操作系统复习

八道大题

- 虚拟内存调度
  - LRU.FIFO算法等
  - 页面
- 银行家算法
  - 死锁避免
- 声明信号量
  - 用法,实现,死锁与饥饿
  - 注意信号量写法,不可以判断,不可以if,必须wait和...看书上的写法
- 进程同步算法
- gatte图
- 虚拟内存实际内存的转换
  - 快表,页表,内存
- 磁盘调度

第一章复习/概述

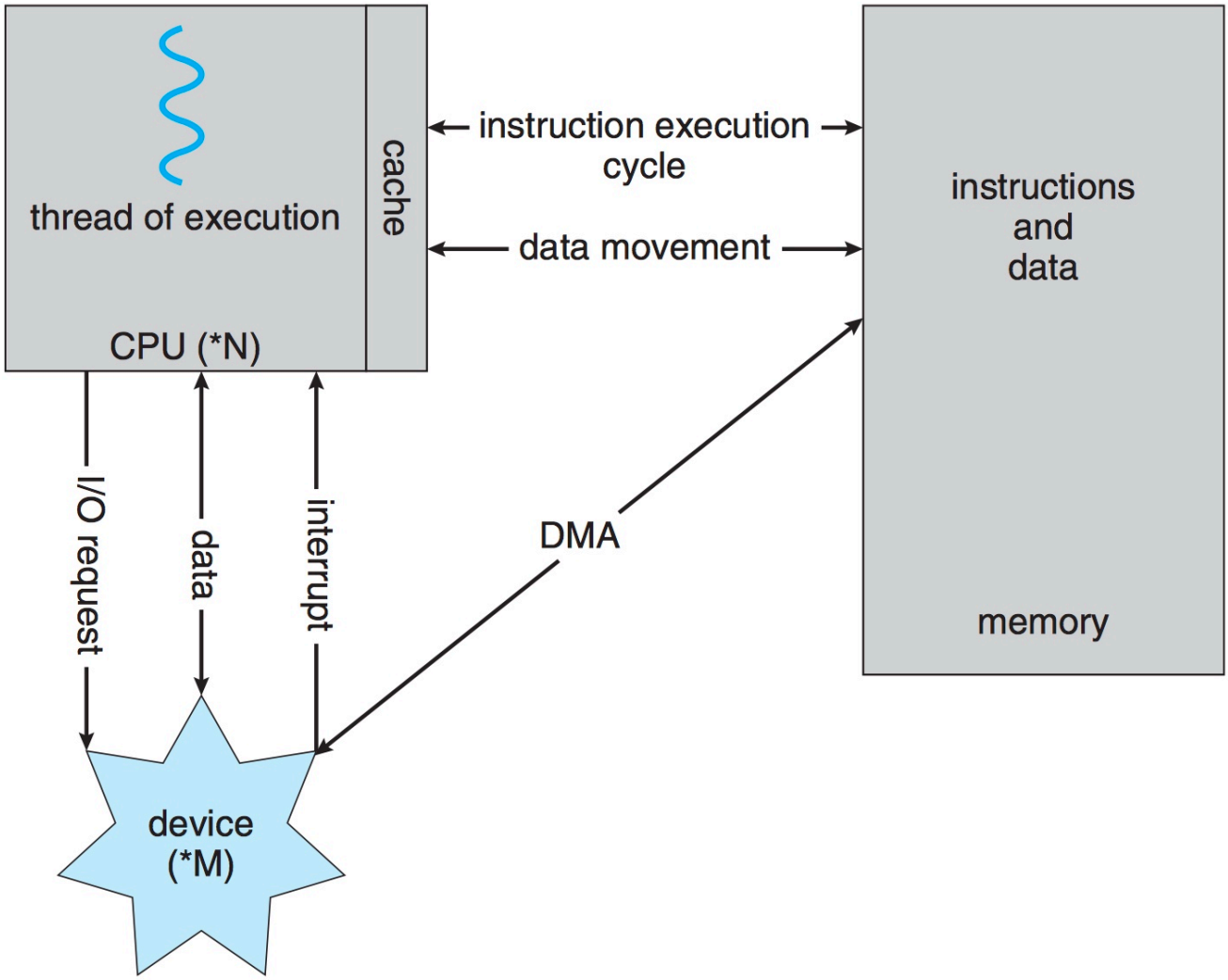


Figure 1.5 How a modern computer system works.

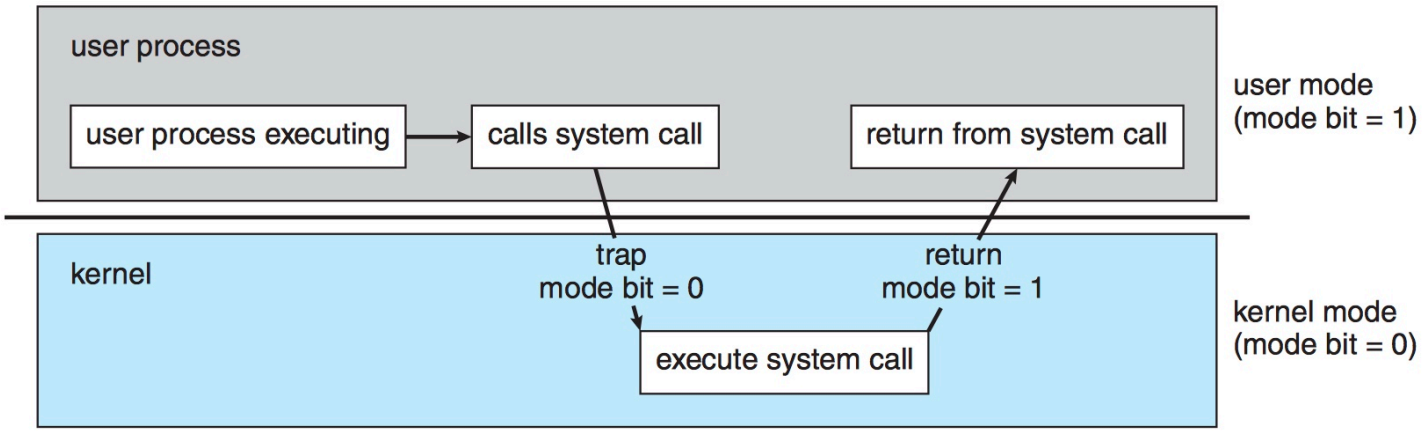


Figure 1.10 Transition from user to kernel mode.

* 图1.用户模式到内核模式的转换
* 现代操作系统是中断驱动的

- 关键习题

1.10 中断（interupt）的目的是什么？陷阱（trap）与中断的区别是什么？陷阱可以被用户程序（user program）有意地的产生吗？如果可以，那目的是什么？

Answer: 中断是一种在系统内硬件产生的流量变化。中断操作装置是用来处理中断请求；然后返回控制中断的上下文和指令。陷阱是软件产生的中断。中断可以被用来标志 I/O 的完成，从而排除设备投票站（device polling）的需要。陷阱可以被用来调用操作系统的程序或者捕捉到算术错误。

第二章复习/操作系统结构

- 重要习题

2.3 讨论向操作系统传递参数的三个主要的方法。

Answer:

1.通过寄存器来传递参数

2.寄存器传递参数块的首地址

3.参数通过程序存放或压进堆栈中，并通过操作系统弹出堆栈。

2.7 命令解释器的用途是什么？为什么它经常与内核是分开的？用户有可能通过使用由操作系统提供的系统调用接口发展一个新的命令解释器？

Answer:命令解释器从用户或文件中读取命令并执行，一般而言把他们转化成系统调用。它通常是不属于内核，因为命令解释会有所变动。用户能够利用由操作系统提供的系统调用接口开发新的命令解释器。这命令解释器允许用户创建、管理进程和确定它们通信的方法（例如通过管道和文件）。所有的功能都被用户程序通过系统调用来使用，这个也可能有用户开发一个新的命令行解释。

第三章/进程

- 进程的概念/执行中的程序,形成所有计算的基础

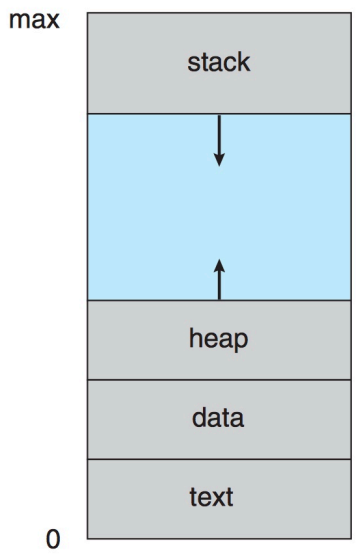


Figure 3.1 Process in memory.

- 进程状态
  - 新的:进程正在被创建
  - 运行: 指令正在被执行
  - 等待:进程等待某个事件的发生
  - 就绪:进程等待分配处理器
  - 终止:进程完成执行

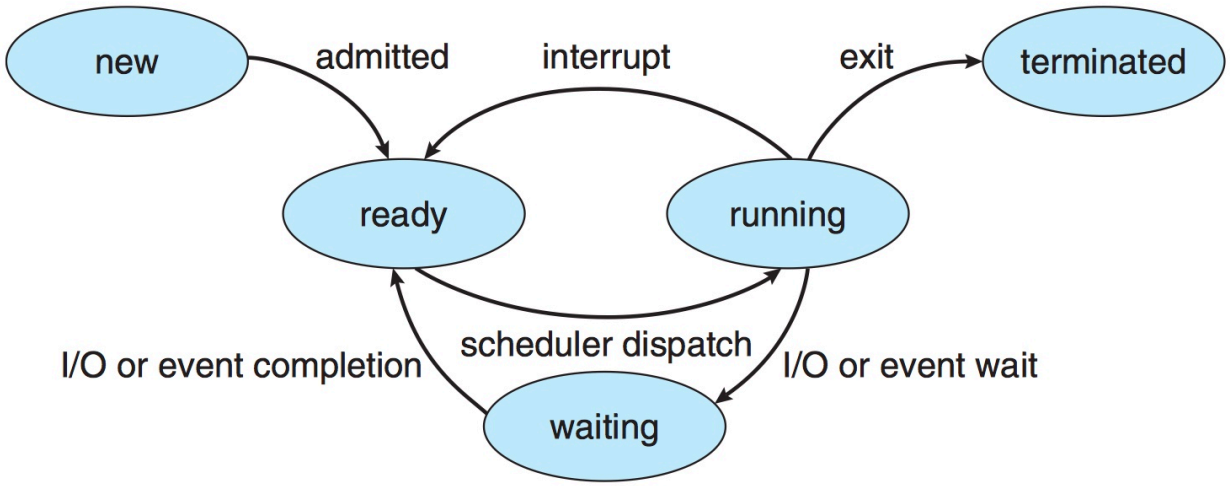


Figure 3.2 Diagram of process state.

- 进程调度
- 重要作业

3.5 问:下面设计的好处和坏处分别是什么?系统层次和用户层次都要考虑到.

A,对称和非对称通信

B,自动和显式缓冲

C,复制发送和引用发送

D,固定大小和可变大小消息

Answer:A.对称和非对称通信：对称通信的影响是它允许发送者和接收者之间有一个集合点。缺点是阻塞发送时，不需要集合点，而消息不能异步传递。因此，消息传递系统，往往提供两种形式的同步。

B.自动和显式缓冲：自动缓冲提供了一个无限长度的队列，从而保证了发送者在复制消息时不会遇到阻塞，如何提供自动缓存的规范，一个方案也许能保存足够大的内存，但许多内存被浪费缓存明确指定缓冲区的大小。在这种状况下，发送者不能在等待可用空间队列中被阻塞。然而，缓冲明确的内存不太可能被浪费。

C.复制发送和引用发送：复制发送不允许接收者改变参数的状态，引用发送是允许的。引用发送允许的优点之一是它允许程序员写一个分布式版本的一个集中的应用程序。Java’s RMI 公司提供两种发送，但引用传递一个参数需要声明这个参数是一个远程对象。

D.固定大小和可变大小消息：涉及的太多是有关缓冲问题，带有定长信息，一个拥有具体规模的缓冲课容纳已知数量的信息缓冲能容纳的可变信息数

第四章/线程

- 线程的概念-一种CPU利用的基本单元,他是形成多线程计算机的基础

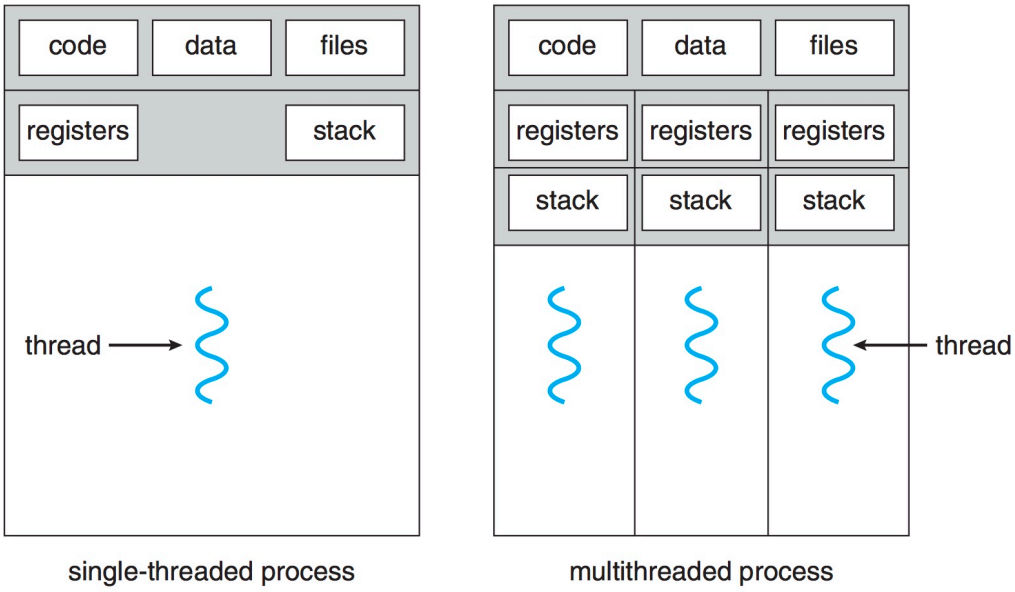


Figure 4.1 Single-threaded and multithreaded processes.

- 优点/相应度高/资源共享/经济/多处理器体系的利用
- 多线程模型/多对一模型/一对一/多对多
    - 内核线程和用户线程

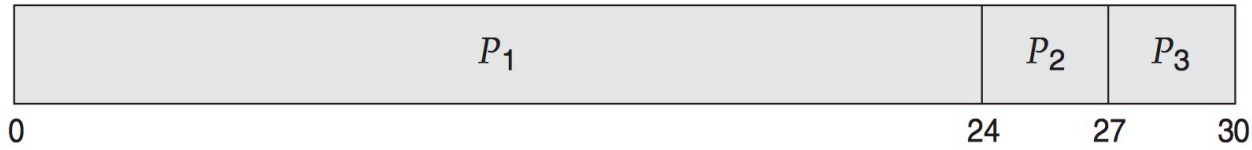
第五章/CPU调度

- CPU调度是多道程序操作系统的基础
- 分派程序/切换上下文/切换用户模式/跳转到用户程序的合适位置
- 调度准则
    - CPU使用率
    - 吞吐量
    - 周转时间
    - 等待时间
    - 响应时间
- **调度算法/gantt图/考点**
    - 先到先服务调度算法FCFS

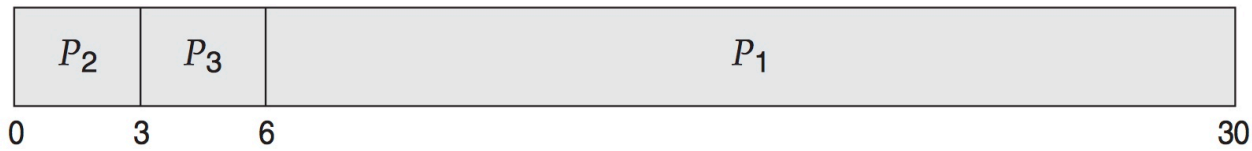
On the negative side, the average waiting time under the FCFS policy is often quite long. Consider the following set of processes that arrive at time 0, with the length of the CPU burst given in milliseconds:

Process	Burst Time
P_1	24
P_2	3
P_3	3

If the processes arrive in the order P_1, P_2, P_3, and are served in FCFS order, we get the result shown in the following **Gantt chart**, which is a bar chart that illustrates a particular schedule, including the start and finish times of each of the participating processes:



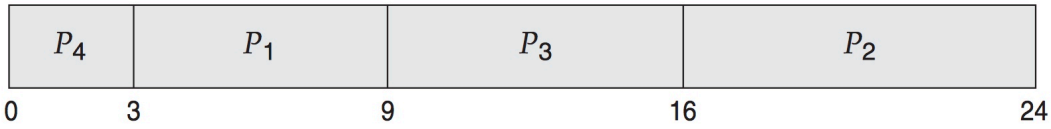
The waiting time is 0 milliseconds for process P_1, 24 milliseconds for process P_2, and 27 milliseconds for process P_3. Thus, the average waiting time is (0 + 24 + 27)/3 = 17 milliseconds. If the processes arrive in the order P_2, P_3, P_1, however, the results will be as shown in the following Gantt chart:



- 最短作业优先调度SJF
  - 平均等待时间最短/分为抢占和非抢占

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

Using SJF scheduling, we would schedule these processes according to the following Gantt chart:



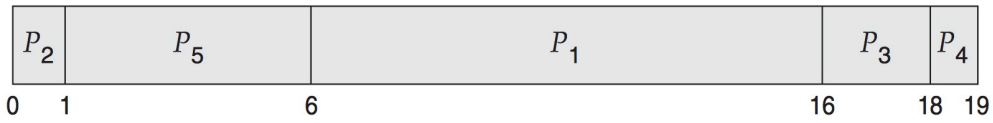
The waiting time is 3 milliseconds for process P_1, 16 milliseconds for process P_2, 9 milliseconds for process P_3, and 0 milliseconds for process P_4. Thus, the average waiting time is $(3 + 16 + 9 + 0)/4 = 7$ milliseconds. By comparison, if we were using the FCFS scheduling scheme, the average waiting time would be 10.25 milliseconds.

- 优先级调度/priority scheduling algorithm/PS
  - 无穷阻塞/饥饿/对于优先级很低的进程

As an example, consider the following set of processes, assumed to have arrived at time 0 in the order $P_1, P_2, \cdots, P_5$, with the length of the CPU burst given in milliseconds:

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

Using priority scheduling, we would schedule these processes according to the following Gantt chart:

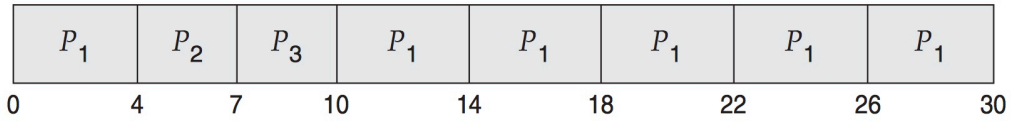


The average waiting time is 8.2 milliseconds.

- 轮转法调度/RR/round-robin
  - 时间片

Process	Burst Time
P_1	24
P_2	3
P_3	3

If we use a time quantum of 4 milliseconds, then process P_1 gets the first 4 milliseconds. Since it requires another 20 milliseconds, it is preempted after the first time quantum, and the CPU is given to the next process in the queue, process P_2. Process P_2 does not need 4 milliseconds, so it quits before its time quantum expires. The CPU is then given to the next process, process P_3. Once each process has received 1 time quantum, the CPU is returned to process P_1 for an additional time quantum. The resulting RR schedule is as follows:



Let's calculate the average waiting time for this schedule. P_1 waits for 6 milliseconds (10 - 4), P_2 waits for 4 milliseconds, and P_3 waits for 7 milliseconds. Thus, the average waiting time is 17/3 = 5.66 milliseconds.

- 多级队列调度算法/multilevel queue scheduling algorithm
  - 根据进程的属性,内存大小,进程优先级,每个队列由自己的调度算法
  - 队列之间调度一般采用优先级抢占调度

- **算法评估/考点**
  - 平均等待时间

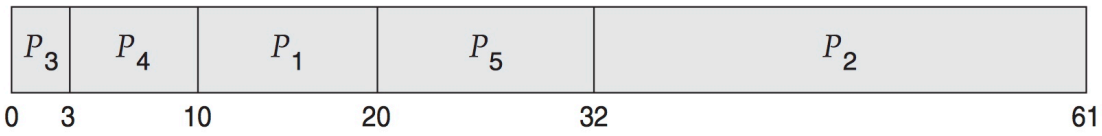
Process	Burst Time
P_1	10
P_2	29
P_3	3
P_4	7
P_5	12

Consider the FCFS, SJF, and RR (quantum = 10 milliseconds) scheduling algorithms for this set of processes. Which algorithm would give the minimum average waiting time?

For the FCFS algorithm, we would execute the processes as

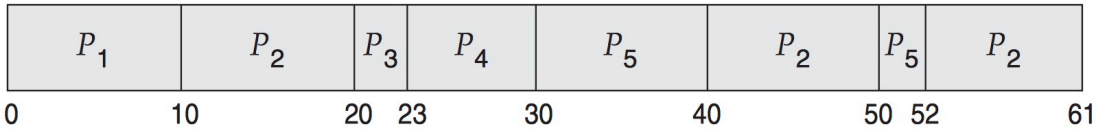
The waiting time is 0 milliseconds for process P_1, 10 milliseconds for process P_2, 39 milliseconds for process P_3, 42 milliseconds for process P_4, and 49 milliseconds for process P_5. Thus, the average waiting time is (0 + 10 + 39 + 42 + 49)/5 = 28 milliseconds.

With nonpreemptive SJF scheduling, we execute the processes as



The waiting time is 10 milliseconds for process P_1, 32 milliseconds for process P_2, 0 milliseconds for process P_3, 3 milliseconds for process P_4, and 20 milliseconds for process P_5. Thus, the average waiting time is (10 + 32 + 0 + 3 + 20)/5 = 13 milliseconds.

With the RR algorithm, we execute the processes as



The waiting time is 0 milliseconds for process P_1, 32 milliseconds for process P_2, 20 milliseconds for process P_3, 23 milliseconds for process P_4, and 40 milliseconds for process P_5. Thus, the average waiting time is (0 + 32 + 20 + 23 + 40)/5 = 23 milliseconds.

- 重点习题

a.画出 4 个 Gantt 图分别演示用 FCFS、SJF、非抢占优先级（数字小代表优先级高）和 RR（时间片=1）算法调度时进程的执行过程。

b.在 a 里每个进程在每种调度算法下的周转时间是多少？

c.在 a 里每个进程在每种调度算法下的等待时间是多少？

d.在 a 里哪一种调度算法的平均等待时间对所有进程而言最小？

答：a.甘特图略

b.周转时间

	FCFS	RR	SJF	非抢占优先级
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.等待时间

	FCFS	RR	SJF	非抢占优先级

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

d.SJF

5.5 下面哪些算法会引起饥饿

a.先来先服务

b.最短工作优先调度

c.轮换法调度

d.优先级调度

答：最短工作优先调度和优先级调度算法会引起饥饿

第六章/进程同步

- 关键点:临界区

```
do {
        entry section
                critical section
        exit section
                remainder section
} while (true);
```

Figure 5.1 General structure of a typical process P_i.

- 信号量

```
do {
   wait(full);
   wait(mutex);
      . . .
   /* remove an item from buffer to next_consumed */
      . . .
   signal(mutex);
   signal(empty);
      . . .
   /* consume the item in next_consumed */
      . . .
} while (true);
```

Figure 5.10 The structure of the consumer process.

- 习题

  6.3 忙等待的含义是什么？在操作系统中还有哪些其他形式的等待？忙等待能完全避免吗？

  给出你的答案。
  答：忙等待意味着一个进程正在等待满足一个没有闲置处理器的严格循环的条件。或者，一个进程通过放弃处理器来等待，在这种情况下的块等待在将来某个适当的时间被唤醒。忙等待能够避免，但是承担这种开销与让一个进程处于沉睡状态，当相应程序的状态达到的时候进程又被唤醒有关。

- **信号量的标准写法?**

第七章 死锁/银行家算法

- 如果所申请的资源被其他等待进程占有,那么等待进程有可能再也无法改变状态,称为**死锁**
- 死锁特征:必要条件:互斥/占有并等待/非抢占/循环等待
- 避免:资源分配图算法/银行家算法
- 不能有环存在
- 银行家

7.5.3.2 Resource-Request Algorithm

Next, we describe the algorithm for determining whether requests can be safely granted.

Let $\boldsymbol{Request}_i$ be the request vector for process P_i. If $\boldsymbol{Request}_i[j] == k$, then process P_i wants k instances of resource type R_j. When a request for resources is made by process P_i, the following actions are taken:

1. If $\boldsymbol{Request}_i \leq \boldsymbol{Need}_i$, go to step 2. Otherwise, raise an error condition, since the process has exceeded its maximum claim.
2. If $\boldsymbol{Request}_i \leq \boldsymbol{Available}$, go to step 3. Otherwise, P_i must wait, since the resources are not available.
3. Have the system pretend to have allocated the requested resources to process P_i by modifying the state as follows:

$$\boldsymbol{Available} = \boldsymbol{Available} - \boldsymbol{Request}_i;$$
$$\boldsymbol{Allocation}_i = \boldsymbol{Allocation}_i + \boldsymbol{Request}_i;$$
$$\boldsymbol{Need}_i = \boldsymbol{Need}_i - \boldsymbol{Request}_i;$$

If the resulting resource-allocation state is safe, the transaction is completed, and process P_i is allocated its resources. However, if the new state is unsafe, then P_i must wait for $\boldsymbol{Request}_i$, and the old resource-allocation state is restored.

- 死锁恢复
  - 进程终止/资源抢占/选择一个牺牲品
- 习题

7.11 Consider the following snapshot of a system:

	Allocation	Max	Available
	A B C D	A B C D	A B C D
P0	0 0 1 2	0 0 1 2	1 5 2 0
P1	1 0 0 0	1 7 5 0	
P2	1 3 5 4	2 3 5 6	
P3	0 6 3 2	0 6 5 2	
P4	0 0 1 4	0 6 5 6	

Answer the following questions using the banker 痴 algorithm:
a. What is the content of the matrix Need?
b. Is the system in a safe state?
c. If a request from process P1 arrives for (0,4,2,0), can the request be granted immediately?

Answer:
a. What is the content of the matrix Need? The values of Need for processes P0 through P4 respectively are (0, 0, 0, 0), (0, 7, 5, 0), (1, 0, 0, 2), (0, 0, 2, 0), and (0, 6, 4, 2).
b. Is the system in a safe state? Yes. With Available being equal to (1, 5, 2, 0), either process P0 or P3 could run. Once process P3 runs, it releases its resources which allow all other existing processes to run.
c. If a request from process P1 arrives for (0,4,2,0), can the request be granted immediately? Yes it can. This results in the value of Available being (1, 1, 0, 0). One ordering of processes that can finish is P0 , P2 , P3 , P1 , and P4 .

第八章 内存管理

- 内存硬件的组织方式/内存管理技术(分页/分段)/
- 一大块可用内存称为孔,从一组可用孔选择一个空闲孔的方法有首次适应,最佳适应,最差适应

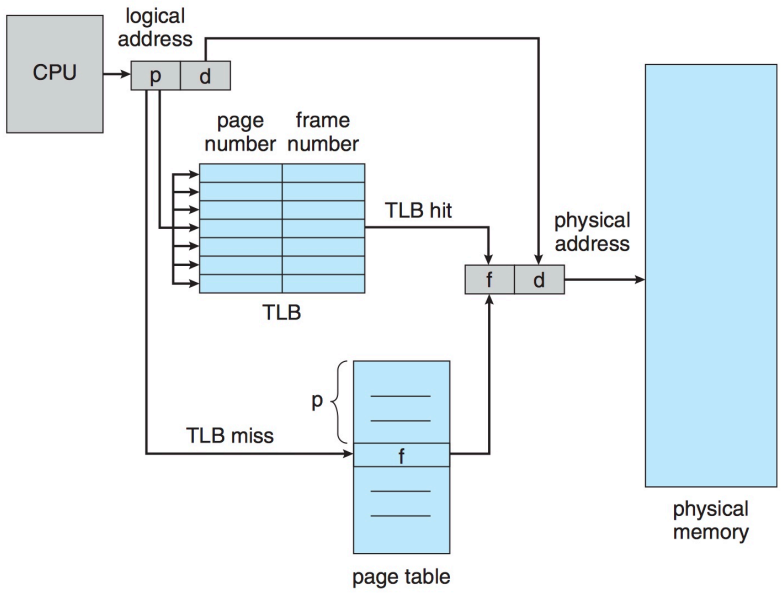


Figure 8.14 Paging hardware with TLB.

- 分页/段「页必须存储页入口和出口,段存储段基址」
- 习题解析

> **8.9** 考虑一个分页系统在内存中存储着一张页表。**a.**如果内存的查询需要 **200** 毫秒，那么一个分页内存的查询需要多长时间？**b.**如果我们加上相关联的寄存器，**75%**的页表查询可以在相关联的寄存器中找到，那么有效的查询时间是多少？（假设如果入口存在的话，在相关的寄存器中找到页表入口不花费时间）
>
> **Answer**：a.400 毫秒：200 毫秒进入页表，200 毫秒进入内存中的字
>
> b.有效进入时间=0.75*200 毫秒+0.25*400 毫秒=250 毫秒

8.12 Consider the following segment table:

Segment	Base	Length
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

What are the physical addresses for the following logical addresses?
a. 0,430 b. 1,10 c. 2,500 d. 3,400 e. 4,112

Answer:
a. 219 + 430 = 649
b. 2300 + 10 = 2310
c. illegal reference, trap to operating system d. 1327 + 400 = 1727
e. illegal reference, trap to operating system

第九章 虚拟内存

- 页面置换
  - 页面置换算法
  - FIFO页置换

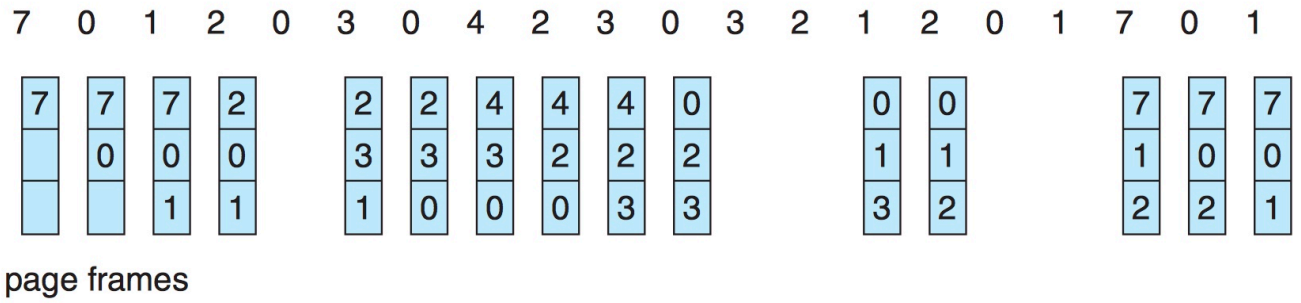


Figure 9.12 FIFO page-replacement algorithm.

  - 最优置换

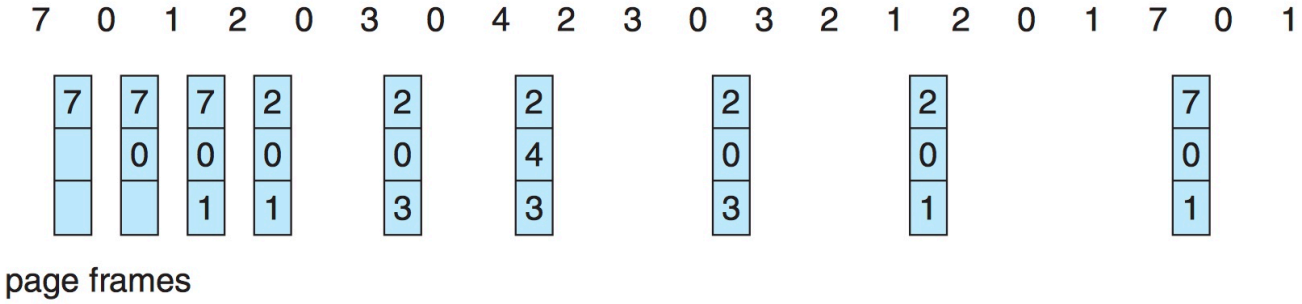


Figure 9.14 Optimal page-replacement algorithm.

  - LRU页置换

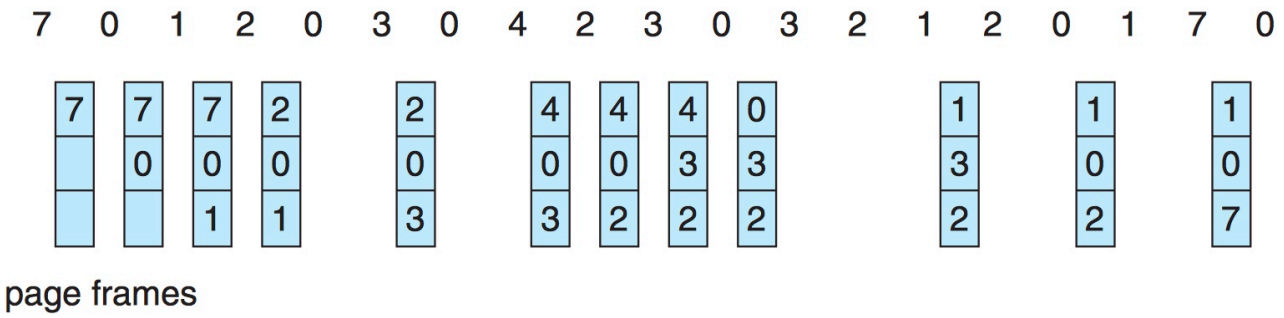


Figure 9.15 LRU page-replacement algorithm.

- 习题答案

9.4 问：某个计算机给它的用户提供了 2^{32} 的虚拟内存空间，计算机有 2^{14}B 的物理内存，虚拟内存使用页面大小为4094B的分页机制实现。一个用户进程产生虚拟地址11123456，现在说明一下系统怎么样建立相应的物理地址，区分一下软件操作和硬件操作。（第六版有翻译）

答：该虚拟地址的二进制形式是 0001 0001 0001 0010 0011 0100 0101 0110。由于页面大小为 212，页表大小为 220，因此，低 12 位的“0100 0101 0110 ”被用来替换页（page），而前 20 位“0001 0001 0001 0010 0011”被用来替换页表(page table)。

9.14 假设一个请求调页系统具有一个平均访问和传输时间为 **20ms** 的分页磁盘。地址转换是通过在主存中的页表来进行的，每次内存访问时间为 **1μs**。这样，每个通过页表进行的内存引用都要访问内存两次。为了提高性能，加入一个相关内存，当页表项在相关内存中时，可以减少内存引用的访问次数。

假设 **80%**的访问发生在相关内存中，而且剩下中的 **10%**（总量的 **2%**）会导致页错误。内存的有效访问时间是多少？

Answer:

有效访问时间= (0.8) × (1 μsec)+ (0.1) × (2 μsec) + (0.1) × (5002 μsec)

= 501.2 μsec

= 0.5 millisec

第十章 文件系统接口

- 文件属性

    - 名称/标识符/类型/位置/大小/保护/时间
- 文件操作
    - 创建文件
    - 读文件
    - 在文件内重定位
    - 删除文件
    - 截断文件
- 操作系统的主要任务是将逻辑文件概念影射到物理存储设备
- 目录结构
    - 单层结构目录
    - 双层结构目录
    - 树状结构目录
    - 无环图目录

第十一章 文件系统实现

- 文件系统持久驻留在外存上.
- 文件在磁盘上有三种不同空间分配方法:
    - 连续的
    - 链接的
    - 索引分配
- 习题

11.3假设有一个系统，它的空闲空间保存在空闲空间链表中：

a. 假设指向空闲空间链表的指针丢失了，系统能不能重建空闲空间链表，为什么？

b. 试想一个文件系统类似 UNIX 的使用与分配索引，有多少磁盘 I / O 操作可能需要阅读的内容，一个小地方的档案在 a/b/c？假设此时没有任何的磁盘块，目前正在缓存。

c.设计一个方案以确定发生内存错误时候总不会丢失链表指针

答：

a.为了重建自由名单，因此有必要进行“垃圾收集” 。这就需要搜索整个目录结构，以确定哪些网页已经分配给了工作。这些剩余的未分配的网页可重新作为自由空间名单。

b.在自由空间列表里指针可存储在磁盘上，但也许在好几个地方。

c.指针可以存储在磁盘上的数据结构里或者在非挥发性RAM （ NVRAM. ）

11.6 设想一个在磁盘上的文件系统的逻辑块和物理块的大小都为512B。假设每个文件的信息已经在内存中，对3种分配方法（连续分配，链接分配和索引分配），分别回答下面的问题：

A，逻辑地址到物理地址的映射在系统中怎么样进行的？（对于索引分配，假设文件总是小于512块长）

B，假设现在处在逻辑块10（最后访问的块是块10），限制想访问块4，那么必须从磁盘上读多少个物理块）

答：设想Z是开始文件的地址（块数），

a.毗连。分裂逻辑地址由512的X和Y所产生的份额和其余的分别。

1：将X加入到Z获得物理块号码。 Y是进入该区块的位移。

2.：1

b.联系。分裂逻辑地址由511的X和Y所产生的份额和其余的分别。

1.：找出联系名单（将X + 1块）。 Y + 1是到最后物理块的位移

2.：4

c.收录。分裂的逻辑地址由512的X和Y所产生的份额和其余的分别。

第十二章 大容量存储器的结构

- 磁盘传输速率
- 磁盘调度
  - FCFS/先来先服务
  - SSTF调度/最短寻道时间优先算法
  - SCAN/电梯
  - C-SCAN
  - LOOK
  - C-LOOK
- 习题

12.2 Suppose that a disk drive has 5000 cylinders, numbered 0 to 4999. The drive is currently serving a request at cylinder 143, and the previous request was at cylinder 125. The queue of pending requests, in FIFO order, is 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130 Starting from the current head position, what is the total distance (in cylinders) that the disk arm moves to satisfy all the pending requests, for each of the following disk-scheduling algorithms?
a. FCFS b. SSTF c. SCAN d. LOOK e. C-SCAN

Answer:
a. The FCFS schedule is 143, 86, 1470, 913, 1774, 948,1509,1022,1750, 130. The total seek distance is 7081.
b. The SSTF schedule is 143,130,86,913,948,1022,1470,1509,1750,1774. The total seek distance is 1745.
c. The SCAN schedule is 143, 913,948,1022,1470,1509,1750,1774,4999,130,86. The total seek distance is 9769.
d. The LOOK schedule is 143,913,948,1022,1470,1509,1750,1774,130,86. The total seek distance is 3319.
e. The C-SCAN schedule is 143,913,948,1022,1470,1509,1750,1774,4999, 86, 130. The total seek distance is 9813.
f. (Bonus.) The C-LOOK schedule is 143,913,948,1022,1470,1509,1750, 1774, 86, 130. The total seek distance is 3363.

第十三章 I/O输入系统

- 习题

13.3 Consider the following I/O scenarios on a single-user PC.
a. A mouse used with a graphical user interface
b. A tape drive on a multitasking operating system (assume no device preallocation is available)
c. A disk drive containing user files
d. A graphics card with direct bus connection, accessible through memory-mapped I/O For each of these I/O scenarios, would you design the operating system to use buffering, spooling, caching, or a combination? Would you use polled I/O, or interrupt-driven I/O? Give reasons for your choices.

Answer:
a. A mouse used with a graphical user interface Buffering may be needed to record mouse movement during
times when higher-priority operations are taking place. Spooling and caching are inappropriate. Interrupt driven I/O is most appropriate.
b. A tape drive on a multitasking operating system (assume no device preallocation is available) Buffering may be needed to manage throughput difference between the tape drive and the source or destination of the I/O, Caching can be used to hold copies of data that resides on the tape, for faster

access. Spooling could be used to stage data to the device when multiple users desire to read from or write to it. Interrupt driven I/O is likely to allow the best performance.
c. A disk drive containing user files Buffering can be used to hold data while in transit from user space to the disk, and visa versa. Caching can be used to hold disk-resident data for improved performance. Spooling is not necessary because disks are shared-access devices. Interrupt-driven I/O is best for devices such as disks that transfer data at slow rates.
d. A graphics card with direct bus connection, accessible through memory-mapped I/O Buffering may be needed to control multiple access and for performance (double-buffering can be used to hold the next screen image while displaying the current one). Caching and spooling are not necessary due to the fast and shared-access natures of the device. Polling and interrupts are only useful for input and for I/O completion detection, neither of which is needed for a memory-mapped device.

13.4 In most multiprogrammed systems, user programs access memory through virtual addresses, while the operating system uses raw physical addresses to access memory. What are the implications of this design on the initiation of I/O operations by the user program and their execution by the operating system?

Answer: The user program typically specifies a buffer for data to be transmitted to or from a device. This buffer exists in user space and is specified by a virtual address. The kernel needs to issue the I/O operation and needs to copy data between the user buffer and its own kernel buffer before or after the I/O operation. In order to access the user buffer, the kernel needs to translate the virtual address provided by the user program to the corresponding physical address within the context of the user program 痴 virtual address space. This translation is typically performed in software and therefore incurs overhead. Also, if the user
buffer is not currently present in physical memory, the corresponding page(s) need to obtained from the swap space. This operation might require careful handling and might delay the data copy operation.