操作系统大题 Operating System

一、内存管理例题 1：

·有一页式系统,其页表存放在内存中.

1） 如果对内存的一次存取需要 1.5us,问实现一次页面访问的存取时间是多少?

2） 如果系统增加有快表,平均命中率为 85%,当页表项在快表中时,其查找时间忽略不计,问此时的存取时间为多少?

解：

（1） 页表中访问内存-------先访问页表，

再访问内存地址-------一共访问 2 次

所以访问存取时间为

答案：$2\times1.5=3$ us

（2） 当页表项在快表中时,查找时间忽略为 0

——无法确定是否在快表中可以找到

增加快表之后，需要考虑 2 种情况

1.访问时，可以直接在快表中访问到的

2.访问时，在快表中找不到，需要在内存中访问，依次访问页表，和物理块-----一共访问 2 次

所以综上所述：两种情况综合考虑

快表中命中率 85%

那剩余不能在快表中找到：15%

答案：$85\%\times1.5+15\%\times1.5\times2=1.725$ us

二、内存管理例题 2：

·在页式、段式和段页式存储管理中，当访问一条指令或数据时，各需要访问内存几次？其过程如何？

·假设一个页式存储系统有快表，多数活动页表项都可存在其中，如果页表存放在内存中，内存访问时间是 1us，检索快表的时间为 0.2us，若快表的命中率是 85%，则有效存取时间是多少？

·若快表的命中率为 50%,则有效存取时间是多少？

解：

（1）页式存储管理中，访问指令或数据时，首先要访问内存中的页表，查找到指令或数据所在页面对应的页表项，然后再根据页表项查找访问指令或数据所在的内存页面。需要访问内存两次。

段式存储管理同理，需要访问内存两次。

段页式存储管理，首先要访问内存中的段表，然后再访问内存中的页表，最后访问指令或数据所在的内存页面。需要访问内存三次。

对于比较复杂的情况，如多级页表，若页表划分为 N 级，则需要访问内存 N+1 次。若系统中有快表，则在快表命中时，只需要一次访问内存即可。

（2）

按（1）中的访问过程分析，有效存取时间为：

$(0.2+1)\times 85\%+ (0.2+1+1)\times(1-85\%)=1.35(\mu s)$

（3）同理可计算得：

$(0.2+1)\times 50\% +(0.2+1+1)\times(1-50\%)=1.7\ (us)$

从结果可以看出，快表的命中率对访存时间影响非常大。当命中率从 85%降

低到 50%时，有效存取时间增加一倍。因此在页式存储系统中，应尽可能地提高快表的命中率，从而提高系统效率。

注意：在有快表的分页存储系统中，计算有效存取时间时，需注意访问快表与访问内存的时间关系。通常的系统中，先访问快表，未命中时再访问内存；在有些系统中，快表与内存的访问同时进行，当快表命中时就停止对内存的访问。这里题中未具体指明，我们按照前者进行计算。但如果题中有具体的说明，计算时则应注意区别。

三、内存管理例题 3：

在一个页式存储管理系统中，地址空间分页（每页 1kb），物理空间分块，设主存总容量是 256kb，描述主存分配情况的位示图如下图所示，0 表示未分配，1 表示已分配，此时作业调度程序选中一个长为 5.2kb 的作业投入内存，试问：

1）为该作业分配内存后，分配内存时，首先分配低地址的内存空间，请填写该作业的页表内容

2）页式存储管理有无零头存在，若有，会存在什么零头？为该作业分配内存后，会产生零头吗？如果会，大小为多少？

3）假设一个 64MB 内存容量的计算机，其操作系统采用页式存储管理（页面大小 4KB）,内存分配采用位示图方式管理，请问位示图将占用多大的内存？

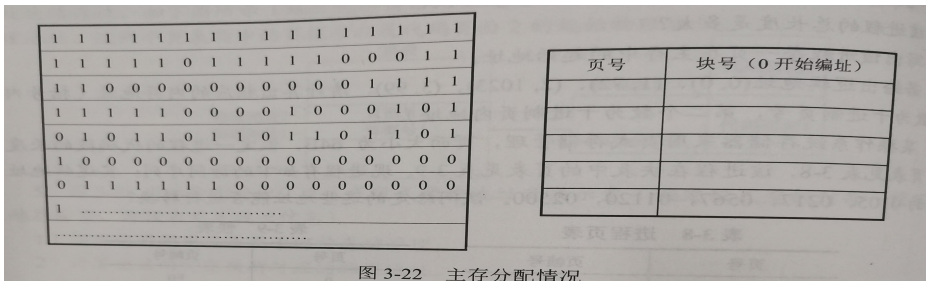

图 3-22 主存分配情况

解：

（1）页表内容如下：

页号	块号
0	21
1	27
2	28
3	29
4	34
5	35

（2）页式存储管理中有零头的存在，会存在内零头，为该作业分配内存后，会产生零头，因为此作业大小为 5.2K，占 6 页，前 5 页满，最后一页只占了 0.2K 的空间，则零头大小为 1K−0.2K=0.8K。

（3）2KB；64M 内存，一页大小为 4K，则共可分成 64K×1K/4K=16K 个物理盘块，在位示图中每一个盘块占 1 位，则共占 16Kb 空间，因为 1B=8b，所以此位示图共占 2KB 空间的内存。

四、存储器分配例题：

给定存储器的划分，依次为 100KB、450KB、250KB、300KB、和 600KB，现有 4 个进程分别依次为：212KB、417KB、112KB、426KB.为了在给定的存储空间中安置进程，现有三种算法：首次适应算法、最佳适应算法和下次适应算法。在这三种算法中，哪一种算法更能充分利用存储空间。

解：

首次适应算法：无法满足，212K 进程放入 450K 的空闲区中，还剩余 238K；417K 的放入 600k 的空闲区中；112K 放入 238 中，426k 的进程无法满足

最佳适应算法：可以满足，212K 放入 250K 空闲区中；417K 放入 450K 空闲区中，112K 放入 300K 空闲区中，426K 放入 600K 空闲区中。

下次适应算法：无法满足，212 放入 600K 空闲区中，剩余 388K；417K 放入 450K 空闲区中，最后导致 426K 的进程请求无法满足。所以，最佳适应算法利用内存最充分。

·补充：

·首次适应算法（First Fit）：该算法从空闲分区链首开始查找，直至找到一个能满足其大小要求的空闲分区为止。然后再按照作业的大小，从该分区中划出一块内存分配给请求者，余下的空闲分区仍留在空闲分区链中。

特点：该算法倾向于使用内存中低地址部分的空闲区，在高地址部分的空闲区很少被利用，从而保留了高地址部分的大空闲区。显然为以后到达的大作业分配大的内存空间创造了条件。

缺点：低地址部分不断被划分，留下许多难以利用、很小的空闲区，而每次查找又都从低地址部分开始，会增加查找的开销。

·最佳适应算法（Best Fit）：该算法总是把既能满足要求，又是最小的空闲分区分配给作业。为了加速查找，该算法要求将所有的空闲区按其大小排序后，以递增顺序形成一个空白链。这样每次找到的第一个满足要求的空闲区，必然是最优的。孤立地看，该算法似乎是最优的，但事实上并不一定。因为每次分配后剩余的空间一定是最小的，在存储器中将留下许多难以利用的小空闲区。同时每次分配后必须重新排序，这也带来了一定的开销。

特点：每次分配给文件的都是最合适该文件大小的分区。

缺点：内存中留下许多难以利用的小的空闲区。

·最坏适应算法（Worst Fit）:最坏适应算法是将输入的作业放置到主存中与它所需大小差距最大的空闲区中。空闲区大小由大到小排序。

特点：尽可能地利用存储器中大的空闲区。

缺点：绝大多数时候都会造成资源的严重浪费甚至是完全无法实现分配。

关于三种放置策略的讨论：首次适应算法、最佳适应算法、最坏适应算法的队列结构。

五、PV 操作例题

桌子上有一只盘子，最多可容纳一个水果，每次只能放入或取出一个水果。爸爸专向盘子中放苹果(apple)，妈妈专向盘子中放桔子(orange)，两个儿子专等吃盘子中的桔子， 两个女儿专等吃盘子中的苹果。请用 PV 操作来实现爸爸、妈妈、儿子、女儿之间的同步与互斥关系。

解：

由题可知，

盘子为互斥资源，因为可以放一个水果，所以 empty 初值为 1；（若容纳两个水果，empty=2）

信号量 mutex 控制对盘子的互斥访问，初值为 1；

apple 和 orange 分别表示盘中苹果和橘子的个数，初值为 0

代码：

```
semaphore empty=1,mutex=1,apple=0,orange=0;
void father(){
     do{
          P(empty);     //等待盘子为空
          P(metux);     //等待获取对盘子的操作
           爸爸向盘中放一个苹果;
          V(mutex);    //释放对盘子的操作
          V(apple);    //通知女儿可以来盘子中取苹果
    }while(TRUE);
}

void mather(){
     do{
          P(empty);        //等待盘子为空
          P(metux);        //等待获取对盘子的操作
           妈妈向盘中放一个桔子;
          V(mutex);        //释放对盘子的操作
          V(orange);       //通知儿子可以来盘子中取橘子
    }while(TRUE);
}
```

```
void son(){
     do{
          P(orange);       //判断盘子中是否有桔子
          P(metux);        //等待获取对盘子的操作
           儿子取出盘中的桔子;
          V(mutex);      //释放对盘子的操作
          V(empty);      //盘子空了，可以继续放水果了
    }while(TRUE);
}
void daugther(){
     do{
          P(apple);       //判断盘子中是否有苹果
          P(metux);        //等待获取对盘子的操作
           女儿取出盘中的苹果;
          V(mutex);      //释放对盘子的操作
          V(empty);      //盘子空了，可以继续放水果了
    }while(TRUE);
}
void main() {              //四个并发进程的同步执行
    cobegin
       father(); mather(); son(); daugther();
    coend
}
```

六、银行家算法（B 站自己看）

七、缺页（如上）

八、进程管理（如上）