

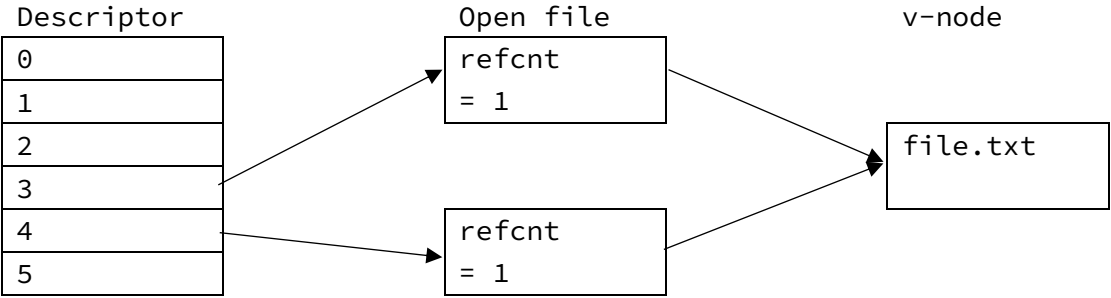
1. 假设磁盘上有空文件 `file.txt`。程序运行过程中的所有系统调用均成功。

```

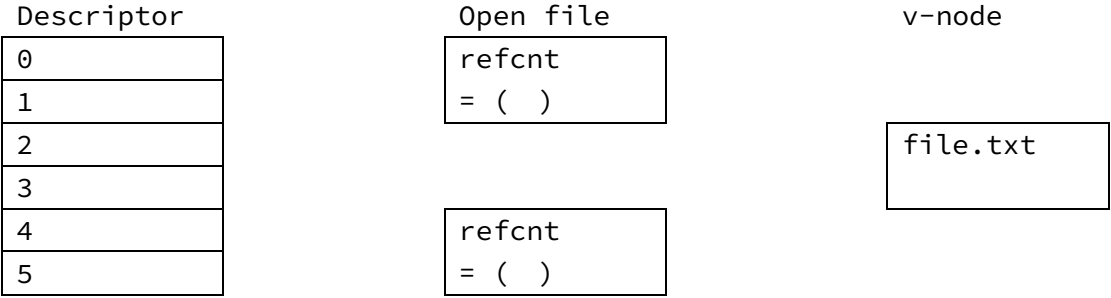
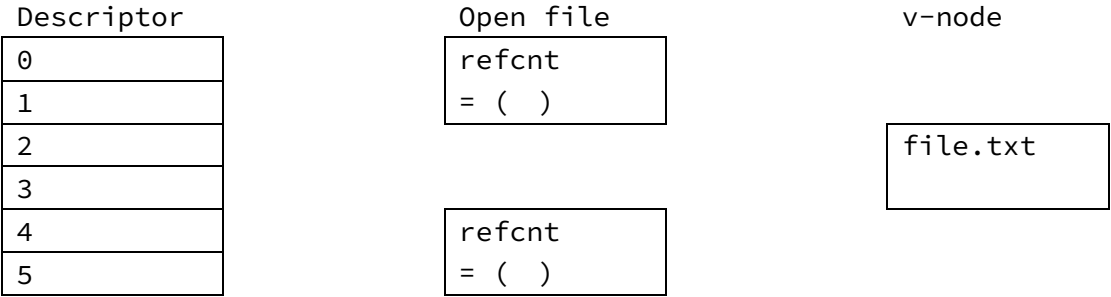
int main() {
    int fd1 = open("file.txt", O_RDWR|O_CREAT, S_IRUSR|S_IWUSR);
    int fd2 = open("file.txt", O_RDWR|O_CREAT, S_IRUSR|S_IWUSR);
    printf("%d %d\n", fd1, fd2);
    // A
    write(fd1, "012", 3);
    write(fd2, "ab", 2);
    dup2(fd2, fd1);
    // B
    write(fd1, "123", 3);
    write(fd2, "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：



(1) 请画出程序在执行到 B,C 处时 Linux 三级表的结构



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

2. 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

3. 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);
```

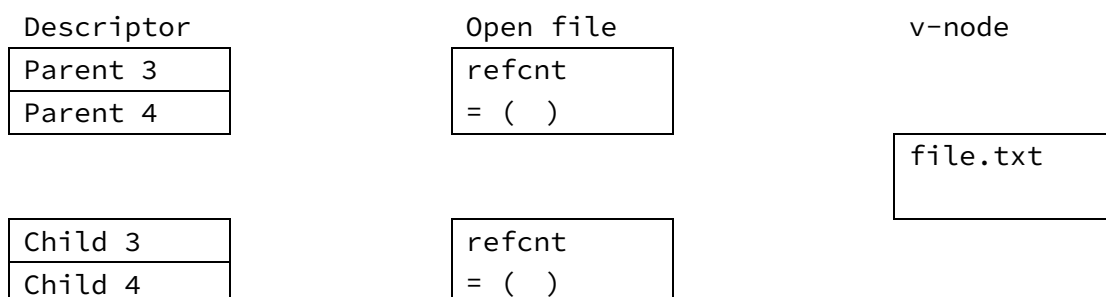
关于得到的结果,说法正确的是:

- A. 运行正常完成, 现在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

4. 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



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

5. 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. 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
6. 下列关于虚存和缓存的说法中, 正确的是:  
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
7. 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\_\_\_\_\_。  
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
8. 关于写时复制 (copy-on-write, COW) 技术的说法, 不正确的是:  
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?  
A. 20  
B. 27  
C. 30  
D. 33
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 \_\_\_\_\_。  
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 ( )。

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

- A. 21195  
B. 29387  
C. 21126  
D. 47195

12. 假定整型变量 A 的虚拟地址空间为 0x12345cf0, 另一整型变量 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:

- 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?

- 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:

- A.  $VP0 = CI + C0$   
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

## 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:

得分

第六题（15 分）

为了提升虚拟内存地址的转换效率，降低遍历两级页表结构所带来的地址转换开销，英特尔处理器中引入了大页 TLB，即一个 TLB 项可以涵盖整个 4MB 对齐的地址空间（针对 32 位模式）。只要设置页目录页中页目录项（PDE）的大页标志位，即可让 MMU 识别这是一个大页 PDE，并加载到大页 TLB 项中。大页 PDE 中记录的物理内存页面号必须是 4MB 对齐的，并且整个连续的 4MB 内存均可统一通过该大页 PDE 进行地址转换。

在 32 位的 Linux 系统中，为了方便访问物理内存，内核将地址 0~768MB 间的物理内存映射到虚拟内存地址 3GB~3GB+768MB 上，并通过大页 PDE 进行该区间的地址转换。任何 0~768MB 的物理内存地址可以直接通过加 3G（0xC0000000）的方式得到其虚拟内存地址。在内核中，除了该区间的内存外，其他地址的内存通常都通过普通的两级页表结构来进行地址转换。

假设在我们使用的处理器中有 2 个大页 TLB 项，其当前状态如下：

索引号	TLB 标记	页面号	有效位
0	0xC48	0x04800	1
1	0xC9C	0x09C00	1

有 4 个普通 TLB 项，当前的状态如下：

索引号	TLB 标记	页面号	有效位
0	0xF8034	0x04812	1
1	0xF8033	0x09812	1
2	0xF4427	0x12137	1
3	0xF44AE	0x17343	1

当前页活跃的目录页（PD）中的部分 PDE 的内容如下：

PDE 索引	页面号	其他标志	大页位	存在位
786	0x04800	...	1	1
807	0x09C00	...	1	1
977	0x09C33	...	0	1
992	0x09078	...	0	1

注：普通页面大小为 4KB，并且 4KB 对齐。每个页面的页面号为其页面起始物理地址除以 4096 得到。大页由连续 1024 个 4KB 小页组成，且 4MB 对齐。

1. 分析下面的指令序列，

```
movl $0xC48012024, %ebx
movl $128, (%ebx)
movl $0xF8034000, %ecx
movl $36(%ecx), %eax
```

请问，执行完上述指令后，eax 寄存器中的内容是（ ）；在执行上述指令过程中，共发生了（ ）次 TLB miss？同时会发生（ ）次 page fault？

注：不能确定时填写“--”。

2. 请判断下列页面号对应的页面中, 哪些一定是页表页? 哪些不是? 哪些不确定?

页面号	是否为页表页 (是/不是/不确定)
0x04800	4
0x09C33	5
0x09812	6

3. 下列虚拟地址中哪一个对应着够将虚拟内存地址 0xF4427048 映射到物理内存地址 0x14321048 的页表项 ( ) ?

- (A) 0x09C33027                      (B) 0xC9C3309C  
(C) 0xC9C33027                      (D) 0x09C3309C

通过上述虚拟地址, 利用 `movl` 指令修改对应的页表项, 完成上述映射, 在此过程中, 是否会产生 TLB miss? ( ) (回答: 会/不会/不确定)

修改页表项后, 是否可以立即直接使用下面的指令序列将物理内存地址 0x14321048 开始的一个 32 位整数清零? 为什么?

```
movl $0xF4427048, %ebx
movl $0, (%ebx)
```

答: