2018 年春季学期《操作系统原理与设计》期末试题

一、填空题 12 * 1% = 12%

(1)Fork 3 次创建 **7（如果不算父进程）**个进程?

(2)操作系统是与硬件紧密结合的程序，从这个角度来看，它是一个**资源分配器**和一个**控制程序**。
(3)系统服务的软件接口是**系统调用**。

(4)进程通信的两种基本模型:**共享内存** 和 **消息传递**。

(5)***写时复制(copy-on-write)***技术允许子进程和父进程开始共享同一段内存，只有在修改时子进程才会进行数据复制。

(6)如果满足**一次只有一个进程可以操作变量条件（互斥）**就可以消除竞争条件(race condition)。

(7)死锁的四个条件:**互斥，占有并等待，非抢占，循环等待**。

二、概念解释题 6 * 2% = 12%

(1)多道程序与多任务

多道程序是通过安排作业（编码和数据），使得CPU总有一个执行作业，从而提高CPU利用率。其方法是将多个作业保存在作业池上，作业选择和作业调度同时进 行，当某个作业需要等待时，CPU就切换到另一个作业。

多任务(分时系统)是多道程序的自然延伸，对于多任务系统，CPU还是通过切换作业来执行多个程序，但是由于切换频率很高，用户可以在程序运行时与其交互，并且应当保证响应时间较短。

(2)线程管理方式/模型

有两种方式：用户线程，位于内核之上的用户层，无需内核支持。内核线程，由操作系统直接管理。用户线程和内核线程的典型关系分为：

多对一模型——多个用户级线程映射到一个内核线程，线程管理由用户空间的线程库完成，效率更高，但任意时刻只有一个线程可以访问内核。

一对一模型——每个用户线程对应一个内核线程，一个线程阻塞时不影响其他线程，并发性好，但是内核线程多，开销较大。

多对多模型——以上两者的折衷，多路复用多个用户线程到同样数量或者更少的内核线程。

(3)段错误

段错误是当非法访问一段内存区域时（例如越界访问），操作系统产生相应的异常保护，返回的错误名称就是segmentation fault。

(4)缺页异常

Page fault是一种当MMU发现所要访问的virtual page不可用(或者未被分配实际的物理空间)时CPU产生的中断，它将调用 Page fault服务子程序来为触发它的请求分配一段实际的物理空间，并更新page table。

(5)Belady 异常

一般来说，分配帧数量（缓存）越大，命中率越高，缺页率越低。但对于有些页面置换算法，例如FIFO算法，有时却可能相反。以FIFO为例，这种异常的原因是在同一时刻，较大页框和较小页框保存的页面，二者不是超集子集关系。

三、简答题 6 * 4% = 24%

(1)操作系统提供哪些服务?

操作系统提供的服务根据计算机组成/体系结构或者是设计需求的不同而可能有所不同。根据书中所述，操作系统提供的常见服务有: 提供用户界面、程序执行（加载程序到内存并运行）、I/O操作、文件系统操作、进程间通信、资源分配（如CPU调度和内存管理）、记账、错误检测、保护和安全。

(2)解释系统调用、API 与操作系统的关系?

系统调用提供操作系统服务接口，操作系统在进行内核操作时需要进行系统调用，通常在操作系统中，每秒有成千上万的系统调用执行。

API是提供给程序员的一组函数，相对系统调用来说，更加简洁易用，一个API可能封装了多个系统调用函数。通过API，操作系统接口的大多数细节可以被隐藏起来，且可由运行时库来管理。

(3)临界区问题解决需要满足的三个条件?

互斥——如果某个进程正在临界区执行，那么其他进程都不能在其临界区执行。

进步——通过没有进程在临界区，且有进程需要进入临界区，那么可以进行选择哪个进程可以进入临界区，且这种选择不能无限推迟。

有限等待——从一个进程请求进入临界区到请求被允许，时间是有限的，也就是说其他进程允许进入临界区的次数有上限。

(4)文件系统采取连续分配空间有哪些坏处?

对于文件的创建，会有外部碎片的问题导致空间浪费，也就是即使有足够的空闲空间，但因为是连续分配方式，由于这些空闲空间并不连续，导致无法写入新文件。尽管可以通过移动原有文件，合并空闲空间解决，但是这样做的代价是非常大的。

另一个问题是文件分配空间大小的拓展，对于有些文件，可能在一开始无法确定它需要多少空间，可能需要先创建再后续拓展，但在连续分配的方法下，如果这个文件的两端空间都已被使用，此时就无法再拓展空间了。

(5)FAT32 与 EXT2/3 目录文件有何异同?

两个文件系统的整体结构都大致遵循保留区，FAT/INODE表区，ROOT Directory，data区这样的基本结构。

对于FAT32的目录文件，通常来说（非LFN），它是一个包含了文件名，文件属性，创建时间，簇地址，文件大小等等的数据结构。

对于EXT2/3, directory entry存储的是文件名，文件类型，inode号，记录长度，到下一个directory entry的偏移。

store the the inode number, file name, file type, record length, offset to the next directory entry.

(6)硬链接与软链接有何区别?

A hard link is a directory entry pointing to an existing file. Conceptually speaking, this creates a file with two pathnames. They point to the same inode (no new file content) .

In contrast, a symbolic link is a file which stores the pathname(shortcut). Unlike the hard link, a new inode is created for each symbolic link.

四、解答题 52%

(1)写出 fork 之后 X 行和 Y 行输出结果 5%

```c
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

#define SIZE 5

int nums[SIZE] = {0,1,2,3,4};

int main()
{
int i;
pid_t pid;

   pid = fork();

   if (pid == 0) {
      for (i = 0; i < SIZE; i++) {
         nums[i] *= -i;
         printf("CHILD: %d ",nums[i]); /* LINE X */
      }
   }
   else if (pid > 0) {
      wait(NULL);
      for (i = 0; i < SIZE; i++)
         printf("PARENT: %d ",nums[i]); /* LINE Y */
   }

   return 0;
}
```

LINE X——CHILD：0 CHILD：-1 CHILD：-4 CHILD：-9 CHILD：-16

LINE Y——PARENT：0 PARENT：1 PARENT：2 PARENT：3 PARENT：4

(2)根据先到先服务、最近优先、SCAN 算法计算 9%

Suppose that a disk drive has 5,000 cylinders, numbered 0 to 4999. The drive is currently serving a request at cylinder 2150, and the previous request was at cylinder 1805. The queue of pending requests, in FIFO order, is: 2069, 1212, 2296, 2800, 544, 1618, 356, 1523, 4965, and 3681. 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

2150, 2069, 1212, 2296, 2800, 544, 1618, 356, 1523, 4965, 3681.The total seek distance is 13,011.

b. SSTF

2150, 2069, 2296, 2800, 3681, 4965, 1618, 1523, 1212, 544, 356.The total seek distance is 7586.

c. SCAN

2150, 2296, 2800, 3681, 4965, 4999, 2069, 1618, 1523, 1212, 544, 356. The total seek distance is 7492.

(3)阐述如何从根目录访问\os\files 目录，假设所有 iNode 均未被缓存。

Accessing /os/files involves 3 separate disk operations:

首先reading in the disk block containing the root directory /

然后从根目录"/"找到os的inode号，从inode号找到os文件的block位置后，reading in the disk block containing the directories os

类似的，在os子目录中，找到files的inode号，reading in the disk block containing the directories files，即可访问到files的内容。

(4)计算使读取平均时间达到 200ns 的最大缺页率

Assume we have a demand-paged memory. It takes 8 milliseconds to service a page fault if an empty page is available or the replaced page is not modified, and 20 milliseconds if the replaced page is modified. Memory access time is 100 nanoseconds. Assume that the page to be replaced is modified 70 percent of the time. What is the maximum acceptable page-fault rate for an effective access time of no more than 200 nanoseconds? 在此基础上，增加一个条件:TLB 命中率 50%

设 page-fault rate 为 p , Memory access 100ns相对于page fault的处理时间可以忽略不计。

200ns = (1-p)*100ns+(30% * 8ms+ 70% * 20ms)p

=> p=6x10^(-6)

TLB 命中率 50%，意味着50%的可能只需要一次Memory access，50%需要两次Memory access，命中时不会出现page fault。

200ns = 50*100ns + 50% * (100ns + (1-p)*100ns+(30% * 8ms+ 70% * 20ms)p)

=> p=6x10^(-6)

(5)设计写者优先的读者写者问题(第二类读者写者问题)的解决方案

– **Rule 1.** 多个读者可以同时进行读。

– **Rule 2.** 当有一个写者正在写时，不允许其他写者和读者访问。

– **Rule 3.** 如果有一个写者正在等待，则不允许读者再进入，直到写者写完为止，即优先唤醒写者，防止写者饥饿。

这里采用一个排队信号量：queue。读写进程访问文件前都要在此信号量上排队。每个读进程最开始都要申请一下 queue 信号量，之后在真正做读操作前即让出(使得写进程可以随时申请到 queue)。而只有第一个写进程需要申请 queue，之后就一直占着不放了，直到所有写进程都完成后才让出。区别对待读写进程便可达到提高写进程优先级的目的。

另外再增加一个 write_cnt 以记录提出写访问申请和正在写的进程总数。

```
semaphore db = 1;  //访问锁，Guarantee the mutual exclusion between the readers and the writers.
semaphore queue = 1 //授权queue
semaphore rd_cnt_mutex = 1; //protect read_cnt
int read_cnt = 0;//正在读的读者数量

semaphore wrt_cnt_mutex = 1; //protect writer_cnt
int writer_cnt = 0;//申请写和正在写的进程总数

void Writerfunction(){
  While(true){
    down(&wrt_cnt_mutex);
      if (writer_cnt==0) down(&queue);  //只有第一个写者需要申请queue，申请到后占着不放
      writer_cnt++;   //记录写者数量
    up(&wrt_cnt_mutex);

    down(&db);       //如果有读者在读的话，这时候db锁在第一个读者手里，写不了。
      write();       //执行写操作临界区
    up(&db);

    down(&wrt_cnt_mutex);
      writer_cnt--;
      if (writer_cnt==0) up(&queue);     //直到最后一个写者写完释放queue，这时候读者才有机会拿
    up(&wrt_cnt_mutex);
  }
}

void Readerfunction(){
  While(true){
    prepare_read();
    down(&queue);      //首先要请求queue，只有没有写者的时候才能排队
      down(&read_cnt_mutex);
      if (read_cnt==0) down(&db);     //如果是第一个读者，且取得了queue授权，可以请求访问锁
      read_cnt++;
      up(&read_cnt_mutex);
    up(&queue);      //读者需要立即返还queue授权，以便后面的读者能进来，或者还给写者。

    read();          //执行读操作临界区

    down(&read_cnt_mutex);
      read_cnt--;
      if (read_cnt==0) up(&db);    //最后一个读者返还db锁
    up(&read_cnt_mutex);
  }
}
```