1. 假设磁盘上有空文件 file.txt。程序运行过程中的所有系统调用均成功。

|  |
| --- |
| **int** main() {  **int** fd1 = open("file.txt", O\_RDWR|O\_CREAT, S\_IRUSR|S\_IWUSR);  **int** fd2 = open("file.txt", O\_RDWR|O\_CREAT, S\_IRUSR|S\_IWUSR);  printf("%d %d\n", fd1, fd2);  // A  write(fd1, "012", 3);  write(fd2, "ab", 2);  dup2(fd1, fd2);  // B  write(fd1, "123", 3);  write(fd2, "cd", 2);  close(fd1);  // C  close(fd2);  **return** 0;  } |

已知在程序执行到A处时，画出Linux三级表的结构如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Descriptor |  | Open file |  | v-node |
| 0 |  | refcnt  = 1 |  |  |
| 1 |  |  |  |
| 2 |  |  |  | file.txt |
| 3 |  |  |  |
| 4 |  | refcnt  = 1 |  |  |
| 5 |  |  |  |

(1)请画出程序在执行到B,C处时Linux三级表的结构

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Descriptor |  | Open file |  | v-node |
| 0 |  | refcnt  = (**2**) |  |  |
| 1 |  |  |  |
| 2 |  |  |  | file.txt |
| 3 |  |  |  |
| 4 |  | refcnt  = (**0**) |  |  |
| 5 |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Descriptor |  | Open file |  | v-node |
| 0 |  | refcnt  = (**1**) |  |  |
| 1 |  |  |  |
| 2 |  |  |  | file.txt |
| 3 |  |  |  |
| 4 |  | refcnt  = (**0**) |  |  |
| 5 |  |  |  |

(2)程序结束时，标准输出上的内容是\_**3 4**\_，file.txt中的内容是\_**ab2123cd**\_.

1. 判断以下说法的正确性

|  |  |
| --- | --- |
| (**√**) | 目录(directory)是一种特殊的文件，包含一组链接(link)，每个链接将一个文件名映射到一个文件。 |
| (**×**) | 关闭一个已经关闭的描述符时，不会出错 |
| (**√**) | 在进程调用fork()之后，子进程会继承父进程的描述符表(file descriptor table)，也会继承stdio的缓冲区 |
| (**√**) | 在编写网络程序时，应该使用Unix I/O而不是标准I/O |

1. 假设某进程有且仅有五个已打开的文件描述符：0~4，分别引用了五个不同的文件，尝试运行以下代码：

|  |
| --- |
| dup2(3,2); dup2(0,3); dup2(1,10); dup2(10,4); dup2(4,0); |

关于得到的结果，说法正确的是：**A**

A. 运行正常完成，现在有四个描述符引用同一个文件

B. 运行正常完成，现在进程共引用四个不同的文件

C. 由于试图从一个未打开的描述符进行复制，发生错误

D. 由于试图向一个未打开的描述符进行复制，发生错误

1. 假设磁盘上有空文件 file.txt。程序运行过程中的所有系统调用均成功。缓冲区足够

大，且 stdout 只有在关闭文件、换行与 fflush 的情况下才会刷新缓冲区。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **int** main() {  pid\_t pid; int child\_status;  **int** fd1 = open("file.txt", O\_RDWR|O\_CREAT, S\_IRUSR|S\_IWUSR);  if ((pid = fork()) > 0) { // Parent  printf("P:%d ", fd1);  write(fd1, "123", 3);  waitpid(pid, &child\_status, 0);  } else { // Child  printf("C:%d ", fd1);  write(fd1, "45", 2);  }  close(fd1); **return** 0;  } | | | | |
| (1)在子进程关闭fd1前，画出Linux三级表的结构如下 | | | | |
| Descriptor |  | Open file |  | v-node |
| Parent 3 |  | refcnt  = (**2**) |  |  |
| Parent 4 |  |  |  |
|  |  |  |  | file.txt |
|  |  |  |  |
| Child 3 |  | refcnt  = (**0**) |  |  |
| Child 4 |  |  |  |

(2)程序结束时，标准输出上的内容是\_\_\_\_\_\_\_，file.txt中的内容是\_\_\_\_\_\_\_\_.

**C:3 P:3 ;12345 或 45123 注意:C一定在P前输出**

1. 根据本课程介绍的 Intel x86-64 存储系统，填写表格中某一个进程从用户态切换至内核态时，和进程切换时对 TLB 和 cache 是否必须刷新 **A**

A. ①不必刷新 ②不必刷新 ③刷新 ④不必刷新

B. ①不必刷新 ②不必刷新 ③不必刷新 ④不必刷新

C. ①刷新 ②不必刷新 ③刷新 ④刷新

D. ①刷新 ②不必刷新 ③不必刷新 ④刷新

1. 下列关于虚存和缓存的说法中，正确的是： **D**

A. TLB 是基于物理地址索引的高速缓存

B. 多数系统中，SRAM 高速缓存基于虚拟地址索引

C. 在进行**线程**切换后，TLB 条目绝大部分会失效

D. 多数系统中，在进行进程切换后，SRAM 高速缓存中的内容不会失效

1. 对于虚拟存储系统,一次访存过程中,下列命中组合不可能发生的是**D**.

A.TLB 未命中, Cache 未命中, Page 未命中

B.TLB 未命中,Cache 命中, Page 命中

C.TLB 命中, Cache 未命中, Page 命中

D.TLB 命中, Cache 命中, Page 未命中

1. 关于写时复制（copy-on-write, COW）技术的说法，不正确的是： **D**

A. 写时复制既可以发生在父子进程之间，也可以发生在对等线程之间

B. 写时复制既需要硬件的异常机制，也需要操作系统软件的配合

C. 写时复制既可以用于普通文件，也可以用于匿名文件

D. 写时复制既可以用于共享区域，也可以用于私有区域

9. 假设有一台 64 位的计算机的物理页块大小是 8KB,采用三级页表进行虚拟地址寻址,它的虚拟地址的 VPO(Virtual Page Offset,虚拟页偏移)有 13 位,问它的虚拟地址的 VPN(Virtual Page Number,虚拟页号码)有多少位?**C**

A. 20

B. 27

C. 30

D. 33

**本题考查对页表组成的理解.页块大小为 8KB,即 2^13 byte.在 64 位机器上,一个页表条目为8byte.故共有页表条目 2^10 项,故每一级页表可以表示 10 位地址.因此三级页表存储, 共需要10\*3=30 位.**

10. 进程 P1 通过 fork()函数产生一个子进程P2.假设执行 fork()函数之前, 进程 P1 占用了 53 个(用户态的)物理页,则 fork 函数之后,进程 P1 和进程P2 共占用\_\_\_\_\_\_\_\_个(用户态的)物理 页;假设执行 fork()函数之前进程 P1中有一个可读写的物理页,则执行 fork()函数之后,进程 P1 对该物理页的页表项权限为**B**.

A. 53,读写

B. 53,只读

C. 106,读写

D. 106,只读

11. Intel的IA32体系结构采用二级页表，称第一级页表为页目录（Page Directory），第二级页表为页表（Page Table）。页面的大小为4KB，页表项4字节。以下给出了页目录与若干页表中的部分内容，例如，页目录中的第1个项索引到的是页表3，页表1中的第3个项索引到的是物理地址中的第5个页。则十六进制逻辑地址8052CB经过地址转换后形成的物理地址应为十进制的（ **B**）。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 页目录 | | 页表1 | | 页表2 | | 页表3 | |
| VPN | 页表号 | VPN | 页号 | VPN | 页号 | VPN | 页号 |
| 1 | 3 | 3 | 5 | 2 | 1 | 2 | 9 |
| 2 | 1 | 4 | 2 | 4 | 4 | 3 | 8 |
| 3 | 2 | 5 | 7 | 8 | 6 | 5 | 3 |

1. 21195
2. 29387
3. 21126
4. 47195

**4KB=2^12,所以页内地址有 12 位.4KB/4B=1K,所以页目录和每个页表中的页表项数为 1K 个.因此,在 32 位的虚拟地址中,最低的 12 位为页内地址(Offset),最高的 10 位为页目录的虚拟地址(Dir),中间 10 位为页表的虚拟地址(Table).**

**十六进制逻辑地址 8052CB 转换为二进制后为 100000000101 001011001011,Dir 为 10,即 10 进制的 2,在表中对应到页表 1.**

**Table 为101,即 10 进制的 5,在表中对应到物理页面 7.因此,物理地址应为 7 的二进制 111 和 Offset 的拼合,即 111001011001011,转换为十进制为 29387,答案为 B.**

12. 假定整型变量 A 的虚拟地址空间为 0x12345cf0,另一整形变量 B 的虚拟地址 0x12345d98,假定一个 page 的长度为 0x1000 byte,A 的物理地址数值和 B 的物理地址数值关系应该为: **B,在同一页中**

A.A 的物理地址数值始终大于 B 的物理地址数值

B.A 的物理地址数值始终小于 B 的物理地址数值

C.A 的物理地址数值和 B 的物理地址数值大小取决于动态内存分配策略

D.无法判定两个物理地址值的大小

13. 下列与虚拟内存有关的说法中哪些是不对的？**D**

A.操作系统为每个进程提供一个独立的页表,用于将其虚拟地址空间映射到物理地址空间.

B.MMU使用页表进行地址翻译时,虚拟地址的虚拟页面偏移与物理地址的物理页面偏移是相同的.

C.若某个进程的工作集大小超出了物理内存的大小,则可能出现抖动现象.

D.动态内存分配管理,采用双向链表组织空闲块,使得首次适配的分配与释放均是空闲块数量的线性时间.

14. 在 Core i7 中,关于虚拟地址和物理地址的说法,不正确的是:**B**

A.VPO = CI + CO

B.PPN = TLBT + TLBI

C.VPN1 = VPN2 = VPN3 = VPN4

D.TLBT + TLBI = VPN

15. 已知某系统页面长 8KB,页表项 4 字节,采用多层分页策略映射 64 位虚拟地址空间.若限定最高层页表占 1 页,则它可以采用多少层的分页策略？

A.3 层B.4 层C.5 层D.6 层

**C.由题意,64 位虚拟地址的虚拟空间大小为 264.页面长为 8KB,页表项 4 字节,所以一个页面可存放 2K 个表项.由于最高层页表占 1 页,也就是说其页表项个数最多为 2K 个,每一项对应一页,每页又可存放 2K 个页表项,依次类推可知,采用的分页层数为:5 层.**

## 第五题（10分）

以下程序运行时系统调用全部正确执行，且每个信号都被处理到。请给出代码运行后所有可能的输出结果。

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>

int c = 1;

void handler1(int sig) {

c++;

printf("%d", c);

}

int main() {

signal(SIGUSR1, handler1);

sigset\_t s;

sigemptyset(&s);

sigaddset(&s, SIGUSR1);

sigprocmask(SIG\_BLOCK, &s, 0);

int pid = fork()?fork():fork();

if (pid == 0) {

kill(getppid(), SIGUSR1);

printf("S");

sigprocmask(SIG\_UNBLOCK, &s, 0);

exit(0);

} else {

while (waitpid(-1, NULL, 0) != -1);

sigprocmask(SIG\_UNBLOCK, &s, 0);

printf("P");

}

return 0;

}

答：**共 3 种： S2PS2P SS2P2P S2SP2P**