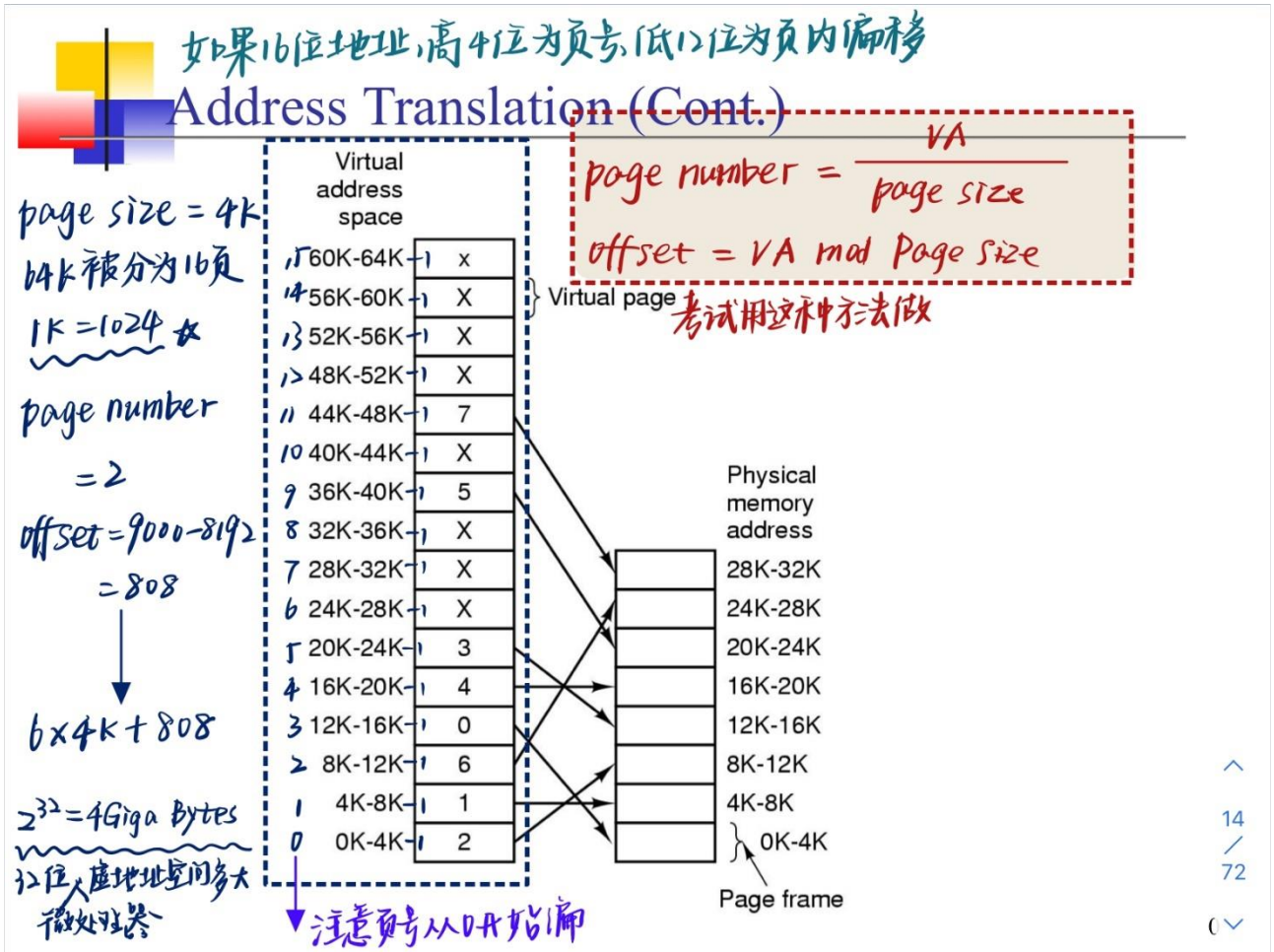
$Physical\ address = (page\ frame\ number) * (page\ size) + offset$

$= 6 * 4K\ + 1808 = 26384$

Page frame number 可由 page number 来查找，题上会给相关信息

15	60k-64k	X
14	56k-60k	X
13	52k-56k	X
12	48k-52k	X
11	44k-48k	7
10	40k-44k	X
9	36k-40k	5
8	32k-36k	X
7	28k-32k	X
6	24k-28k	X
5	20k-24k	3
4	16k-20k	4
3	12k-16k	0
2	8k-12k	6
1	4k-8k	1
0	0-4k	2

7	28k-32k
6	24k-28k
5	20k-24k
4	16k-20k
3	12k-16k
2	8k-12k
1	4k-8k
0	0-4k



8.1. Suppose the page table for the process currently executing on the processor looks like the following. All numbers are decimal, everything is numbered starting from zero, and all addresses are memory byte addresses. The page size is 1,024 bytes.

Virtual page number	Valid bit	Reference bit	Modify bit	Page frame number
0	1	1	0	4
1	1	1	1	7
2	0	0	0	—
3	1	0	0	2
4	0	0	0	—
5	1	0	1	0

a. Describe exactly how, in general, a virtual address generated by the CPU is translated into a physical main memory address.
b. What physical address, if any, would each of the following virtual addresses correspond to? (Do not try to handle any page faults, if any.)
(i) 1,052
(ii) 2,221
(iii) 5,499

*Split binary address into virtual **page number and offset**; use VPN(Virtual page number) as index into page table; extract **page frame number**; **concatenate offset** to get physical memory address*
将二进制地址拆分为虚拟页号和偏移量;使用 VPN 作为索引进入页表;提取页帧数;连接偏移量以获得物理内存地址

(i) 1052 = 1024 + 28 maps to VPN 1 in PFN 7, (7 × 1024+28 = 7196)

(ii) 2221 = 2 × 1024 + 173 maps to VPN 2, **page fault**

(iii) 5499 = 5 × 1024 + 379 maps to VPN 5 in PFN 0, (0 × 1024+379 = 379)

11. What is TLB?

Translation lookaside buffer (TLB)转移后备缓冲器
*In principle, every virtual memory reference can cause **two physical memory accesses**:*
one to fetch the appropriate page table entry
another to fetch the desired data.
*To overcome this problem, most virtual memory schemes make use of **a special high-speed cache** for page table entries.*

12. Page replacement algorithms (Optimal/LRU/FIFO/CLOCK) in paging system.

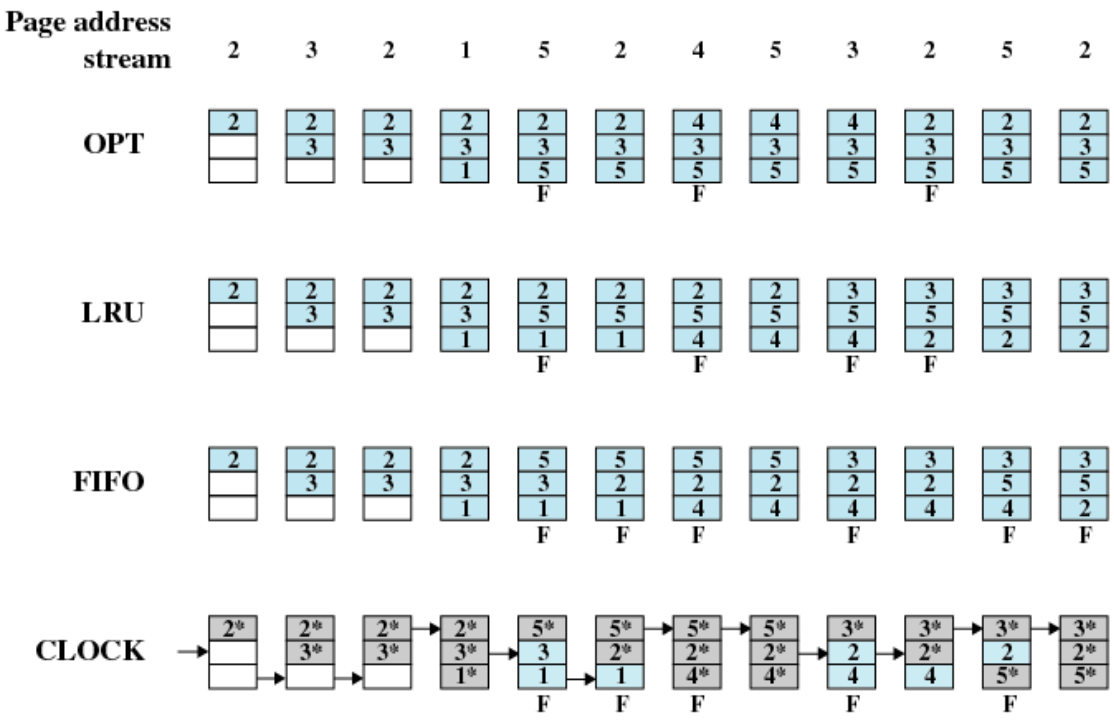


Figure 8.15 Behavior of Four Page-Replacement Algorithms

OPTIMICAL:

- *Replace the page **for which the time to the next reference is the longest***
- *Impossible to have perfect knowledge of future events*

LRU:

- Replaces the page that has not been referenced for the longest time *替换主存中上次使用距当前最远的页*
- **By the principle of locality,** this should be the page least likely to be referenced in the near future 根据局部性原理，这也是最近最不可能访问到的页
- Each page could be tagged with the time of last reference. This would require a great deal of overhead 每一页添加一个最后一次访问的时间标签，需要额外的开销

FIFO:

- *Treats page frames allocated to a process as circular buffer*
- *Pages are removed in round-robin style*
- *Simplest replacement policy to implement*
- ***Page that has been in memory the longest is replaced*** *替换驻留在主存中最长时间的页*

CLOCK:

- Additional bit called a **use bit** *附加位称为使用位*
- When a page is first loaded in memory, the use bit is set to 1 *当某一页首次被装入主存中时，该帧的使用位设置为 1*
- When the page is referenced, the use bit is set to 1 *当该页随后被访问到，它的使用位也被置为 1*
- When it is time to replace a page, **the first frame encountered with the use bit set to 0** is replaced. *当需要替换一页时选择遇到的第一个使用位为 0 的帧替换*
- During the search for replacement, **each use bit set to 1 is changed to 0** 在*查找替换页的时候，所有使用位为 1 的帧被重置为 0*

13. What is preemptive or nonpreemptive scheduling decision mode? How to compute turnaround time in different short-term scheduling algorithms?

Nonpreemptive: *In this case, once a process is in the Running state, it continues to execute until (a) it terminates or (b) it blocks itself to wait for I/O or to request some OS service.*

Preemptive: *The currently running process may be interrupted and moved to the Ready state by the OS. The decision to preempt may be performed when a new process arrives; when an interrupt occurs that places a blocked process in the Ready state; or periodically, based on a clock interrupt.*

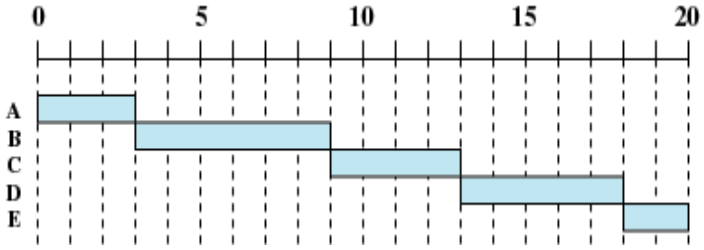
Turnaround Time is the residence time Tr or total time the item spends in the system

$$= \text{waiting time} + \text{services time} = \text{finish time} - \text{arrive time}$$

Table 9.4 Process Scheduling Example

Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

First-Come-First Served (FCFS)



Round Robin

Proc. Number	Creation Time	Required Computing Time
1(A)	0	5
2(B)	1	4
3(C)	2	3
4(D)	4	6
5(E)	5	4

(a) Draw the diagram of scheduling sequence for each process.

根据题意可知采用的是分时调度，时间片为 1 单位时间，应详细给出每个任务被调度的时序图。

提示：要绘制进程的调度时序图，要清楚地知道就绪队列中的进程是如何排队的。要注意到，在 RR 调度过程中只有以下三个时间点就绪队列可能会发生变化:

1）一个新进程到达；

2）一个进程时间片用完；

3）一个进程运行结束；

可以列一个表，给出关键时刻就绪队列的变化，如下：

时刻	事件	调度之前的就绪队列	调度程序选择的进程（处理器上运行）	调度之后的就绪队列	执行时间
0	A 到达	A	A		0
1	B 到达，A 时间片用完	B->A	B	A	A:1(1)
2	C 到达 B 时间片用完	A ->C->B	A	C->B	B:1(1)
3	A 时间片用完	C->B->A	C	B->A	A:1(2)
4	D 到达，C 时间片用完	B->A->D->C	B	A->D->C	C:1(1)
5	E 到达, B 时间片用完	A->D->C->E->B	A	D->C->E->B	B:1(2)
6	A 时间片用完	D->C->E->B->A	D	C->E->B->A	A:1(3)
7	D 时间片用完	C->E->B->A->D	C	E->B->A->D	D:1(1)
8	C 时间片用完	E->B->A ->D->C	E	B->A->D->C	C:1(2)
9	E 时间片用完	B->A ->D->C->E	B	A->D->C->E	E:1(1)
10	B 时间片用完	A->D->C->E->B	A	D->C->E->B	B:1(3)
11	A 时间片用完	D->C->E->B->A	D	C->E->B->A	A:1(4)
12	D 时间片用完	C->E->B->A->D	C	E->B->A->D	D:1(2)
13	C 运行结束	E->B->A->D	E	B->A->D	C:1(3)
14	E 时间片用完	B->A->D->E	B	A->D->E	E:1(2)
15	B 运行结束	A->D->E	A	D->E	B:1(4)
16	A 运行结束	D->E	D	E	A:1(5)
17	D 时间片用完	E->D	E	D	D:1(3)
18	E 时间片用完	D->E	D	E	E:1(3)
19	D 时间片用完	E->D	E	D	D:1(4)
20	E 运行结束	D	D		E:1(4)
21	D 时间片用完	D	D		D:1(5)
22	D 结束	D	D		D:1(6)

根据上表可以给出相应的调度顺序，就很容易画出调度时序图。

(b) Compute turnaround time of each process, and then compute average turnaround time.

先计算出每个进程的周转时间，在此基础上计算平均周转时间。注意，周转时间是一个进程从被创建到运行结束的时间。

A 的周转时间为 16-0=16；　　B 的周转时间为 15-1=14；

C 的周转时间为 13-2=11；　　D 的周转时间为 22-4=18；

E 的周转时间为 20-5=15；

平均周转时间为（16+14+11+18+15）/5 = 74/5=14.8 单位时间

14. Why Disk cache is usually used in operating system?

***Some written data may be accessed next time**, allowing it to be quickly retrieved from the software's disk cache rather than slowly retrieved from disk.*
一些写出的数据也许下次会被访问到，使这些数据能迅速地从软件设置地磁盘高速缓存中取出，而不是缓慢地从磁盘中取出.

15. Disk arm scheduling algorithms

Table 11.2 Comparison of Disk Scheduling Algorithms

(a) FIFO (starting at track 100)		(b) SSTF (starting at track 100)		(c) SCAN (starting at track 100, in the direction of increasing track number)		(d) C-SCAN (starting at track 100, in the direction of increasing track number)	
Next track accessed	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**
55	45	90	10	150	50	150	50
58	3	58	32	160	10	160	10
39	19	55	3	184	24	184	24
18	21	39	16	90	94	18	166
90	72	38	1	58	32	38	20
160	70	18	20	55	3	39	1
150	10	150	132	39	16	55	16
38	112	160	10	38	1	58	3
184	146	184	24	18	20	90	32
Average seek length	55.3	**Average seek length**	27.5	**Average seek length**	27.8	**Average seek length**	35.8

16. What is Absolute path or relative path? Explain how file pathname /usr/ast/mbox is be parsed in Unix SVR4?

2. 绝对路径与相对路径（Absolute Path and Relative Path）

以 C：/User_B/ Word /Unit_A /ABC 为例，假设用户 B 当前工作目录是“Word”，则文件 ABC 的：

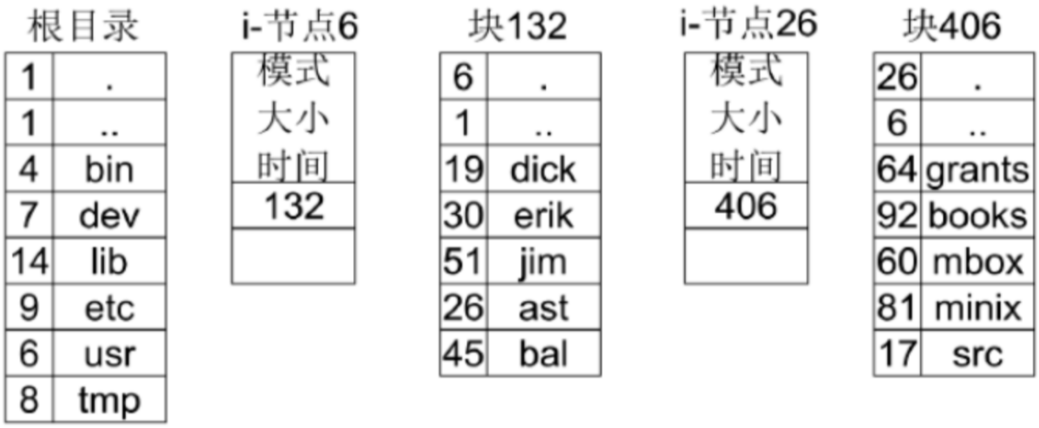
绝对路径：从根目录开始做起点的路径：如 C：/User_B/ Word /Unit_A /ABC

相对路径：从当前目录开始做起点的路径：如 ../.. /Word /Unit_A /ABC

●配置文件时最好用相对路径，则工程从 c 盘移到 D，E,F 盘，仍可编写。

***Absolute path:** the path of a node starting from the root directory*
***Relative path:** the path to a node from the current directory*

Explain how file pathname /usr/ast/mbox is be parsed（解析） in Unix SVR4?



路径解析：

1、从根目录读内容，取出目录项 | 6 | usr |

2、读索引结点 i—node6 的内容，从属性内容中判断出 usr 为目录

3、从索引结点 i—node6 中找到第一个块地址 132

4、读地址为 132 的块信息，取出目录项 | 26 | ast |

5、读索引结点 i—node26 的内容，从属性内容中判断出 ast 为目录

6、从索引结点 i—node26 中找到第一个块地址 406

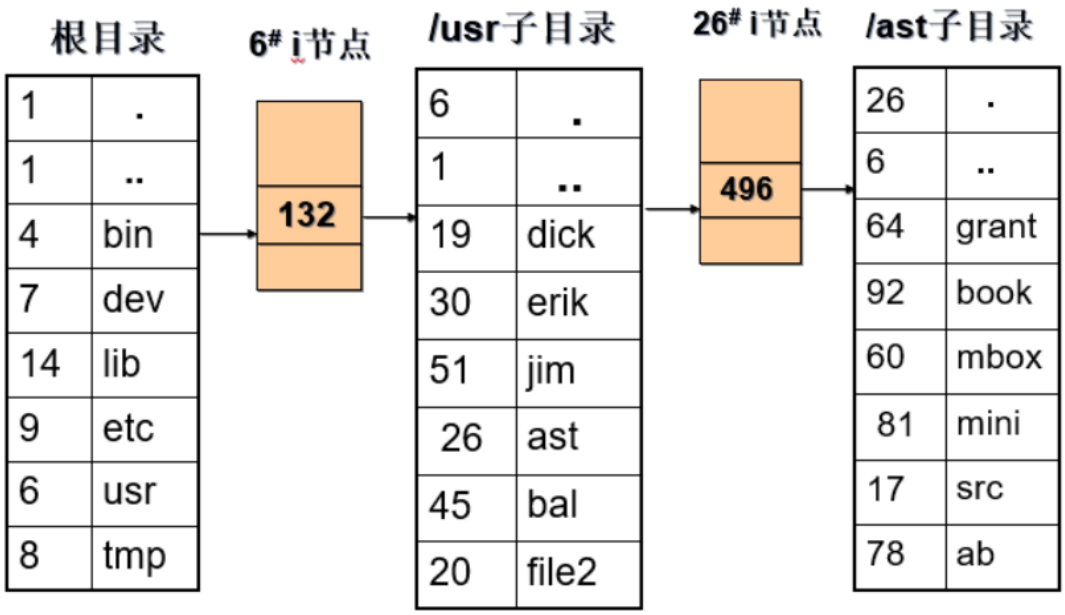
7、读地址为 406 的块信息，取出目录项 | 60 | mbox |

8、读索引结点 i—node406 的内容，从属性内容中判断出 mbox 为文件，结束搜索

例：/usr/ast/mbox

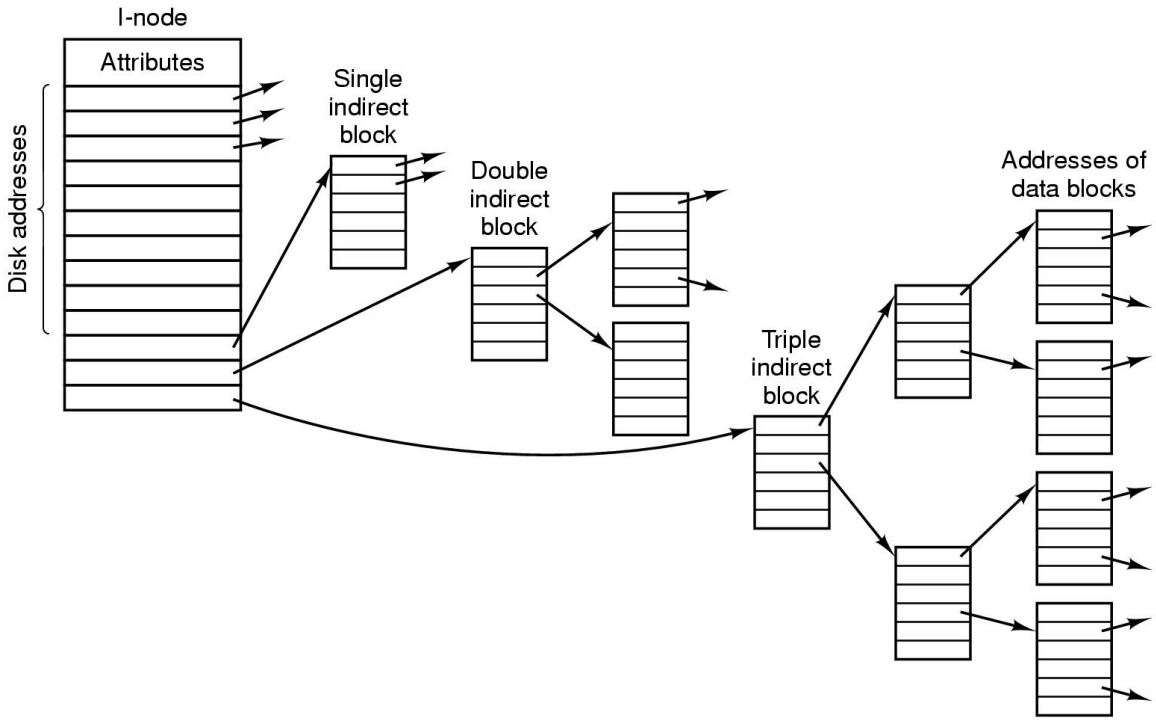
首先在根目录中对 usr 进行文件名的匹配。其对应的是6号i节点（索引节点）。根据6号i节点，我们知道这个子目录存在在132块，也就是文件的物理地址。

然后进入132块，进入 usr 的子目录中查找 ast 这个文件名。同理进入 ast 目录 查找 mbox 文件。通过60号索引节点查找到其对应的物理磁盘块号。



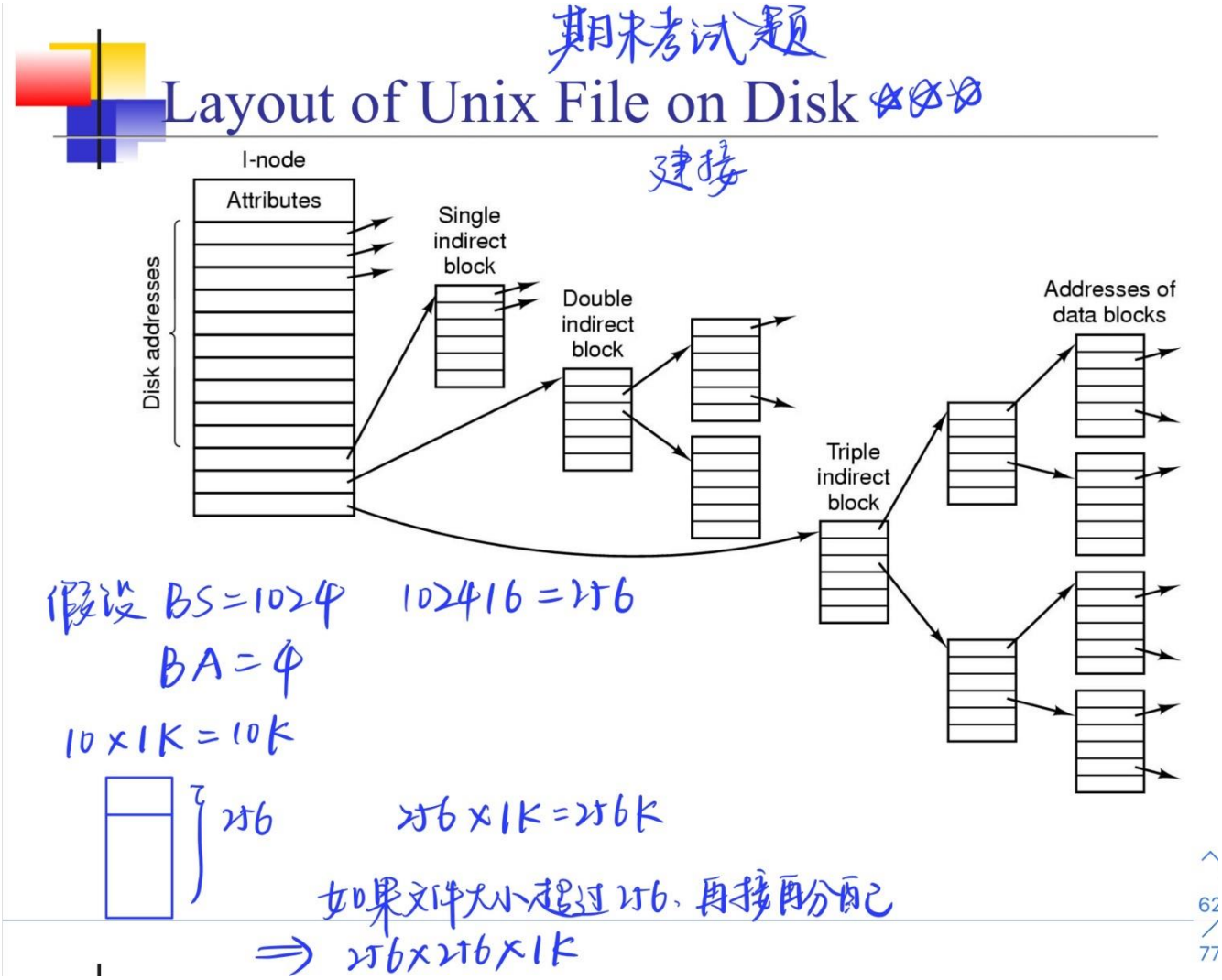
Usr starts with a file name match in the root directory. This ***corresponds to node 6 I (index node)****. From node 6 I, we know that this* ***subdirectory*** *exists in block 132, which is the* ***physical address of the file****.*
Then go to block 132 and look for the ast file name in the usr subdirectory, which corresponds to node 26 I (index node). According to node 26 I, ***the physical address of the file is 496****.* ***Similarly, go to the AST directory to search for the mbox file****. The block ID of the physical disk is obtained from index node 60.*

17.Consider a UNIX-style i-node with 10 direct pointers *直接块地址*, one single-indirect pointer *一级指针*, and one double-indirect pointer. Assume that the disk block size *磁盘块* is 1 Kbytes, and that the size of a disk block address is 4 bytes. How large a file can be indexed using such an i-node?



Disk block size = 1k = 1024 bytes
Disk block address size = 4 bytes
1024/4= 256
10 direct pointers : 10 * 1k = 10k
1 single indirect pointers: 256 * 1k = 256k
1 double indirect pointers: 256 * 256k

Maximum file size: 10k + 256k + 256*256k bytes



Consider the organization of a UNIX file as represented by the inode (Figure 12.15). Assume that there are 12 direct block pointers, and a singly, doubly, and triply indirect pointer in each inode. Further, assume that the system block size and the disk sector size are both 8K. If the disk block pointer is 32 bits, with 8 bits to identify the physical disk and 24 bits to identify the physical block, then a. What is the maximum file size supported by this system? b. What is the maximum file system partition supported by this system? c. Assuming no information other than that the file inode is already in main memory, how many disk accesses are required to access the byte in position 13,423,956

答案：a.找出每一个块中根据指针大小来划分块大小的磁盘块指针的数目：

8K/4 = 2K pointers per block

I-Node 所支持的最大文件大小是：

12	+	2K	+	(2K×2K)	+	(2K×2K×2K)
直接寻址		一级间接寻址		二级间接寻址		三级间接寻址
12	+	2K	+	4M	+	8G blocks

将以上数据乘以块大小（8K），得到：

96KB+16MB+32GB+64TB

这就是该系统支持的最大文件大小。

b.每一个分区中都有 24 位用于识别物理块，由此可知：

2^{24}×8K = 16M×8K = 128GB

c.由问题（a）中所得的信息可知，直接块只覆盖了第一个 96KB 区域，而一级间接块覆盖了接下来的 16MB 区域。被请求文件的位置是 13MB 而其偏移很明显地随机落在了一级间接块中。因此会有 2 次磁盘存储访问。一次是为了一级间接块，一次是为了包含被请求数据的块。