**北京航空航天大学 2023~2024第二学期**

**操作系统期中测试**

1. **判断题（正确的打√，错误的打╳，每题2分，共20分）**
2. 【**√**】同一进程中的不同线程间可以通过传递指向其栈上对象的指针来相互共享数据。
3. 【**√**】用户级线程不能修改自己的页表项。
4. 【**╳**】当进程被fork创建后，父进程和子进程中的同一变量将具有相同的虚拟内存地址，但物理内存地址是不相同的。
5. 【**╳**】消除虚拟内存抖动的最佳方法是增加 I/O 和CPU计算之间的并行程度，也就是让CPU与I/O都“忙”起来，所以增加可运行线程数量可以达到这一目的。
6. 【**╳**】大多数PC机上的操作系统支持多任务，但不支持多用户。
7. 【**√**】磁盘驱动器上只有一个 MBR（主引导记录），但可能有多个引导扇区。
8. 【**√**】多个进程可以同时对应到同一个程序/可执行文件。
9. 【**╳**】从一个进程到另一个进程的上下文切换无需在内核模式下执行操作系统代码即可完成。
10. 【**√**】动态重定位是指硬件在每次内存访问时动态实现地址的转换。
11. 【**√**】进程中的线程必须有自己独立的栈。
12. **单项选择题（每题2分，共20分）**
13. 【B】批处理系统的主要缺点是:

A.CPU的利用率不高 B.失去了交互性

C.不具备并发性 D.以上都不是

1. 【A】以下说法正确的是:

A. 两个不同进程对应的页表中可能包含内容相同的页表项

B. 虚拟地址空间总是大于物理地址空间

C. 在页式内存管理下，页面尺寸越小越有利于消除外碎片，提高内存使用效率

D. 段式内存管理的不同分段大小可不同，从而可消除外碎片，提高内存使用率

1. 【D】关于多级页表，下列说法**不正确**的是:
2. 能够减少页表占用内存的大小
3. 级数越多，平均访问内存的时间越长
4. 有效的页表项中都会存储页框号
5. 使用二级页表的平均访存性能优于一级页表

1. 【A】以下说法正确的是:
2. 进程上下文切换过程一定会陷入内核

B. 陷入内核一定会导致进程切换

C. 正在执行的程序不可以主动放弃CPU

D. 系统调用一定会导致进程上下文切换

1. 【B】在一个多进程操作系统中，以下说法正确的是:

A. 如果一个用户进程进入死循环，则其他进程永远不可能获得执行

B. 如果一个用户进程进入死循环，操作系统可以终止该用户进程执行

C. 如果一个用户进程执行了“跳转到0地址”的指令，操作系统内核会立即崩溃

D. 如果一个用户进程执行了“除以0”的指令后，操作系统内核会立即崩溃

1. 【C】关于PV操作**错误的**是：

A. 信号量如果使用不当，可能导致死锁

B. 进程执行P操作阻塞时，不会占用CPU资源

C. 进程A，B调用P(S)各一次后，信号量S的值与调用顺序有关

D. 信号量操作是原子操作

1. 【D】下列哪项属于反置页表的优点:
2. 查找页表项的速度快
3. 缺页处理速度快
4. 便于进程之间共享数据
5. 页表与逻辑地址空间大小无关
6. 【C】关于页面置换算法，以下说法**不正确**是：

A. 二次机会算法是对FIFO的改进

B. Aging算法是对LRU算法的高效近似实现

C. WSClock算法仅需要在页表中扫描

D. 工作集算法的思路是驱逐不在工作集中的页面

1. 【B】可变分区又称为动态分区，它是在系统运行过程中\_\_\_时动态建立的:

A. 作业未装入 B. 在作业装入

C. 在作业创建 D. 在作业完成

1. 【D】下列说法**错误**的是：

A. 覆盖可减少一个程序运行所需的空间

B. 覆盖对应用程序员不透明

C. 交换是由操作系统实现的

D. 覆盖在不同作业或程序之间进行

1. **填空题（每空2分，共20分）**
2. 某进程运行时依次访问的内存页面为：A B D D E F A A C F G D A C G D C E。采用FIFO算法进行页面置换，共为该进程分配了4个页框，初始时这4个页框均为空，则完成上述内存页面访问会产生 10 次缺页中断；如果采用LRU算法进行页面置换，其他条件不变，则会产生 12 次缺页中断。
3. 在FIFO等页面置换算法中Belady现象是指：   
    给进程分配的页面增加，进程缺页率反而提高的异常现象 。
4. 在一个操作系统中编译好的程序在另一个 ABI 兼容的操作系统中无需重新编译就能运行。
5. 内存中无法被利用的存储空间称为 碎片 。
6. 如果在一个单处理器系统中总共有n个进程，那么在阻塞队列中的进程个数取值范围是 [0, n] 。
7. 分析以下C语言程序段：

|  |
| --- |
| #include <stdio.h>  int a = 100;  int b;  int main() {  static int x;  int y = 10;  int \*p = &y, \*q = &b;  printf("a=%d, b=%d, p=0x%x\n", a, b, p);  } |

将上述程序编译成可执行文件，装载到OS时，在DATA段中分配存储空间的变量有： a 。

1. 请将以下Peterson算法中第13行和第18行补全，实现申请进入和退出临界区的功能：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | #define FALSE 0  #define TRUE 1  #define N 2  int turn;  int interested[N];  void enter\_region( int process )  {  int other;  other = 1 – process;  interested[process] = TRUE;  turn = process;  while( turn == process && interested[other] );  }  void leave\_region( int process )  {  interested[process] = false;  } |

1. 可变分区存储分配算法中， BestFit 算法总是挑选可以容纳作业的最小的空闲区进行存储分配。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| FIFO | A | B | D | D | E | F | A | A | C | F | G | D | A | C | G | D | C | E |
| 1 | A | A | A | A | A | F | F | F | F | F | F | D | D | D | D | D | D | E |
| 2 |  | B | B | B | B | B | A | A | A | A | A | A | A | A | A | A | A | E |
| 3 |  |  | D | D | D | D | D | D | C | C | C | C | C | C | C | C | C | C |
| 4 |  |  |  |  | E | E | E | E | E | E | G | G | G | G | G | G | G | G |
| 缺页 | Y | Y | Y |  | Y | Y | Y |  | Y |  | Y | Y |  |  |  |  |  | Y |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| LRU | A | B | D | D | E | F | A | A | C | F | G | D | A | C | G | D | C | E |
| 1 | A | A | A | A | A | F | F | F | F | F | F | F | F | C | C | C | C | C |
| 2 |  | B | B | B | B | B | A | A | A | A | A | D | D | D | D | D | D | D |
| 3 |  |  | D | D | D | D | D | D | C | C | C | C | A | A | A | A | A | E |
| 4 |  |  |  |  | E | E | E | E | E | E | G | G | G | G | G | G | G | G |
| 缺页 | Y | Y | Y |  | Y | Y | Y |  | Y |  | Y | Y | Y | Y |  |  |  | Y |

1. **存储管理（共15分）**

1. 在一个32位虚拟内存系统中，页面大小为4KB。（共2题，共8分）

（1）如果页表被全部占用，采用1级页表，需要多少页表项？采用20级页表呢？（4分）

（2）如果一个进程只分配1页内存，对于1级页表，需要分配多少页表项？对于20级页表呢？（4分）

2. 假设一个20位多级内存管理机制采用如下地址格式：（7分）

|  |  |  |
| --- | --- | --- |
| 虚拟段号（4位） | 虚拟页号（8位） | 页内偏移（8位） |

物理地址为16位，形式为：

|  |  |
| --- | --- |
| 物理页号（8位） | 页内偏移（8位） |

一个段表项指向一个物理内存中页表的基地址，一个页表包含一系列16位的页表项。段表、页表项格式、当前物理内存中内容见后面。利用后面所给信息，请在下表中写出以下指令的执行结果。

对于Load指令，如果成功执行，写出读入的数据（读取一个字节），否则写Error；

对于Store指令，如果成功执行，写OK，否则写Error。

|  |  |
| --- | --- |
| 指令 | 结果 |
| Load [0x30114] |  |
| Store [0x30115] |  |
| Load [0x41015] |  |
| Load [0x00115] |  |
| Store [0x00210] |  |
| Load [0x21202] |  |
| Load [0x11145] |  |

段表 (最大段号=3，除下面列出的外，无其他段表项)

|  |  |  |  |
| --- | --- | --- | --- |
| 段号 | 页表基地址 | 段内最大页面数量 | 有效状态 |
| 0 | 0x2000 | 0x20 | Valid |
| 1 | 0x1200 | 0x10 | Valid |
| 2 | 0x3100 | 0x40 | Invalid |
| 3 | 0x4000 | 0x20 | Valid |

页表项（PTE）格式

|  |  |
| --- | --- |
| 第1字节 | 第2字节 |
| 物理页框号 | 标志位，可能取值为：  0x00=Invalid  0x06=Valid, Read Only  0x07=Valid, Read/Write |

物理内存（大尾端）

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Address | +0 | +1 | +2 | +3 | +4 | +5 | +6 | +7 | +8 | +9 | +A | +B | +C | +D | +E | +F |
| 0x0000 | 0E | 0F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A | 1B | 1C | 1D |
| 0x0010 | 1E | 1F | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 2A | 2B | 2C | 2D |
| …. |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0x1010 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 4A | 4B | 4C | 4D | 4E | 4F |
| 0x1020 | 40 | 07 | 41 | 06 | 30 | 06 | 31 | 07 | 00 | 07 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0x1030 | 51 | 07 | 4F | 07 | 3F | 07 | 31 | 07 | 01 | 07 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0x1040 | 40 | 07 | 41 | 07 | 31 | 07 | 31 | 07 | 02 | 07 | 00 | 00 | 00 | 00 | 00 | 00 |
| …. |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0x2000 | 02 | 20 | 10 | 00 | 12 | 07 | 05 | 50 | 06 | 60 | 07 | 70 | 08 | 80 | 09 | 90 |
| 0x2010 | 0A | A0 | 0B | B0 | 0C | C0 | 0D | D0 | 0E | E0 | 0F | F0 | 10 | 01 | 11 | 11 |
| 0x2020 | 12 | 21 | 13 | 31 | 14 | 41 | 15 | 51 | 16 | 61 | 17 | 71 | 18 | 81 | 19 | 91 |
| 0x2030 | 10 | 06 | 11 | 00 | 12 | 07 | 40 | 07 | 41 | 07 | 00 | 00 | 00 | 00 | 00 | 00 |
| …. |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0x30F0 | 00 | 11 | 22 | 33 | 44 | 55 | 66 | 77 | 88 | 99 | AA | BB | CC | DD | EE | FF |
| 0x3100 | 01 | 12 | 23 | 34 | 45 | 56 | 67 | 78 | 89 | 9A | AB | BC | CD | DE | EF | 00 |
| 0x3110 | 02 | 13 | 24 | 35 | 20 | 07 | 68 | 79 | 8A | 9B | AC | BD | CE | DF | F0 | 01 |
| 0x3120 | 03 | 06 | 25 | 36 | 47 | 58 | 69 | 7A | 8B | 9C | AD | BE | CF | E0 | F1 | 02 |
| 0x3130 | 04 | 15 | 26 | 37 | 48 | 59 | 70 | 7B | 8C | 9D | AE | BF | D0 | E1 | F2 | 03 |
| …. |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0x4000 | 30 | 00 | 31 | 06 | 32 | 07 | 33 | 07 | 34 | 06 | 35 | 00 | 43 | 38 | 32 | 79 |
| 0x4010 | 50 | 28 | 84 | 19 | 71 | 69 | 39 | 93 | 75 | 10 | 58 | 20 | 97 | 49 | 44 | 59 |
| 0x4020 | 23 | 07 | 20 | 07 | 00 | 06 | 62 | 08 | 99 | 86 | 28 | 03 | 48 | 25 | 34 | 21 |

答:

1. 采用1级页表：2^20=1M页表项。

采用20级页表，每级页表在虚拟地址中仅占1位，能够区分2个页表项。页表项全部被占用，则相当于构建一个满二叉树。所以所有页表项数量为：2+4+…+2^20=2(2^20-1)，约2M页表项。

如果考虑自映射，2M页表项中有一半是重复的，因此答1M页算对。

1. 如果进程只占用1页内存，对于1级页表仍要分配1M页表项（并且要连续存储空间），而对于20级页表来说，仅需要20个级别每一级分配1个页表（2个页表项），总共需要40个页表项。

|  |  |
| --- | --- |
| 指令 | 结果 |
| Load [0x30114] | 0x20 |
| Store [0x30115] | Error(Access violation) |
| Load [0x41015] | Error(Bad Segment) |
| Load [0x00115] | Error(Access Violation) |
| Store [0x00210] | Ok |
| Load [0x21202] | Error(Bad Segment) |
| Load [0x11145] | Error(Segment Overflow) |

1. **页表自映射（5分）**

一个32位的虚拟存储系统有两级页表，其逻辑地址中，第22到31位是第一级页表索引，12位到21位是第二级页表索引，页内偏移占0到11位。一个进程的地址空间为4GB，如果从0x8C000000开始映射4MB的页表，请：

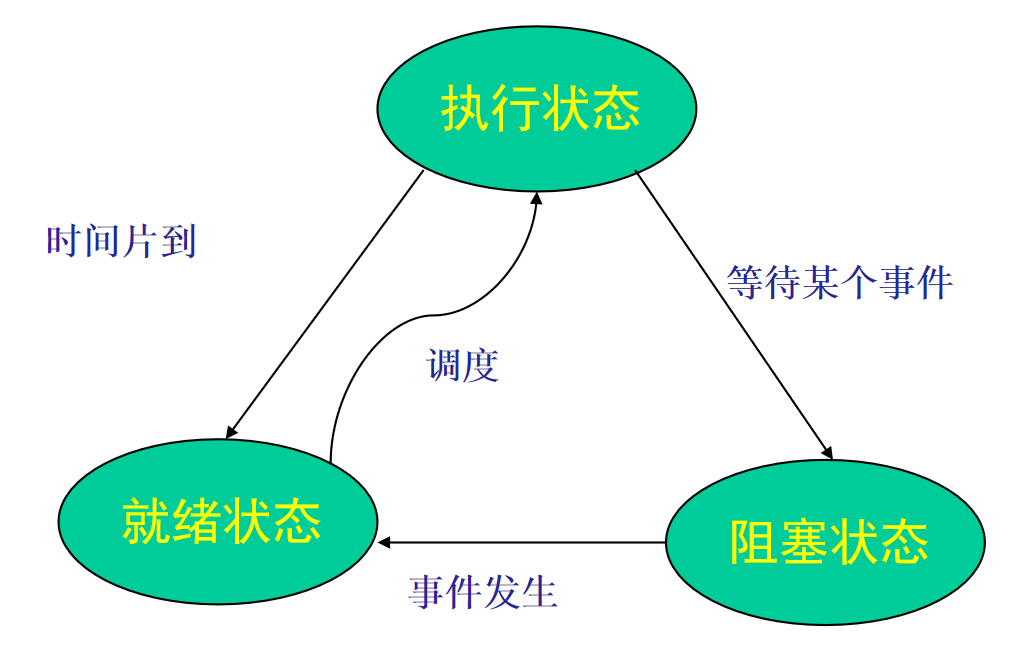
1. 给出一级页表的起始虚拟地址；（2分）
2. 给出一级页表中映射自己的表项的虚拟地址。（注意B代表字节，一个32位地址占4字节）（3分）

答：1). 0x8C230000; 2). 0x8C2308C0

1. **进程与线程（共10分）**

假设进程只有三种基本状态，画出进程的状态转换图，并举例说明这些转换发生的条件是什么。

答案：



就绪--> 运行

时间一到，调度程序选择一个进程运行

运行--> 就绪

运行进程用完了时间片

运行进程被中断，因为一高优先级进程处于就绪状态

运行--> 阻塞

当一进程所需的东西必须等待时

OS尚未完成服务

对一资源的访问尚不能进行

初始化I/O 且必须等待结果

等待某一进程提供输入(IPC)

阻塞--> 就绪

当所等待的事件发生时

1. **进程管理：信号量基础（10分）**

著名的计算机科学家Edsger Dijkstra提出了信号量机制，为实现进程同步和互斥提供了一种有效的方法。信号量的提出主要是为了解决传统同步/互斥方法因为“忙等待”而导致的CPU时间浪费问题以及进程优先级反转问题。对于信号量s，s.count表示信号量的值，s.queue表示信号量内部的队列。

1. 请分别解释调用一次P(s)和V(s)操作后，s.count和s.queue会产生什么样的变化。（本小题 4分）

答案：

1. 调用P(s)，s.count会减1，如果s.count < 0，则阻塞调用P(s)的进程，并将其放入s.queue；
2. 调用V(s)，s.count会加1，如果s.count <= 0, 则从s.queue中取出一个进程，将其置为就绪。
3. 实际上，当有多个进程并发调用P(s)或者V(s)时，也需要进行互斥控制。test-and-set指令经常用于实现自旋锁，请用test-and-set指令实现P(s)和V(s)操作，给出伪代码。**提示**：test-and-set(boolean\* lock)的语义是：如果\*lock为1，则test-and-set(lock)返回值为1；否则，将\*lock置1，返回0; \*lock的初始值为0。（本小题6分）

答案：

1. P(s)

while(test-and-set(lock) == 1);

s.count --;

if(s.count < 0){

block进程；

将进程放到s.queue;

}

\*lock = 0;

1. V(s)

while(test-and-set(lock) == 1);

s.count ++;

if(s.count <= 0) 从s.queue中取出一个进程，置为就绪态；

\*lock = 0;