

# 北京大学信息科学技术学院考试试卷

考试科目：计算机系统导论 姓名：\_\_\_\_\_ 学号：\_\_\_\_\_

考试时间：2019 年 1 月 7 日 小班号：\_\_\_\_\_ 小班教师：\_\_\_\_\_

题号	一	二	三	四	五	六	七	八	总分
分数									
阅卷人									

## 北京大学考场纪律

1、考生进入考场后，按照监考老师安排隔位就座，将学生证放在桌面上。无学生证者不能参加考试；迟到超过 15 分钟不得入场。在考试开始 30 分钟后方可交卷出场。

2、除必要的文具和主考教师允许的工具书、参考书、计算器以外，其它所有物品（包括空白纸张、手机、或有存储、编程、查询功能的电子用品等）不得带入座位，已经带入考场的必须放在监考人员指定的位置。

3、考试使用的试题、答卷、草稿纸由监考人员统一发放，考试结束时收回，一律不准带出考场。若有试题印制问题请向监考教师提出，不得向其他考生询问。提前答完试卷，应举手示意请监考人员收卷后方可离开；交卷后不得在考场内逗留或在附近高声交谈。未交卷擅自离开考场，不得重新进入考场答卷。考试结束时间到，考生立即停止答卷，在座位上等待监考人员收卷清点后，方可离场。

4、考生要严格遵守考场规则，在规定时间内独立完成答卷。不准交头接耳，不准偷看、夹带、抄袭或者有意让他人抄袭答题内容，不准接传答案或者试卷等。凡有违纪作弊者，一经发现，当场取消其考试资格，并根据《北京大学本科考试工作与学术规范条例》及相关规定严肃处理。

5、考生须确认自己填写的个人信息真实、准确，并承担信息填写错误带来的一切责任与后果。

学校倡议所有考生以北京大学学生的荣誉与诚信答卷，共同维护北京大学的学术声誉。

以下为试题和答题纸，共 \_\_\_\_\_ 页。

得分

第一题 单项选择题（每小题 2 分，共 30 分）

注：选择题的回答请填写在下表中。

题号	1	2	3	4	5	6	7	8	9	10
回答										
题号	11	12	13	14	15	16	17	18	19	20
回答						/	/	/	/	/

1. 在如下的代码中，下列关于局部性的说法正确的是：

```
int add (int *a, int n){
    int sum = 0;
    for (int i = 0; i < n; ++i)
        sum += a[i];
    return sum;
}
```

- A. 变量 sum 具有良好的空间局部性.
- B. 数组 a 具有良好的空间局部性.
- C. 数组 a 具有良好的时间局部性
- D. 以上说法都正确

2. 有如下结构定义和程序片段

```
struct A
{
    char c;
    int i;
    double d;
    int array[10];
};
```

```
struct B
{
    int array[10];
```

```

    double d;
    char c;
    int i;
};

void foo(struct A *pa, struct B *pb, int index)
{
    pb->i = pa->array[index];
}

```

在 Linux 下使用 GCC 编译器，仅采用 -O2 选项，上述代码对应的汇编语言是：（将选项依次填入空格内）

```

movslq %edx, %rdx
movl __(%rdi,%rdx,__), %eax
movl %eax, __(%rsi)

```

A. (16, 4, 52)    B. (24, 4, 52)    C. (16, 4, 49)    D. (24, 4, 49)

3. 下面关于缓存替换策略的说法哪个是正确的 (N 为 cache 的大小)

- A. FIFO 的性能总是优于随机替换
- B. LRU 适用于数组的顺序访问
- C. 假定 cache 初始状态为空，若某一输入片段使用 LRU 造成了  $N + 1$  次 miss，则对任意策略至少产生两次 miss
- D. 对于同一确定性替换策略，增大 N 的大小一定减少 miss 次数

4. 下列关于链接技术的描述，错误的是（ ）

- A. 在 Linux 系统中，对程序中全局符号的不恰当定义，会在链接时刻进行报告。
- B. 在使用 Linux 的默认链接器时，如果有多个弱符号同名，那么会从这些弱符号中任意选择一个占用空间最大的符号。

- C. 编译时打桩 (interpositioning) 需要能够访问程序的源代码, 链接时打桩需要能够访问程序的可重定位对象文件, 运行时打桩只需要能够访问可执行目标文件。
- D. 链接器的两个主要任务是符号解析和重定位。符号解析将目标文件中的全局符号都绑定到唯一的定义, 重定位确定每个符号的最终内存地址, 并修改对那些目标的引用。

5. 关于进程, 以下说法正确的是:

- A. 没有设置模式位时, 进程运行在用户模式中, 允许执行特权指令, 例如发起 I/O 操作。
- B. 调用 `waitpid(-1, NULL, WNOHANG & WUNTRACED)` 会立即返回: 如果调用进程的所有子进程都没有被停止或终止, 则返回 0; 如果有停止或终止的子进程, 则返回其中一个的 ID。
- C. `execve` 函数的第三个参数 `envp` 指向一个以 `null` 结尾的指针数组, 其中每一个指针指向一个形如 "name=value" 的环境变量字符串。
- D. 进程可以通过使用 `signal` 函数修改和信号相关联的默认行为, 唯一的例外是 `SIGKILL`, 它的默认行为是不能修改的。

6. 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

7. 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。
8. 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
9. 虚拟内存中两层页表和单层页表相比，最主要的优势在于：
- A. 更快的地址翻译速度
  - B. 能够提供页面更加精细的保护措施
  - C. 能够充分利用代码的空间局部性
  - D. 能够充分利用稀疏的内存使用模式
10. 下面有关计算机网络概念的叙述中，**正确**的是
- A. 大写字母的 Internet 用来描述互联网的一般概念，而小写字母的 internet 用来描述一种具体的实现，也就是全球 IP 互联网。
  - B. 在一个基于集线器(hub)的以太网(Ethernet)中，如果往一台主机发送一段数据帧(frame)，那么其他主机无法看到这个帧。
  - C. IP 协议提供基本的命名方法和递送机制，因此我们能够借助 IP 协议，从一台主机往另一台主机发送包，即使两台主机不在同一个 LAN 内。
  - D. 当一段数据通过路由器，从 LAN1 被发送到 LAN2 时，附加的互联网络包头和局

域网帧头保持不变。

11. 下面有关套接字接口(Socket API)的叙述中, **错误**的是
- A. 套接字接口常常被用来创建网络应用
  - B. Windows 10 系统没有实现套接字接口
  - C. `getaddrinfo()` 和 `getnameinfo()` 可以被用于编写独立于特定版本的 IP 协议的程序
  - D. `socket()` 函数返回的描述符, 可以使用标准 Unix I/O 函数进行读写
12. 使用浏览器打开网页 `www.pku.edu.cn` 的过程中, 下列网络协议中, 可能会被用到的网络协议有\_\_\_个
- ① DNS   ② TCP   ③ IP   ④ HTTP
- A.   1      B.   2      C.   3      D.   4

13. 下面关于线程安全和可重入的描述中，哪一个是正确的？
- A. 如果一个函数的所有参数都是值传递的且无返回值，该函数一定是可重入的
  - B. 函数的可重入版本一定比不可重入版本高效
  - C. 可重入函数一定是线程安全的
  - D. 以上说法都不正确

14. 下列关于进程与线程的描述中，哪一个是不正确的？
- A. 一个进程可以包含多个线程
  - B. 进程中的各个线程共享进程的代码、数据、堆和栈
  - C. 进程中的各个线程拥有自己的线程上下文
  - D. 线程的上下文切换比进程的上下文切换快

15. 给定下列代码片段：

```
char **ptr; /* global var */
int main(int main, char *argv[]) {
    long i; pthread_t tid;
    char *msgs[2] = {" Hello from foo", " Hello from bar" };
    ptr = msgs;
    for (i = 0; i < 2; i++)
        Pthread_create(&tid, NULL, thread, (void *)i);
    Pthread_exit(NULL);}

void *thread(void *vargp)
{
    long myid = (long)vargp;
    static int cnt = 0;
    printf("[%ld]:  %s (cnt=%d)\n", myid, ptr[myid], ++cnt);
    return NULL;
}
```

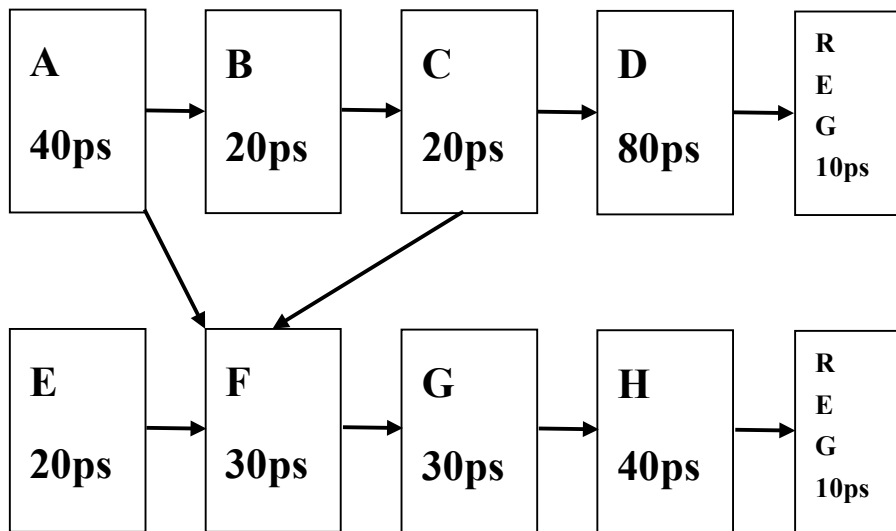
下列哪一组变量集合是对等线程 1 引用的？

- A. ptr, cnt, i.m, msgs.m, myid.p0, myid.p1
- B. ptr, cnt, msgs.m, myid.p0
- C. ptr, cnt, msgs.m, myid.p1
- D. ptr, cnt, i.m, msgs.m, myid.p1

得分

第二题（12 分）

如图所示，每个模块表示一个单独的组合逻辑单元，每个单元的延迟以及数据依赖关系已在图中标出。通过在两个单元间添加寄存器的方式，可以对该数据通路进行流水化改造。假设每个寄存器的延迟为 10ps。



1) 如果改造为一个二级流水线，为获得最大的吞吐率，该寄存器应在哪里插入？请计算该流水线的吞吐率，并说明计算过程。结果可以是分数形式也可以是小数形式。

（6 分）

2) 如果改造为一个三级流水线，为获得最大的吞吐率，寄存器应在哪里插入？请计算该流水线的吞吐率，并说明计算过程。结果可以是分数形式也可以是小数形式。（请给出至少两种实现最大吞吐率的设计方案）



(6 分)

得分

第三题（10 分）

本题基于下列 m.c 及 foo.c 文件所编译生成的 m.o 和 foo.o，编译过程未加优化选项。

<pre>//m.c void foo(); int buf[2] = {1, 2}; int main() {     foo();     return 0; }</pre>	<pre>//foo.c extern int buf[]; int *bufp0 = &amp;buf[0]; int *bufp1; void foo() {     static int count = 0;     int temp;     bufp1 = &amp;buf[1];     temp = *bufp0;     *bufp0 = *bufp1;     *bufp1 = temp;     count++; }</pre>
---	--

对于每个 foo.o 中定义和引用的符号，请用“是”或“否”指出它是否在模块 foo.o 的 .symtab 节中有符号表条目。如果存在条目，则请指出定义该符号的模块（foo.o 或 m.o）、符号类型（局部、全局或外部）以及它在模块中所处的节；如果不存在条目，则请将该行后继空白处标记为“/”。

符号	.symtab 条目?	符号类型	定义符号的模块	节
bufp0	是	全局	foo.o	.data
buf				
bufp1				
foo				
temp				
cnt				

下图左边给出了 m.o 和 foo.o 的反汇编文件，右边给出了采用某个配置链接成可执行程序后再反汇编出来的文件。根据答题需要，其中的信息略有删减。

0000000000.....<main>: 55                   push   %rbp	00000000000000fe8 <main>: fe8: 55                   push   %rbp
--	--

<pre> 48 89 e5      mov     %rsp,%rbp b8 00 00 00 00    mov     \$0x0,%eax e8 00 00 00 00    callq  e &lt;main+0xe&gt;      ① b8 00 00 00 00    mov     \$0x0,%eax 5d             pop     %rbp c3             req </pre>	<pre> fe9: 48 89 e5      mov     %rsp,%rbp fec: b8 00 00 00 00    mov     \$0x0,%eax ff1: e8      ①      callq  1000 &lt;foo&gt; ff6: b8 00 00 00 00    mov     \$0x0,%eax ffb: 5d             pop     %rbp ffc: c3             retq </pre> <p>.....略去部分和答题无关的信息.....</p>
<pre> 0000000000.....&lt;foo&gt;: 55             push    %rbp 48 89 e5      mov     %rsp,%rbp 48 c7 05 00 00 00 00 00 00 00 00                                 movq     \$0x0,0x0(%rip)                                 ③② 48 8b 05 00 00 00 00 00 mov     0x0(%rip),%rax      ④ 8b 00             mov     (%rax),%eax 89 45 fc      mov     %eax,-0x4(%rbp) 48 8b 05 00 00 00 00 00 mov     0x0(%rip),%rax      ⑤ 48 8b 15 00 00 00 00 00 mov     0x0(%rip),%rdx      ⑥ 8b 12             mov     (%rdx),%edx 89 10             mov     %edx,(%rax) 48 8b 05 00 00 00 00 00 mov     0x0(%rip),%rax      ⑦ 8b 55 fc      mov     -0x4(%rbp),%edx 89 10             mov     %edx,(%rax) 8b 05 00 00 00 00 00 00 mov     0x0(%rip),%eax ⑧ 83 c0 01      add     \$0x1,%eax 89 05 00 00 00 00 00 00 mov     %eax,0x0(%rip) ⑨ 90             nop 5d             pop     %rbp c3             retq </pre>	<pre> 00000000000001000 &lt;foo&gt;: 1000: 55             push    %rbp 1001: 48 89 e5      mov     %rsp,%rbp 1004: 48 c7 05 00 00 00 00 00 00 00 00                                 movq     ② , ③                                 (%rip) 100f: 48 8b 05 ?? ?? ?? ?? mov     0x????(%rip),%rax 1016: 8b 00             mov     (%rax),%eax 1018: 89 45 fc      mov     %eax,-0x4(%rbp) 101b: 48 8b 05 00 00 00 00 00 mov     ⑤ (%rip),%rax 1022: 48 8b 15 00 00 00 00 00 mov     0x????(%rip),%rdx 1029: 8b 12             mov     (%rdx),%edx 102b: 89 10             mov     %edx,(%rax) 102d: 48 8b 05 00 00 00 00 00 mov     0x????(%rip),%rax 1034: 8b 55 fc      mov     -0x4(%rbp),%edx 1037: 89 10             mov     %edx,(%rax) 1039: 8b 05 00 00 00 00 00 00 mov     0x????(%rip),%eax 103f: 83 c0 01      add     \$0x1,%eax 1042: 89 05 00 00 00 00 00 00 mov     %eax, ⑨ (%rip) 1048: 90             nop 1049: 5d             pop     %rbp 104a: c3             retq </pre> <p>.....略去部分和答题无关的信息.....</p> <pre> 00000000000002330 &lt;buf&gt;: .....略去部分和答题无关的信息..... 00000000000002338 &lt;bufp0&gt;: .....略去部分和答题无关的信息..... 00000000000003024 &lt;count.1837&gt;: .....略去部分和答题无关的信息..... 00000000000003028 &lt;bufp1&gt;: </pre>

在上图对所涉及到的重定位条目进行用数字①至⑨进行了标记, 请根据下表中所提供的重定位条目信息, 计算相应的重定位引用值并填写下表。

编号	重定位条目信息	应填入的重定位引用值
①	r.offset = 0xa r.type = R_X86_64_PC32 r.symbol = 本题不提供 r.addend = -4	
②	r.offset = 0xb r.type = R_X86_64_32 r.symbol = buf r.addend = +4	
③	r.offset = 0x7 r.symbol = bufp1	

	r.type = R_X86_64_PC32	r.addend = -8	
⑤	r.offset = 0x1e r.type = R_X86_64_PC32	r.symbol = bufp0 r.addend = -4	
⑨	r.offset = 0x44 r.type = R_X86_64_PC32	r.symbol = 本题不提供 r.addend = -4	

得分

#### 第四题（10 分）

Bob 是一名刚刚学完异常的同学，他希望通过配合 kill 和 signal 的使用，能让两个进程向同一个文件中交替地打印出字符。可惜他的 tshlab 做得不过关，导致他写的这个程序有各种 BUG。你能帮帮他吗？

```

1  #include "csapp.h"
2  #define MAXN 6
3  int parentPID = 0;
4  int childPID = 0;
5  int count = 1;
6  int fd1 = 1;
7  void handler1() {
8      if (count > MAXN)
9          return;
10     for (int i = 0; i < count; i++)
11         write(fd1, "+", 1);
12         X
13     kill(parentPID, SIGUSR2);
14 }
15 void handler2() {
16     if (count > MAXN)
17         return;
18     for (int i = 0; i < count; i++)
19         write(fd1, "-", 1);
20         Y
21     kill(childPID, SIGUSR1);
22 }
23
24 int main() {
25     signal(SIGUSR1, handler1);
26     signal(SIGUSR2, handler2);
27     parentPID = getpid();
28     childPID = fork();
29     fd1 = open("file.txt", O_RDWR);

```

```
30     if (childPID) {
31         Z
32         kill(childPID, SIGUSR1);
33     }
34     exit(0);
35 }
```

**注意：**假设程序能在任意时刻被系统打断、调度，并且调度的时间切片大小是不确定的，可以足够地长。在每次程序执行前，file.txt 是一个已经存在的空文件。

**Part A.** (1 分) 此时，X 处语句和 Y 处语句都是 count++;，Z 处语句是空语句。Alice 测试该代码，发现有时 file.txt 中没有任何输出！请解释原因。（提示：考虑 28 行语句 fork 以后，下一次被调度的进程，并从这个角度回答本题。不需要给出解决方案）

**Part B.** (6 分) Bob 根据 Alice 的反馈，在某两行之间加了若干代码，修复了 Part A 的问题。当 X 处代码和 Y 处代码都是 count++;、Z 处为空时，Bob 期望 file.txt 中的输出是：

+--+-----+--+-----+--+-----+--+-----

可 Alice 测评 Bob 的程序的时候，却发现有时 Bob 的程序在 file.txt 中的输出是：

+--+-----+--+-----

而与此同时，终端上出现了如下的输出：

+

Bob 找不到自己的代码的 BUG，只好向 Alice 求助。Alice 帮他做了如下分析：

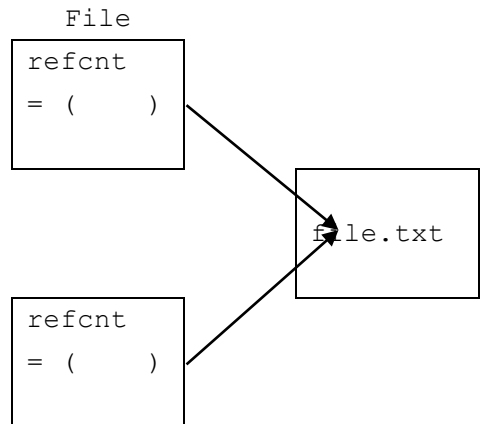
分析 1. 当程序第一次在终端上输出+的瞬间，请完成下表。要求：

- (1) 在“描述符表”一栏中，用“√”勾选该进程当前 fd1 的值。
- (2) 在“打开文件表”一栏中，填写该项的 refcnt（即，被引用多少次）。如果某一项不存在，请在括号中写“0”（并忽略其指向 v-node 表的箭头）。
- (3) 画出“描述符表”到“打开文件表”的表项指向关系。不需要画关于标准输入/标准输出/标准错误的箭头。评分时不对箭头评分，**请务必保证前两步的解答与箭头的连接情况匹配。**

描述符表	打开文件表	v-node 表
Descriptor	Open	V-node

父进程 Parent	(    ) 0
	(    ) 1
	(    ) 2
	(    ) 3

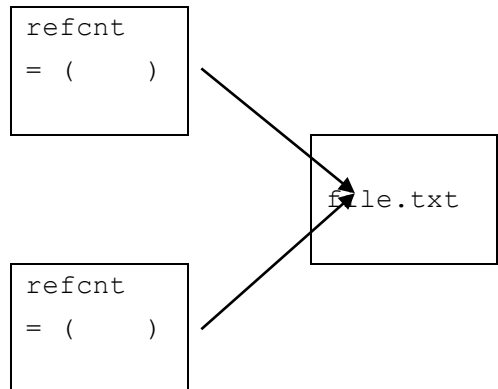
子进程 Child	(    ) 0
	(    ) 1
	(    ) 2
	(    ) 3



分析 2. 当程序第一次在 **file.txt** 中输出+的瞬间，仿照上题要求完成下表：

父进程 Parent	(    ) 0
	(    ) 1
	(    ) 2
	(    ) 3

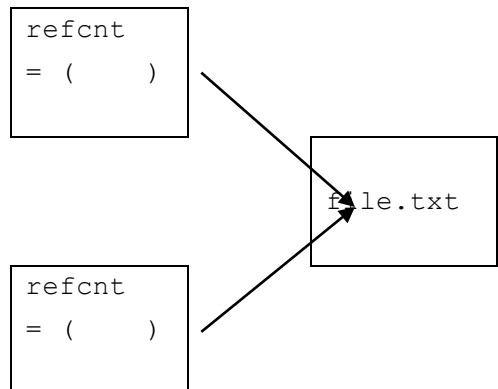
子进程 Child	(    ) 0
	(    ) 1
	(    ) 2
	(    ) 3



分析 3. 如果要产生 Bob 预期的输出，三级表的关系应当是什么？仿照上题要求完成下表：

父进程 Parent	(    ) 0
	(    ) 1
	(    ) 2
	(    ) 3

子进程 Child	(    ) 0
	(    ) 1
	(    ) 2
	(    ) 3



**Part C.** (2 分) Bob 很高兴，他知道 Part B 的代码是怎么错的了！不过 Alice 仍然想考考 Bob。对于 Part B 的错误代码，如果终端上输出的是+++，那么

--

X 处填写为: `count += 2;`  
Y 处填写为: `count += 2;`  
Z 处填写为:

--



得分

第五题（12 分）

1. （2 分，每空一分，考察多级页表和单级页表的设计思想的理解）

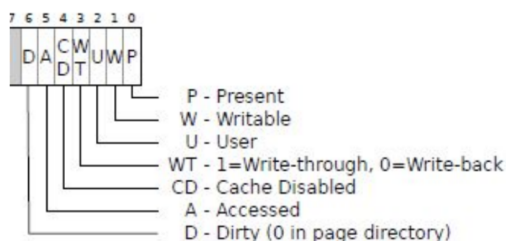
在进行地址翻译的过程中，操作系统需要借助页表(Page Table)的帮助。考虑一个 32 位的系统，页大小是 4KB，页表项(Page Table Entry)大小是 4 字节(Byte)，如果不使用多级页表，常驻内存的页表一共需要\_\_\_\_\_页。

考虑下图已经显示的物理内存分配情况，在二级页表的情况下，已经显示的区域 的页表需要占据\_\_\_\_\_页。

VP0	已分配页
...	
VP1023	
VP1024	
...	
VP2047	
Gap	未分配页
1023 unallocated pages	
VP10239	已分配页
VP10240	
...	
VP11263	

2. （6 分，每错一空扣一分，扣完为止。考察地址翻译过程）

IA32 体系采用小端法和二级页表。其中两级页表大小相同，页大小均为 4KB，结构也相同。TLB 采用直接映射。TLB 和页表每一项的后 7 位含义如下图所示。为简便起见，假设 TLB 和页表每一项的后 8~12 位都是 0 且不会被改变。注意后 7 位值为“27”则表示可读写。



当系统运行到某一时刻时，TLB 内容如下：

索引	TLB 标记	内容	有效位
0	0x04013	0x3312D027	1
1	0x01000	0x24833020	0
2	0x005AE	0x00055004	1
3	0x00402	0x24AEE020	0
4	0x0AA00	0x0005505C	0
5	0x0000A	0x29DEE000	1
6	0x1AE82	0x00A23027	1
7	0x28DFC	0x00023000	0

一级页表的基地址为 0x0C23B00，物理内存中的部分内容如下：

地址	内容	地址	内容	地址	内容	地址	内容
00023000	E0	00023001	BE	00023002	EF	00023003	BE
00023120	83	00023121	C8	00023122	FD	00023123	12
00023200	23	00023201	FD	00023202	BC	00023203	DE
00023320	33	00023321	29	00023322	E5	00023323	D2
0005545C	97	0005545D	C2	0005545E	7B	0005545F	45
00055464	97	00055465	D2	00055466	7B	00055467	45
0C23B020	27	0C23B021	EB	0C23B022	AE	0C23B023	24
0C23B040	27	0C23B041	40	0C23B042	DE	0C23B043	29
0C23B080	05	0C23B081	5D	0C23B082	05	0C23B083	00
2314D200	23	2314D201	12	2314D202	DC	2314D203	0F
2314D220	A9	2314D221	45	2314D222	13	2314D223	D2

29DE404C	27	29DE404D	42	29DE404E	BA	29DE404F	00
29DE4400	D0	29DE4401	5C	29DE4402	B4	29DE4403	2A

此刻，系统先后试图对两个已经缓存在 cache 中的内存地址进行写操作，请分析**完成写之后**系统的状态（写的地址和上面的内存地址无交集），完成下面的填空。  
若不需要某次访问或者缺少所需信息，请填“\”。

第一次向地址 0xD7416560 写入内容，TLB 索引为：\_\_\_\_\_，完成写之后该项 TLB 内容为：\_\_\_\_\_，

二级页表页表项地址为：\_\_\_\_\_，物理地址为：\_\_\_\_\_。

第二次向地址 0x0401369B 写入内容，

TLB 索引为：\_\_\_\_\_，完成写之后该项 TLB 内容为：\_\_\_\_\_

二级页表页表项地址为：\_\_\_\_\_，物理地址为：\_\_\_\_\_。

### 3. （2 分，考察对于虚拟内存独立地址空间的理解）

本学期的 fork bomb 作业中，大家曾用 fork 逼近系统的进程数量上限。下面有一个类似的程序，请仔细阅读程序并填空。

```
#include <stdio.h>
#include <sys/wait.h>
#include <unistd.h>
#define N 4
int main() {
    volatile int pid, cnt = 1;
    for (int i = 0; i < N; i++) {
        if ((pid = fork()) > 0) {
            cnt++;
        }
        while (wait(NULL) > 0);
        return 0;
    }
}
```

整个过程中，变量 cnt 最大值是\_\_\_\_\_。假设所有的数据都已经存在于内存中，pid 和 cnt 在同一个物理页中。从第一个进程开始执行 for 语句开始，此过程对于 cnt 的操作至少会导致页表中\_\_\_\_\_次虚拟页对应的物理页被修改。

得分

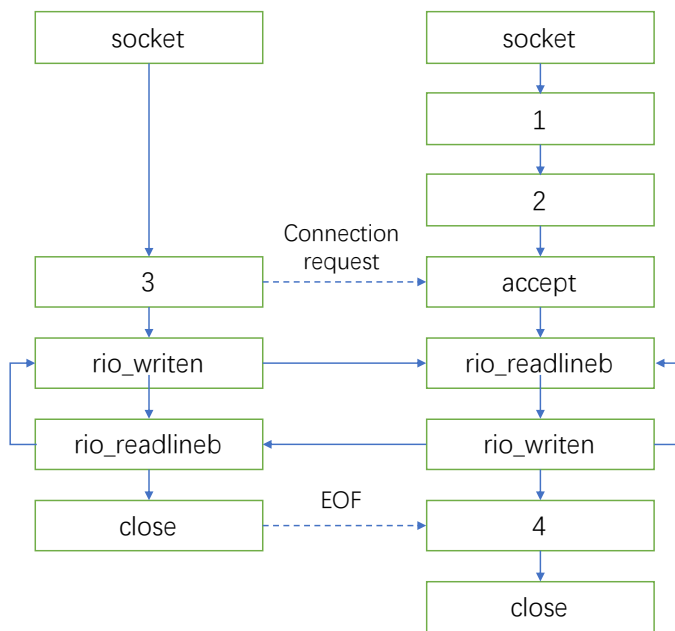
第六题（10 分）

1. 请在以下的表格中填入相应内容，每个空格仅需填一项。其中第 1 列可填选项包括：网络层、传输层、应用层；第 2 列可填选项 IP、UDP、TCP、HTTP；第 3 列可填选项包括：是、否。

（说明：面向连接的协议保障数据按照发送时的顺序被接收）

协议层次	协议名称	是面向连接的吗？
网络层		
	HTTP	
	TCP	

2. 下图描述了客户端与服务器套接字连接和通信的过程，请在空格处补充相关步骤。



得分

第七题（10 分）

给定如下程序：

```
#include <stdio.h>
#include <pthread.h>
int i = 0;
int j = 0;
void *do_stuff1(void * arg __attribute__((unused))) {
    int a;
    for (a = 0; a < 1000; a++)
        {i++; j++;}
    return NULL;
}
void *do_stuff2(void * arg __attribute__((unused))) {
    int a;
    for (a = 0; a < 1000; a++)
        {j++; i++;}
    return NULL;
}
int main() {
    pthread_t tid1, tid2;
    pthread_create(&tid1, NULL, do_stuff1, NULL);
    pthread_create(&tid2, NULL, do_stuff2, NULL);
    pthread_join(tid1, NULL);
    pthread_join(tid2, NULL);
    printf("%d, %d\n", i, j);
    return 0;
}
```

- （2 分）用以下元素的编号写出 i++编译后的汇编代码。  
 (a) mov    (b) add    (c) 0x601040    (d) %eax    (e) \$0x1  
 比如，回答(a) (e) (d)表示 mov \$0x1, %eax

2. (4 分) 请回答该程序是否有可能输出如下结果, 并简述原因。

- (a) 2000, 2000
- (b) 1500, 1500
- (c) 1000, 1000
- (d) 2, 2

3. (2 分) 卜廷江同学在学习了信号量之后, 决定要让程序能稳定输出 2000, 2000, 于是对程序进行了如下改写。

```
1. #include <stdio.h>
2. #include <pthread.h>
3. int i = 0;
4. int j = 0;
5. sem_t si;
6. sem_t sj;
7. void *do_stuff1(void * arg __attribute__((unused))) {
8.     int a;
9.     for (a = 0; a < 1000; a++) {
10. P(&si);
11. P(&sj);
12. i++;
13. j++;
14. V(&si);
15. V(&sj);
16. }
17. return NULL;
18. }
19. void *do_stuff2(void * arg __attribute__((unused))) {
20. int a;
21. for (a = 0; a < 1000; a++) {
22. P(&sj);
23. P(&si);
24. j++;
25. i++;
26. V(&si);
27. V(&sj);
28. }
```



```
29. return NULL;
30. }
31. int main() {
32. pthread_t tid1, tid2;
33. Sem_init(&si, 0, 1);
34. Sem_init(&sj, 0, 1);
35. pthread_create(&tid1, NULL, do_stuff1, NULL);
36. pthread_create(&tid2, NULL, do_stuff2, NULL);
37. pthread_join(tid1, NULL);
38. pthread_join(tid2, NULL);
39. printf("%d\n", i);
40. return 0;
41. }
```

请问卜廷江同学的程序有什么潜在问题？为什么？

4. （2 分）如果对卜廷江同学的程序改动一处数字来消除上述问题，同时仍然保证输出结果稳定为 2000, 2000，应该如何改动？

回答：将\_\_\_\_\_（填行号）行的\_\_\_\_\_（填数字）改为\_\_\_\_\_（填数字）。

得分

#### 第八题（10 分，每空 1 分）

题目假设：软硬件系统为 64 位 Linux 操作系统，页大小为 4KB；栈的分配空间最大为 8MB；题目中的 echoserveri/echoserverp/echoservert 分别为原书中的迭代 echo 服务器、基于进程的并发 echo 服务器、基于线程的并发 echo 服务器；当客户端连入服务器时，假设客户端只进行了连接，没有进行后续的读写操作。

启动 echoserveri，查看内存占用情况如下：（输出结果中删除了部分没有用到的列）

```
linux$ pmap -X `pidof echoserveri`
18859:  ./echoserveri 15213
Address Perm  Offset Device Size  Rss Pss Anonymous Mapping
00400000 r-xp 00000000 fc:02 20 20 20 0 echoserveri
00604000 r--p 00004000 fc:02 4 4 4 4 echoserveri
00605000 rw-p 00005000 fc:02 4 4 4 4 echoserveri
00a9a000 rw-p 00000000 00:00 132 8 8 8 (1)
7f56da7af000 r-xp 00000000 fc:00 1792 1220 11 0 libc-2.23.so
7f56da96f000 ---p 001c0000 fc:00 2048 0 0 0 0 libc-2.23.so
7f56dab6f000 r--p 001c0000 fc:00 16 16 16 16 libc-2.23.so
7f56dab73000 rw-p 001c4000 fc:00 8 8 8 8 libc-2.23.so
7f56dab75000 rw-p 00000000 00:00 16 16 16 16
7f56dab79000 r-xp 00000000 fc:00 96 92 1 0 libpthread-2.23.so
7f56dab91000 ---p 00018000 fc:00 2044 0 0 0 0 libpthread-2.23.so
7f56dad90000 r--p 00017000 fc:00 4 4 4 4 libpthread-2.23.so
7f56dad91000 rw-p 00018000 fc:00 4 4 4 4 libpthread-2.23.so
7f56dad92000 rw-p 00000000 00:00 16 4 4 4 4
7f56dad96000 r-xp 00000000 fc:00 152 152 1 0 (2)
7f56dafac000 rw-p 00000000 00:00 16 16 16 16
7f56dafbb000 r--p 00025000 fc:00 4 4 4 4 (2)
7f56dafbc000 rw-p 00026000 fc:00 4 4 4 4 (2)
7f56dafbd000 rw-p 00000000 00:00 4 4 4 4
7ffd382f8000 (3) 00000000 (4) 132 24 24 24 [stack]
7ffd38364000 r--p 00000000 00:00 12 0 0 0 0 [vvar]
```

```

7ffd38367000 r-xp 00000000 00:00 8 4 0 0 [vdso]
fffffffffff6000000 r-xp 00000000 00:00 4 0 0 0 [vsyscall]
=====
6540 1608 153 120 KB

```

填空：

1、 \_\_\_\_\_

2、 \_\_\_\_\_

3、 \_\_\_\_\_

4、 \_\_\_\_\_

在上面的表格中，Size 一列指的是 **VSS (Virtual Set Size)**，表示一个进程可访问的总的地址空间的大小 (the total accessible address space of a process)；**RSS (Resident Set Size)** 表示一个进程实际驻留在 RAM 中的空间大小 (the total memory actually held in RAM for a process)，其中包括了该进程所用到的所有共享空间(如共享库或私有 COW 空间)；**PSS (Proportional Set Size)** 与 RSS 的区别在于共享空间的尺寸，当某个共享空间为 30 页时，如果有三个进程共享该空间，则每个进程的该共享空间的 PSS 仅为 10 页，而每个进程的该共享空间的 RSS 仍为 30 页。根据上述定义，可以做出如下判断：VSS >= RSS >= PSS。

可以根据 echoserveri 的相关数值，对 echoserverp 和 echoservert 的运行情况进行估计：（单选题）

5、使用 echoserverp 作为服务器端，当十个客户端连入 echo 服务器时，总的 VSS 值可能为：

- A. 约 80MB
- B. 约 16MB
- C. 约 6MB
- D. 约 2MB

6、使用 echoserverp 作为服务器端，当十个客户端连入 echo 服务器时，总的 RSS 值可能为：

- A. 约 80MB
- B. 约 16MB

- C. 约 6MB
- D. 约 2MB

7、使用 echoserverp 作为服务器端，当十个客户端连入 echo 服务器时，总的 PSS 值可能为：

- A. 约 16MB
- B. 约 6MB
- C. 约 2MB
- D. 约 500KB

8、使用 echoservert 作为服务器端，当十个客户端连入 echo 服务器时，总的 VSS 值可能为：

- A. 约 80MB
- B. 约 16MB
- C. 约 6MB
- D. 约 2MB

9、使用 echoservert 作为服务器端，当十个客户端连入 echo 服务器时，总的 RSS 值可能为：

- A. 约 80MB
- B. 约 16MB
- C. 约 6MB
- D. 约 2MB

10、使用 echoservert 作为服务器端，当十个客户端连入 echo 服务器时，总的 PSS 值可能为：

- A. 比 echoserverp 作为服务器时的总 PSS 值大
- B. 比 echoserverp 作为服务器时的总 PSS 值小