北京大学信息科学技术学院试题

考试科目：_____ 姓名：_____ 学号：_____

考试时间：____年___月___日 任课教师：_____ 小班号：____

题号	一	二	三	四	五	六	**总分**
分数							
阅卷人							

北京大学考场纪律

1、考生要按规定的考试时间提前 5 分钟进入考场，隔位就座或按照监考人员的安排就座，将学生证放在桌面。无学生证者不能参加考试；迟到超过 15 分钟不得入场；与考试无关人员不得进入考场。如考试允许提前交卷，考生在考试开始 30 分钟后可交卷离场；未交卷擅自离开考场，不得重新进入考场继续答卷；交卷后应离开考场，不得在考场内逗留或在考场附近高声喧哗。

2、除非主考教师另有规定，学生只能携带必要的文具参加考试，其它所有物品（包括空白纸张、手机和智能手表等电子设备）不得带入座位；已经带入考场的手机和智能手表等电子设备必须关机，并与其他物品一起集中放在监考人员指定位置，不得随身携带或带入座位及旁边。

3、考试使用的试题、答卷、草稿纸由监考人员统一发放和收回，考生不得带出考场。考生在规定时间前答完试卷，应举手示意请监考人员收卷后方可离开；答题时间结束监考人员宣布收卷时，考生应立即停止答卷，在座位上等待监考人员收卷清点无误后，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。不准旁窥、交头接耳、打暗号或做手势，不准携带与考试课程内容相关的材料，不准携带具有发送、接收信息功能或存储有与考试课程内容相关材料的电子设备（如手机、智能手表、非教师允许的计算器等），不准抄袭或协助他人抄袭试题答案或者与考试课程内容相关的资料，不准窃取、索要、强拿、传、接或者交换试卷、答卷、草稿纸或其他物品，不准代替他人或让他人代替自己参加考试，等等。凡违反考试纪律或作弊者，按《北京大学本科考试工作与学习纪律管理规定》给予相应处分。

5、考生须确认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。

诚信宣言：

我承诺，严格遵守校规校纪，诚信考试！

考生签名：_____

得分

第一题．单选题，请结合教材的有关知识回答问题**(每题 2 分，共 40 分)**

注：选择题的回答必须填写在下表中，写在题目后面或其他地方，视为无效。

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．现定义了如下变量：

```
int x, y, z;
float fx, fy, fz;
```

则以下表达式不恒为真的是：

（其中 fx,fy 均不为无穷或非数，NaN 和 int 类型溢出按照模运算处理）

```
A.x * y == y * x
B.(x + y) * z == x * z + y * z
C.fx + fy == fy + fx
D.(fx + fy) * fz == fx * fz + fy * fz
```

2．下列选项中是合法 x86-64 汇编指令的是：

```
A. movq %rax, $1
B. xorq %rsp, %rsp
C. movq (%rax), (%rbx)
D. movq (%rax, %rbx, 3), %rcx
```

3．下列关于 RISC 和 CISC 的描述中，正确的是：

A.在 RISC 指令集的发展过程中，其指令数量始终少于 100 条。

B.CISC 指令集中的指令比 RISC 指令集更复杂，因此其指令长度总是比 RISC 指令集中的指令要长。

C.早期的RISC指令集没有条件码,对条件检测来说,要用明确的测试指令，这些指令会将测试结果放在一个普通的寄存器中。

D.RISC 指令集中的所有过程都需要进行内存引用。

4．在 Y86-64 PIPE 处理器中（不考虑数据前递），以下哪个指令序列会造成数据冒险？

A.

```
irmovq $10, %rdx
addq %rdx, %rax
```

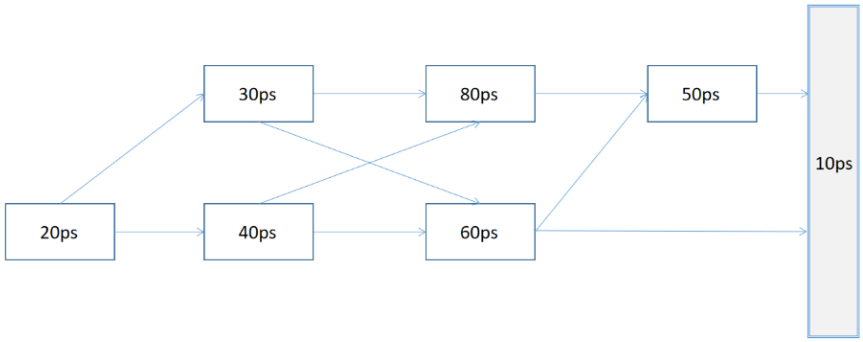
B.

```
irmovq $10, %rdx
nop
addq %rdx, %rax
```

C.
```
irmovq $10, %rdx
nop
nop
addq %rdx, %rax
```

D. 以上三个选项都会引发数据冒险

5. 根据下图所示的流水线结构，判断错误的选项是：



A.当前流水线的延迟为(20ps+40ps+80ps+50ps+10ps)=200ps，吞吐率为5GIPS。
B.可以通过插入寄存器的方式来使这个流水线变为二级流水线。
C.倘若只能插入两个寄存器(延迟均为10ps)，那么该流水线处理每条指令的平均时间最多减少20ps （假设每条指令平均使用一个周期的时间)。
D.流水线的级数越深，处理器就一定能获得越好的性能。

6. 以下关于程序设计优化的说法，错误的是：
A.循环展开有助于进一步变换代码，减少关键路径上的操作数量。
B.分离多个累积变量计算可以提高并行性。
C.大多数编译器会改变浮点数的合并运算（如加法和乘法）顺序以提高程序性能。
D.循环展开的程度增加并不一定能改善程序运行效率，反而会变差。

7. 以下关于存储设备与存储器层次结构的说法，正确的是：
A.一个有5个盘片，每个面10000条磁道，每条磁道平均有400个扇区，每个扇区有512个字节的硬盘容量为20.48GB。
B.SRAM的存取速度快，但是断电后数据会丢失，DRAM的存取速度较慢，但断电后数据不会丢失。
C.在存储器层次结构中，上一层存储的信息一定是下一层存储的信息的一个子集。
D.基于缓存的存储器层次结构只利用了程序的时间局部性，没有利用程序的空间局部性，因为缓存不会存储我们没访问过的元素。

8．布局是数字芯片设计自动化流程的必要步骤，下面哪一项不是布局算法需要优化的目标？

A．互连线长（Wirelength）

B．可布线性（Routability）

C．时序（Timing）

D．逻辑深度（Logic depth）

9．以下关于 Linux 系统上处理可执行文件的工具的说法错误的是：

A 使用 objdump 反汇编 .text 节的机器码。

B 使用 readelf 读取文件的节头部表(section header)和程序头部表(program header)。

C 使用 ls 查询文件是文本文件还是二进制文件。

D 使用 gdb 加载可执行文件、设置断点，然后单步调试运行。

10． 对如下两个 C 程序,用 gcc 生成对应的.o 模块,链接在一起得到 a.out 可执行程序。则下列说法正确的是：

```
// main.c
#include <stdio.h>
static int a;
int main() {
  int *func();
  printf("%ld\n", func() - &a);
  return 0;
}
```

```
// util.c
int a = 0;
int *func() {
  return &a;
}
```

A 在 main.o 中，符号 a 位于.COMMON 伪节。

B 在 util.o 中，符号 a 位于.COMMON 伪节。

C 无论怎样链接和运行 a.out，输出的结果都一样，但必不为 0。

D 以上说法都不正确。

11． 在标准 Linux 系统中，以下程序可以使用 gcc dl.c -ldl 编译生成 a.out 可执行文件并正常运行。如果缺少了 -ldl 选项，链接时会报错 undefined reference to 'dlopen'等信息。基于对动态链接的正确理解，请分析出以下说法中错误的一项是：

（libc.so 和 libdl.so 在实际系统中会带上版本号，路径名一般也更复杂。在本题中认为此处的 libdl.so 和 libc.so 是二进制文件而非符号链接。）

```
// dl.c
#include <dlfcn.h>

const char *path = "/lib/libc.so";
int (*printf)(const char *x);

int main() {
  // 加载共享库
  void *handle = dlopen(path, RTLD_NOW);
```

```
  // 解析符号 "printf" 并返回地址
  printf = dlsym(handle, "printf");
  // 调用
  printf("2022 is just around the corner.\n");
  // 关闭共享库
  dlclose(handle);
}
```

A 该机器上 libdl.so 模块中包含符号名为 dlopen 的动态链接符号表条目。

B 在 a.out 文件中包含 printf 的 PLT 条目和相应的 GOT 条目。

C 在 a.out 文件中包含 dlopen 的 PLT 条目和相应的 GOT 条目。

D 如果使用 gcc -ldl dl.c 编译程序，会在链接时发生错误。

12. 关于进程和异常控制流，以下说法正确的是：

A. 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。

B 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。

C 从内核态转换到用户态有多种方法，例如设置程序状态字；从用户态转换到内核态的唯一途径是通过中断/异常/陷入机制。

D 中断一定是异步发生的，陷阱可能是同步发生的，也可能是异步发生的。

13. 下列关于系统 I/O 的说法中，正确的是：

A. Linux shell 创建的每个进程开始时都有三个打开的文件：标准输入（文件描述符为 0）、标准输出（文件描述符为 1）、标准错误（文件描述符为 2），这使得程序始终不能使用保留的描述符 0、1、2 读写其他文件。

B. Unix I/O 的 read/write 函数是异步信号安全的，故可以在信号处理函数中使用。

C. RIO 函数包的健壮性保证了对于同一个文件描述符，任意顺序调用 RIO 包中的任意函数不会造成问题。

D. 使用 int fd1 = open(“ICS.txt”, O_RDWR); 打开 ICS.txt 文件后，再用 int fd2 = open(“ICS.txt”, O_RDWR); 再次打开文件，会使得 fd1 对应的打开文件表中的引用计数 refcnt 加一。

14. 考虑以下代码，假设 ICS.txt 中的初始内容为"ICS!!!ics!!!"。

```
int fd = open("ICS.txt", O_RDWR | O_CREAT | O_TRUNC,
S_IRUSR | S_IWUSR);
for (int i = 0; i < 2; ++i){
    int fd1 = open("ICS.txt", O_RDWR | O_APPEND);
    int fd2 = open("ICS.txt", O_RDWR);
    write(fd2, "!!!!!!", 6);
    write(fd1, "ICS", 3);
    write(fd, "ics", 3);
}
```

假设所有系统调用均成功，则这段代码执行结束后，ICS.txt 的内容为：

A. ICSics

B. !!!icsICS

C. !!!icsics!!!ICSICS

D. !!!icsICSICS

15. 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

16. 阅读下列代码。（已知文件“input.txt”中的内容为“12”，头文件没有列出）

```
void *Mmap(void *addr, size_t length, int prot, int flags,
           int fd, off_t offset);

int main(){
    int status;
    int fd = Open("./input.txt", O_RDWR);
    char* bufp = Mmap(NULL, 2, PROT_READ | PROT_WRITE ,
                      MAP_PRIVATE, fd, 0);
    if (Fork()>0){
       while(waitpid(-1,&status,0)>0);
       *(bufp+1) = '1';
       Write(1, bufp, 2); // 1: stdout
       bufp = Mmap(NULL, 2, PROT_READ, MAP_PRIVATE, fd, 0);
       Write(1, bufp, 2);
    }
    else{
        *bufp = '2';
        Write(1, bufp, 2);
    }
    close(fd);
    return 0;
}
```

在 shell 中运行该程序，正常运行时的终端输出应为：

A. 221112　　B. 222121　　C. 222112　　D. 221111

17. 以下符合对 TCP 协议描述的是

①可靠传输　②在主机间传输　③可以基于 IP 协议

④全双工　⑤可以基于 DNS

A. ①②④　B. ③④⑤　C. ①③④　D. ②④

18. 对于建立客户端与服务器的 TCP 连接，以下关于套接字接口，正确的是：

A. 对于 TCP 连接而言，其可以由连接双方的套接字对四元组（双方的 IP 地址和端口号）唯一确定。

B. connect 函数用于客户端建立与服务器的连接，其参数中的套接字地址结构需要指明本客户端使用的端口号

C. socket 函数返回的套接字描述符，可以立即使用标准 Unix I/O 函数进行读写

D. bind 函数用于将用于客户端的主动套接字转化为用于服务器的监听套接字

19. 下列有关 Web 服务错误的是：

A. Web 服务使用客户端-服务器模型，使用了 HTTP 协议，可以传输文本，HTML 页面，二进制文件等多种内容

B. HTTP 响应会返回状态码，它指示了对响应的处理状态

C. Web 服务使用 URL 来标明资源，这提供了一层抽象，使得客户端仿佛在访问远端的文件目录，而服务器处理了 URL 资源和具体文件/动态内容的映射关系

D. 多个域名可以映射到同一个 IP 地址，但一个域名不可以映射到多个 IP 地址

20. 下列关于 C 语言中进程模型和线程模型的说法中，错误的是：

A.每个线程都有它自己独立的线程上下文，包括线程 ID、程序计数器、条件码、通用目的寄存器值等

B.每个线程都有自己独立的线程栈，任何线程都不能访问其他对等线程的栈空间

C.不同进程之间的虚拟地址空间是独立的，但同一个进程的不同线程共享同一个虚拟地址空间

D.一个线程的上下文比一个进程的上下文小得多，因此线程上下文切换要比进程上下文切换快得多

得分

第二题. 请结合教材第六章"存储器层次结构"的有关知识回答问题**(10 分)**

1.(2 分)高速缓存(cache)的**先进先出**替换策略(FIFO)指的是在发生缓存不命中时,最先进入高速缓存的行(cache line)将最先被替换出.考虑两种遵循先进先出替换策略且块大小(block size)相同的**全相联**高速缓存 C1 和 C2.其中 C1 是**3 路**全相联的,C2 是**4 路**全相联的.假设初始情况下 C1,C2 所有行的**有效位都为 0**.则让它们分别连续访问标记(tag)为 0, 1, 2, 3, 0, 1, 4, 0, 1, 2, 3, 4 的行之后, C1 发生的缓存不命中次数**(1)**_____C2 发生的缓存不命中次数**(填">""="或"<")**.

2. 已知某单核处理器有 L1,L2,L3 三级高速缓存且遵循先进先出替换策略,你希望通过触发行驱逐(cache line eviction)的方法测量该处理器的 L1 数据缓存(L1 d-cache)的相联度,实验前已知 L1 d-cache 的容量在 16KiB 到 64KiB 之间(含 16KiB 和 64KiB)且字节数为 2 的幂,且已知高速缓存块的字节数小于 1KiB.你的实验步骤如下:

Step #1: 开辟一块足够大(如 8MiB)的内存空间.

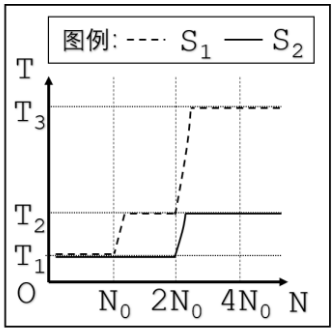
Step #2: 逐步调整访问的**步长 S**(S 依次取 1KiB, 2KiB, 4KiB, …, 32KiB,64KiB)和访问的**元素数量 N**(N 依次取 1, 2, 3, …, 127, 128).对于给定的 S 和 N,连续两次从所开辟内存的起始位置开始,以 S 为步长顺序访问 N 个内存地址.记录第二次访问时平均一个内存地址的访问时间 T.

Step #3: 多次实验取平均值以减小误差.作图并分析实验结果.

这里是给定 S 和 N 后访问内存的一个例子:当 S=64KiB,N=6 时,先按顺序访问一遍所开辟内存空间的第 0B, 64KiB, 128KiB, 192KiB, 256KiB, 320KiB 处的一个字节的值.清空寄存器,接着计时并第二遍按顺序访问所开辟内存空间的第 0B, 64KiB, 128KiB, 192KiB, 256KiB, 320KiB 处的一个字节的值.访问完成后立即停止计时,并记录 T = (结束计时的时刻 - 开始计时的时刻) ÷ 6.0.

(a) (2 分)Step #2 中,给定 S 和 N 后第一遍访问 N 个内存地址的过程被称为**暖身**(warm up),这是为了避免**(2)**_____不命中带来的影响**(填"冷/强制性""冲突"或"容量")**.

(b) (4 分)假设访问内存的过程都在高速缓存中顺次进行,且**不考虑预取**(prefetch)机制的影响.你选取了步长为 S_1, S_2 时的部分实验结果如图所示(此图为示意图).你注意到访存速度越快,则访问单个元素的时间越短.因此从图中可以看出 T_3 最接近**(3)**_____**(填"L1""L2"或"L3")**级缓存的访存时间,S_1**(4)**_____S_2**(填">""="或"<")**.



(c) (2 分)在**(b)**的条件下, 进一步推理可知该处理器的 L1 d-cache 的容量(capacity)为**(5)**_____**(用代数式表示,S_1, S_2, N_0 为已知量,下同.注意计算容量时不考虑有效位和标记位)**,相联度为**(6)**_____.

得分

第三题．请结合教材第七章**"链接"**的有关知识回答问题**(10 分)**

有以下三个 c 文件 hd.h f1.c f2.c。使用 gcc -c f1.c f2.c; gcc f1.o f2.o 编译后得到可执行文件 a.out。回答以下问题。Part A 中涉及的符号所对应的变量已在代码中加粗。**本大题无需理解代码的含义。**

f1.c

```
#include "hd.h"
#include <stdio.h>
const int total = 1 << 30;
static int count = 0;
static Point pnt;
int iter;
int main() {
  for (iter = 0; iter < total; ++iter) {
    rand_point(&pnt);
    count += if_inside(&pnt);
  }
  printf("Integral on [0,1] is %lf.\n",
         1.0 * count / total);
}
```

hd.h

```
typedef struct {
  double x;
  double y;
} Point;
void rand_point(Point *);
int if_inside(Point *);
```

f2.c

```
#include "hd.h"
#include <stdlib.h>
#include <time.h>
void rand_point(Point *ptr) {
  static int seed = 0;
  if (!seed) {
    srand((unsigned)time(NULL));
    seed = 1;
  }
  ptr->x = 1.0 * rand() / RAND_MAX;
  ptr->y = 1.0 * rand() / RAND_MAX;
}
int if_inside(Point *p) {
  return 1 / (1 + p->x) >= p->y;
}
```

Part A. **(每个符号 1 分，共 5 分)** 请说明以下符号是否**在 a.out 的符号表**中。如果是，请进一步指出符号定义所在的节，可能的选择有.text、.data、.bss、.rodata、COM、UNDEF、ABS。

符号名	iter	pnt	Point	total	seed
在符号表中？**（填是/否）**					
定义所在节					

Part B. **(每空 1 分，共 3 分)** 使用 objdump -dx f1.o f2.o 看到如下几条代码。**这里可以将重定位类型 R_X86_64_PLT32 和 R_X86_64_PC32 同等看待。**

```
# objdump 重定位条目格式：
#           OFFSET: TYPE                VALUE
# e.g.          18: R_X86_64_PLT32      rand_point-0x4
# 所有数值均以十六进制表示
# f1.o
0000000000000000 <main>:
... # 省略无关代码
  17: e8 00 00 00 00.         callq  1c <main+0x1c>
          18: R_X86_64_PLT32       rand_point-0x4
  1c: 48 8d 3d 00 00 00 00    lea 0x0(%rip),%rdi
          1f: R_X86_64_PC32        .bss+0xc
  23: e8 00 00 00 00          callq  28 <main+0x28>
          24: R_X86_64_PLT32       if_inside-0x4
  28: 89 c2                   mov %eax,%edx
... # 省略无关代码
# f2.o
0000000000000000 <rand_point>:
... # 省略无关代码
0000000000000074 <if_inside>:
... # 省略无关代码
```

据此可以确定 <main+0x1f> 处的重定位条目是针对符号_____(**填写符号名，不要填写.bss 这个节名**)的重定位。同时该符号定义的位置在 f1.o 中相对于 .bss 节的偏移量是 0x_____。

现已知 a.out 文件中 <main+0x17> 行变成

```
11a1: e8 69 00 00 00          callq  <rand_point>
```

那么 a.out 中 <main+0x23> 行将变成

```
11ad: e8 ___________          callq  <if_inside>
```

Part C. **(每空 1 分，共 2 分)** 使用 execve 加载 a.out 并执行时，其中第一个被执行的语句默认是_____(单选)函数的开头。已知 gcc -e 可以修改该默认行为到一个程序指定的函数，据此推断该函数执行在_____态下(填 用户/内核)。

A. _init　　B.main　　C.__libc_start_main　　D._start

得分

第四题．请结合教材第八章**"异常控制流"**的有关知识回答问题**(10 分)**

PART A． Alice 想用两个进程来顺序输出奇数和偶数，请你帮她补全代码。她的做法是：父进程首先 fork 出两个子进程，之后子进程之间互相通信，顺序输出 1,2,…,N。请严格按照注释描述的功能填写代码，**假设兄弟进程的 pid 是相邻的**。

```
int N;
int nxt; //表示该子进程下一个需要输出的数
int pid_1 = 0; //第一次 fork 出的子进程 pid
int pid_2 = 0; //第二次 fork 出的子进程 pid
// All child process should do their work in handler 1
void handler1(int sig) {
    printf("%d\n", nxt);
    nxt += 2;
    if (nxt & 1) kill(getpid()+1, SIGUSR1);
    else {
        assert(pid_1 != 0);
        ________A________ // 通知兄弟进程
    }
    if (nxt > N) exit(0);  // 子进程完成输出后退出
}

int main(int argc, char* argv[]) {
    ________B________ //将 handler1 绑定到 SIGUSR1 上
    N = atoi(argv[1]);
    nxt = 1;
    if ((pid_1 = fork()) != 0) {
        ________C_______;  // 设置 nxt 的初值
        if ((pid_2 = fork()) != 0) {  // 该进程是父进程
            kill(pid_1, SIGUSR1);
            goto wait_til_end;  //跳转并等待子进程结束
        }
    }
    while (1) { sleep(1); }  // 子进程会在此循环直到输出完成

    wait_til_end:
    int status;
    while (_______D_______)  // 用 waitpid 等待子进程结束，注意
status
        assert(WIFEXITED(status));
    return 0;
}
```

（4分，每空1分）

A. _____

B. _____

C. _____

D. _____

PART B. 阅读如下 C 代码，回答问题

```
int counter = 0;
int pid = 0;
int N = 2;
void handler1(int sig) {
    counter++;
    printf("%d", counter); fflush(stdout);
    // Kill(pid, SIGUSR2);
}
void handler2(int sig) {
    printf("R"); fflush(stdout);
}
int main() {
    Signal(SIGUSR1, handler1);
    Signal(SIGUSR2, handler2);
    if ((pid = Fork()) == 0) { // child
        for (int i = 0; i < N; ++i) {
            printf("C"); fflush(stdout);
            Kill(Getppid(), SIGUSR1);
        }
    } else { // parent
        Wait(NULL);
    }
    return 0;
}
```

1.进程在何时检查待处理信号，并调用相应的signal handler处理信号？_____（1分）

A.随时　　B.用户态切换到内核态　　C.内核态切换到用户态

2.在两次检查并处理信号量的时间间隙中，如果进程接收到n个**相同**信号，那么进程实际会处理几个信号？_____（1分）

A.1　　B.1到n之间的随机数值　　C.n

3.如果N=2，所有可能的输出为：_____

（全部选对得2分，部分选对得1分，选错不得分）

A.CC　　B.CC1　　C.CC12　　D. C1C2

4.如果 N=2，并且取消hanlder1中的注释（第7行），所有**不可能**的输出为：_____

（全部选对得2分，部分选对得1分，选错不得分）

A. CC　　B.CC1　　C.CC1R2　　D.C1RC2　　E.C1CR2

得分

第五题．请结合教材第九章**"虚拟内存"**的有关知识回答问题**(15 分)**

IA32 体系采用**小端法**、32 位虚拟地址和两级页表。两级页表大小相同，页大小都是 4 KB = 2^{12} Byte，结构也相同。TLB 采用**直接映射**，4 位组索引。TLB 和页表每一项格式如图所示：

31 12	11 9	8	7	6	5	4	3	2	1	0
Address of 4KB page frame	Ignored	G	PAT	D	A	PCD	PWT	U/S	R/W	P

部分位的含义如下：

0 (P)：1 表示存在，0 表示不存在

1 (R/W)：1 表示可写，0 表示只读

2 (U/S)：1 表示内核模式，0 表示用户模式

当系统运行到某一时刻，TLB **有效位为 1** 的条目如下（未列出部分都是无效的）：

索引（十进制）	TLB 标记	内容
0	0x0400	0x0ec91313
3	0x02ff	0x5d2bac01
5	0xd551	0x019fa42d
11	0x55a6	0xfdd3c66b
13	0x5515	0xb591926b

一级页表基地址为 0x00e66000，物理内存中的部分内容如下（均为十六进制）：

地址	内容	地址	内容	地址	内容
00615000	21	00615001	2d	00615002	ee
00615003	c0	006154d0	ff	006154d1	a0
00e66001	a1	00e66002	a4	00e66003	67
00e66004	21	00e66005	57	00e66006	61
00e66d07	00	2167e000	42	2167e001	67
2167e002	9a	2167e003	7c	c0ee2000	6f
c0ee2001	d5	c0ee2002	7e	c0ee24d0	48
c0ee24d1	83	c0ee24d2	ec	c0ee2d21	11
c0ee2d22	6b	c0ee2d23	82	c0ee2d24	8a

1．将 cache 清空，访问一个在主存中的虚拟地址，TLB 命中，**没有**触发缺页异常，这一过程中，需要访问物理内存_____次（3 分）。具体来说，如果该地址为 y = 0xd5515213，y 地址所具有的**实质权限**是_____（多选题，选对才得分 2 分）。

①可写 ②只读 ③用户模式权限 ④内核模式权限

2．不考虑第一小问，将 cache 清空，访问一个在主存中的虚拟地址，TLB 不命中，**没有**触发缺页异常，这一过程中，需要访问物理内存_____次（3 分）。具体来说，如果该地址为 x = 0x004004d0，那么 x 对应的二级页表起始地址是_____（填写 16 进制，例如 0x00123000，2 分），x 地址上单字节的**内容**是_____（填写 16 进制，例如 0x00，1 分）。

3．考虑下面计算矩阵和向量乘法代码：

```
1 int *mat_vec_mul(int **A, int *x, int n)
2 {
3     int i, j;
4     int *y = (int *)malloc(n * sizeof(int));
5     for (i = 0; i < n; i++)
6         for (j = 0; j < n; j++)
7             y[i] += A[i][j] * x[j];
8     return y;
9 }
```

⑴ 在 64 位 Linux 机器中运行该代码，输入**同一组合法的参数**后，每次运行返回的向量的值都**不一样**，修复这一错误有一种简单的方法，将第____行改为_____。（第二空填写 C 代码，每空 1 分，共 2 分）

可能用到的函数：

```
void *memcpy(void *dest, const void *src, size_t n);
void *memset(void *s, int c, size_t n);
void *calloc(size_t nelem, size_t elemsize);
void *realloc(void *ptr, size_t size);
```

⑵ 在进行前一问的测试**之前**，还在 64 位 Linux 机器上进行过如下用户代码测试（输入 mat_vec_mul 的参数都**非零**）：

```
int *y = mat_vec_mul(A, x, n);
int z = y[0];
```

结果发生了段错误。通过 gdb 调试发现 mat_vec_mul 函数内并没有发生段错误，但是在初始化变量 z 时发生了段错误，后来发现是参数的输入有问题。写出出现这种错误的**充分必要条件**_____（1 分）

4．Double fault：Intel 处理器中有一种特殊的异常，被称为 double fault。此异常发生表明调用某个**故障（fault）**A 的处理程序后又触发了另一个故障 B。正常情况下，故障 B 会有相应异常处理程序来处理，因此两个故障 B 和 A 可以被顺序解决。但是如果处理器无法正常处理故障 B，或是处理了之后依然无法处理故障 A，就会产生 double fault，并终止（abort）。假设除了缺页异常处理程序外，其他异常处理程序**都不会产生新的故障**。如果在某次**缺页故障**时产生了 double fault，其原因可能是_____（不定项选择，都选对才得分，1 分）

① 运行缺页故障处理程序时， CPU 上的权限位是内核态，但所执行代码段 U/S=0

② 运行缺页故障处理程序时，CPU 接收到了键盘发送的 Ctrl + C 信号

③ 缺页故障处理程序没有加载到主存中

得分

第六题．请结合教材第十二章“**并发编程**”的有关知识回答问题**(15 分)**

“生产者-消费者”问题是并发编程中的经典问题。本题中，考虑如下场景：

a. 所有生产者和所有消费者**共享同一个 buffer**
b. 生产者、消费者各有 NUM_WORKERS 个（**大于一个**）
c. buffer 的容量为 BUF_SIZE，**初始情况下 buffer 为空**
d. 每个生产者向 buffer 中添加一个 item；若 buffer 满，则生产者等待 buffer 中有空槽时才能添加元素
e. 每个消费者从 buffer 中取走一个 item；若 buffer 空，则消费者等待 buffer 中有 item 时才能取走元素

1．阅读以下代码并回答问题（**代码阅读提示：主要关注 producer 和 consumer 两个函数**）

```
1. /* Producer-Consumer Problem (Solution 1) */
2.
3. #include "csapp.h"
4.
5. #define BUF_SIZE 3
6. #define NUM_WORKERS 50
7. #define MAX_SLEEP_SEC 10
8.
9. volatile
        static int items = 0; /* How many items are there in t
        he buffer */
10.
11.     static sem_t mutex; /* Mutual Exclusion */
12.     static sem_t empty; /* How many empty slots are there
        in the buffer */
13.     static sem_t full;  /* How many items are there in the
         buffer */
14.
15.     static void sync_var_init() {
16.         Sem_init(&mutex, 0, 1);
17.
18.         /* Initially, there is no item in the buffer */
19.         Sem_init(&empty, 0, BUF_SIZE);
20.         Sem_init(&full, 0, 0);
21.     }
22.
23.     static void *producer(void *num) {
24.         ①;
25.         ②;
26.
27.         /* Critical section begins */
28.         Sleep(rand() % MAX_SLEEP_SEC);
29.         items++;
30.         /* Critical section ends */
31.
32.         V(&mutex);
33.         V(&full);
```

```
34.
35.         return NULL;
36.     }
37.
38.     static void *consumer(void *num) {
39.         ③;
40.         ④;
41.
42.         /* Critical section begins */
43.         Sleep(rand() % MAX_SLEEP_SEC);
44.         items--;
45.         /* Critical section ends */
46.
47.         V(&mutex);
48.         V(&empty);
49.
50.         return NULL;
51.     }
52.
53.     int main() {
54.         sync_var_init();
55.
56.         pthread_t pid_producer[NUM_WORKERS];
57.         pthread_t pid_consumer[NUM_WORKERS];
58.
59.         for (int i = 0; i < NUM_WORKERS; i++) {
60.             Pthread_create(&pid_producer[i], NULL, produce
        r, (void *)i);
61.             Pthread_create(&pid_consumer[i], NULL, consume
        r, (void *)i);
62.         }
63.
64.         for (int i = 0; i < NUM_WORKERS; i++) {
65.             Pthread_join(pid_producer[i], NULL);
66.             Pthread_join(pid_consumer[i], NULL);
67.         }
68.     }
```

a) 补全代码（请从以下选项中选择，可重复选择，每个 1 分，共 4 分）

①_____（24 行）

②_____（25 行）

③_____（39 行）

④_____（40 行）

选项：

A. P(&mutex)

B. P(&empty)

C. P(&full)

b) 如果交换 24 行与 25 行（两个 P 操作），_____（单选，2 分）

A. 有可能死锁

B. 有可能饥饿

C. 既不会死锁，也不会饥饿

c) 交换 32、33 行（两个 V 操作）是否可能造成同步错误？____（2 分）
A. 可能
B. 不可能

d) rand 函数是不是线程安全的？____（1 分）
A. 是
B. 不是

28 行与 43 行对 rand 函数的使用是否会导致竞争？____（1 分）
A. 会
B. 不会

已知 rand 函数的实现如下
来源：
https://github.com/begriffs/libc/blob/master/stdlib.h
https://github.com/begriffs/libc/blob/master/stdlib.c

```
1. #define RAND_MAX 32767
2.
3. unsigned long _Randomseed = 1;
4.
5. int rand() {
6.     _Randomseed = _Randomseed * 1103515425 + 12345;
7.     return (unsigned int)(_Randomseed>>16) & RAND_MAX;
8. }
9.
10.void srand(unsigned int seed) {
11.    _Randomseed = seed;
12.}
```

2．考虑“生产者-消费者”问题的另一种解法（**代码阅读提示：12-69 行之外均与上一种解法相同**）

```
1. /* Producer-Consumer Problem (Solution 2) */
2.
3. #include "csapp.h"
4.
5. #define BUF_SIZE 3
6. #define NUM_WORKERS 50
7. #define MAX_SLEEP_SEC 10
8.
9. volatile
        static int items = 0; /* How many items are there in t
        he buffer */
10.
11.static sem_t mutex;                /* Mutual Exclusion */
12.static sem_t sem_waiting_producer; /* Wait for empty slots *
        /
13.static sem_t sem_waiting_consumer; /* Wait for available ite
        ms */
14.
15.volatile static int num_waiting_producer = 0;
16.volatile static int num_waiting_consumer = 0;
```

```
17.
18. static void sync_var_init() {
19.     Sem_init(&mutex, 0, 1);
20.
21.     Sem_init(&sem_waiting_producer, 0, ①);
22.     Sem_init(&sem_waiting_consumer, 0, ①);
23. }
24.
25. static void *producer(void *num) {
26.     P(&mutex);
27.     while (items == BUF_SIZE) {
28.         num_waiting_producer++;
29.         ②;
30.         ③;
31.         P(&mutex);
32.     }
33.
34.     /* Critical section begins */
35.     Sleep(rand() % MAX_SLEEP_SEC);
36.     items++;
37.     /* Critical section ends */
38.
39.     if (num_waiting_consumer > 0) {
40.         num_waiting_consumer--;
41.         V(&sem_waiting_consumer);
42.     }
43.     V(&mutex);
44.
45.     return NULL;
46. }
47.
48. static void *consumer(void *num) {
49.     P(&mutex);
50.     while (items == 0) {
51.         num_waiting_consumer++;
52.         ④;
53.         ⑤;
54.         P(&mutex);
55.     }
56.
57.     /* Critical section begins */
58.     Sleep(rand() % MAX_SLEEP_SEC);
59.     items--;
60.     /* Critical section ends */
61.
62.     if (num_waiting_producer > 0) {
63.         num_waiting_producer--;
64.         V(&sem_waiting_producer);
65.     }
66.     V(&mutex);
67.
68.     return NULL;
69. }
70.
71. int main() {
72.     sync_var_init();
73.
```

```
74.     pthread_t pid_producer[NUM_WORKERS];
75.     pthread_t pid_consumer[NUM_WORKERS];
76.
77.     for (int i = 0; i < NUM_WORKERS; i++) {
78.         Pthread_create(&pid_producer[i], NULL, producer, (vo
          id *)i);
79.         Pthread_create(&pid_consumer[i], NULL, consumer, (vo
          id *)i);
80.     }
81.
82.     for (int i = 0; i < NUM_WORKERS; i++) {
83.         Pthread_join(pid_producer[i], NULL);
84.         Pthread_join(pid_consumer[i], NULL);
85.     }
86. }
```

a) 补全代码（请从以下选项中选择，④⑤无需填写，每个 1 分，共 3 分）
①_____（21、22 行）
②_____（29 行）
③_____（30 行）
选项：
A. 0
B. 1
C. P(&sem_waiting_producer)
D. V(&mutex)

b) 如果 27 行和 50 行的 while 换成 if，是否可能造成同步错误？
____（2 分）
A. 可能
B. 不可能