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：2024 年 12 月 30 日 **小班号：**_____

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

装订线内

不要答题

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。

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．假想一类“半字节数”（4-bit）的数据类型，包括有符号整数（int4）以及浮点数（float4）。有符号整数（int4）采用补码表示，浮点数（float4）基于 IEEE 浮点格式，有 1 个符号位，2 个阶码位（k=2）和 1 个小数位（n=1）。有两种函数 f2b(f) 和 f2i(f)，代码实现如下所示：

```
int4 f2b(float4 f) {
  union {
    int4 d;
    float4 f;
  } temp;
  temp.f = f;
  return temp.d;
}
```

```
int4 f2i(float4 f) {
  return (int4) f;
}
```

f2b 函数返回 f 的二进制表示，如若 f 的二进制表示为 0100，则返回的 int4 值为 4。 f2i 函数返回 f 的值向零舍入的结果。我们执行如下函数，其行为是从 0000 到 1111 枚举非 NaN 及 ±INF 的所有 float4 类型能表示的浮点数值，并进行比较。

```
void fun() {
  for (float4 f from 4'b0000 to 4'b1111) {
    if (isnan(f) || isinf(f)) continue;
    if (f2b(f) == f2i(f)) printf("Success!\n");
  }
}
```

请问 f2b 和 f2i 相等时，f 的二进制表示是：

① 0000 ② 0001 ③ 1000 ④ 1101

A．①

B．①②④

C．①③

D．①④

2．下列关于处理器体系结构的说法，**正确**的是：

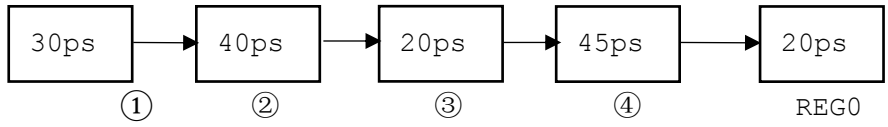
A．RISC 的设计理念是精简指令，为了减少指令的数量，它使用的指令长度可变。

B．在设计处理器时，硬件上复制逻辑块的成本要比软件中复制代码的成本低得多，而且在硬件系统中处理各种特殊情况也比用软件处理简单。

C．在处理器的流水线中，异常指令在到达写回阶段之前，不用让异常事件影响流水线中的指令流，只需要禁止后面的指令更新条件码和修改内存等产生永久影响的操作即可。

D．在处理器的流水线中，在译码阶段需要处理转发（Forwarding）。如果有相同目标的多个转发源，赋予不同优先级是十分重要的，来自越靠后流水线阶段的转发源的优先级越高。

3．如下图所示，现有 4 个组合逻辑单元①~④和一个寄存器 REG0，对应的延迟已在图中标出。现在需要插入 **2 个额外的**流水线寄存器 REG1 和 REG2（延迟均为 20ps），则改造后流水线的最大吞吐量为_____GIPS。



A．16.7　　B．12.5　　C．10　　D．8.33

4．下列关于存储设备的说法，**正确**的是：

A．随机访问存储器（Random-Access Memory，RAM）分为静态的和动态的，这两类也被统称为 SDRAM。

B．只读存储器（Read-Only Memory，ROM）只能被编程一次，也就是它们只能被写一次。只读存储器也被称为非易失性存储器。

C．固态硬盘（Solid State Disk, SSD）的随机写比读慢，因为擦除块需要较长的时间，如果写操作试图修改一个包含已有数据的页，那么这个块中所有带有有用数据的页都要被复制到一个新块中。

D．组相联高速缓存很容易产生“抖动”问题，因此我们引入了直接映射高速缓存。

5．针对如下代码，假定编译器不做任何编译优化，同时编译系统在预处理 a.c 和 b.c 时可以顺利找到 common.h。**运行 gcc a.c b.c 命令**，发现报错。已知报错信息中含有如下内容：

collect2: error: ld returned 1 exit status

```c
// a.c
#include <stdio.h>
#include "common.h"

int main() {
  scanf("%d\n", &x);
  funcB();
  printf("%d\n", x);
}
```

```c
// b.c
#include "common.h"

int funcB(void) {
  x++;
}
```

```c
// common.h

int x = 1;

int funcB(void);
```

则下列说法**正确**的是：

A． 在编译系统及二进制工具链对本题所涉源代码进行处理过程中，会产生名为 a.o、b.o 的中间临时文件，并将其链接到一起。

B．根据题目中的报错信息描述进行推断，报错信息是由编译系统及二进制工具链中的预处理器输出的。

C．报错原因是 a.c、b.c、common.h 其中之一含有语法错误。这个错误可以通过调整编译顺序来解决，即 **gcc b.c a.c**。

D．在编译系统及二进制工具链对本题所涉源代码进行处理过程中，所产生的中间目标文件的.text 节中，对于变量 x 的引用都需要在链接时重定位。

6．本题基于下面 c 语言文件编译生成的可重定位目标文件 main.o。该文件的生成过程在 x86-64 Linux 下完成，其中编译过程未加优化选项。已知**本题不涉及 COMMON 节，不涉及重定位相关节**。下面有关符号及其所在节说法**错误**的是：

```c
//main.c
extern int extern_var;

void global_used_func();

int init_var = 1;

int init_func() {
    return 1145;
}
void f() {
    static int x = 2;
    int local_var = 1;
    local_var++;
    x++;
}
int main() {
```

```
    extern_var++;
    global_used_func();
    return 0;
}
```

A. extern_var 和 global_used_func 在符号表中，且位于 UND 节；而 extern_unused_var **不会**出现在符号表中。
B. init_var 在符号表中，且位于.data 节。
C. init_func **不会**出现在符号表中，因为 main.c 没有调用它。
D. f 函数中的局部静态变量 x 以某种形式出现在符号表中，且位于.data 节；而 f 函数中的 local_var **不会**出现在符号表中，它存在栈上。

7. 下列陈述中，**错误**陈述的数量为：
①进程调度的实现在一定程度上依赖于定时器中断（timer interrupts）。
②execv()函数成功调用后不会返回。
③使用 fork()创建的子进程与父进程对内存的修改是共享的。
④在 X86-64 Linux 中，访存缺页异常属于故障。
⑤异常处理的返回行为包括：返回当前指令、返回下一条指令和终止(abort)。
⑥如果处理得当（如设置 SIGFPE 的信号处理函数），除以零引发的除法错误不会导致终止(abort)。
⑦单核处理器上，可以通过不断地上下文切换交替运行并发的程序，给用户呈现一种并发的程序在同时运行的感觉。
⑧进入信号处理函数时会创建新的进程。
A. 0 个　　B. 1 个　　C. 2 个　　D. 3 个

8. 在 Ubuntu Linux 24.04 环境中，关于系统级 IO，下列说法**正确**的是：
A.在未进行特殊设置和额外操作的情况下，某进程先调用了 fprintf()函数向某文件写入数据，且返回值恰为传入数据字节数。若向该进程发送 SIGKILL 信号终止进程，再打开该文件，一定能看到该进程写入的内容。
B.同一进程内的多个线程不加同步地调用 fprintf()**会**导致程序崩溃。
C.由于慢速系统调用 read()和 write()会潜在地阻塞进程一段较长时间，高级 I/O 函数使用缓冲区来减少对 read()和 write()的调用。
D.无论父进程的标准 I/O 缓冲区——例如 printf()函数的缓冲区是否为空，执行 fork()创建的子进程的缓冲区总是为空。

9．关于虚拟地址和物理地址的转换，下列说法**错误**的是：

A．在 x86-64 Linux 虚拟内存中，物理地址到虚拟地址的映射是通过页表实现的。

B．TLB 用于缓存虚拟地址到物理地址的映射，以加速地址转换。

C．某 x86-64 Linux 中页大小为 4KB，每个页表条目为 8 字节，每个页表占据一页（即 4KB），使用 48 位虚拟地址和 44 位物理地址。如果支持大页（2MB 页），可以减少 1 级页表。

D．Intel Core i7 体系结构中的 CR3 寄存器用于保存一级页表的虚拟地址。

10．关于 Linux 内存映射，下列说法**错误**的是：

A． fork 函数被当前进程调用后，内核将父子进程的页面标记为只读，以支持写时复制（COW）机制。

B．execve 函数被调用后，程序计数器会被重新设置。

C．对 munmap 函数删除的区域再引用会导致段错误。

D．交换空间的大小与当前运行的进程能够分配的虚拟页面总数无关。

11．下列关于全球 IP 因特网的描述中**正确**的是：

A．IP 协议提供了一种从进程到进程的递送机制。

B．TCP 协议提供了进程间可靠的全双工连接。

C． IP 地址结构中地址的字节顺序是大端法还是小端法，要根据主机字节顺序来确定。

D．多个域名不可能被映射为同一个 IP 地址。

12．套接字接口结合 Unix I/O 函数，可以创建网络应用。下列相关描述中，**错误**的是：

A．socket 函数成功返回描述符以后，还不能直接用于读写。

B． TCP 连接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。

C．bind 函数用于将服务端的主动套接字转化为监听套接字。

D．connect 函数会阻塞，直到连接成功建立或者发生错误。

13．下列关于进程线程的描述中，**错误**的是：

A．每个线程有自己的栈，线程运行时不能访问其他线程的栈上的数据。

B．一个进程的多个线程共享该进程的代码段、数据段和堆。

C．操作系统的调度程序选择某个线程在处理器上运行。

D．当主线程正确执行 pthread_create 后，必须使用 pthread_join 或 pthread_detach，对等线程才会运行。

14．下列关于摩尔定律的描述，**不正确**的是：

A．摩尔定律预测了芯片上晶体管数量的增长趋势，这种增长在一定程度上推动芯片厂商尝试采用 3D 堆叠技术实现单位面积晶体管数量的增加。

B．摩尔定律所描述的晶体管数量增长与芯片成本降低的关系，促使半导体产业不断追求技术进步以获取更高利润。

C． 登纳德缩放（Dennard Scaling）定律的失效对摩尔定律产生了负面影响，使得芯片性能无法提升。

D．尽管面临诸多挑战，摩尔定律在过去几十年间仍然是半导体产业发展的重要指导原则，并推动了消费电子产品的创新和普及。

15．下列**不属于**数据密集型（data-intensive） 的任务是：

A．沃尔玛每天处理数亿件商品销售数据，进行供应链管理和市场趋势分析。

B．智利的 LSST 望远镜每 3 天扫描整个南部天空，生成大量图像数据并存储。

C．国家计算流体力学实验室使用“神威•太湖之光”，对“天宫一号”的返回路径进行数值模拟。

D．谷歌数据中心为全球用户提供网页搜索、邮件收发、在线文档等服务。

得分

第二题（15 分）

请结合教材第四章“处理器体系结构”的有关知识回答问题。

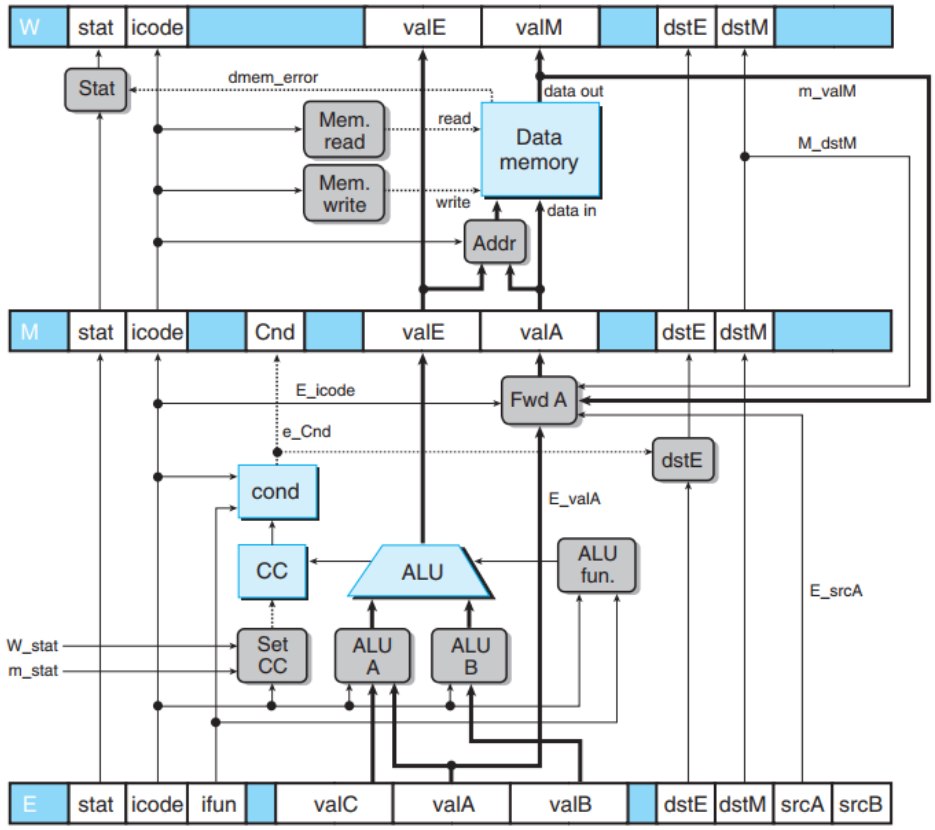
1） 现在 Y86-64 ISA 中加入一条新指令：cdeclXX，条件减一。其功能和用法为，如果条件码满足条件（同 cmovXX 系列指令），则将寄存器数据减 1。cdeclXX 编码如下：

C	Fn	0xF	rB
4 位	4 位	4 位	4 位

考虑教材中的 SEQ 处理器，进行必要的改动使得 cdeclXX 指令能够根据之前指令产生的条件码执行操作，并根据计算结果正常设置条件码。请填写下表中缺失的 4 条操作，每空只包含一条操作（或其一部分）。（每空 2 分，共 8 分）

Stage	cdecl rB
Fetch	icode:ifun ← M_1[PC] rA:rB ← ①_____ ②_____
Decode	valB ← R[rB]
Execute	③_____ Cnd ← Cond(CC, ifun)
Memory	none
Write back	④_____
PC Update	PC ← valP

2） 考虑教材中的 PIPE 处理器，部分如下图所示。在取指阶段，PC 选择逻辑从三个程序计数器源中进行选择。请根据 PIPE 处理器的实现，补全下面的 HCL 代码。（每空 1 分，共 3 分）




```
word f_pc = [
    M_icode ==IJXX && ①_______ : ②_______;
    W_icode == IRET : ③_______;
    1 : F_predPC
]
```

3）在教材中的 PIPE 处理器增加 cdelXX 指令，分支预测的策略为“总是跳转”。考虑如下代码片段，回答以下问题。

```
# long bar(long x)
# x in %rdi
bar:
    xorq    %rax, %rax
    irmovq  $1, %rsi
    rrmovq  %rsi, %r9
Lo:
```

```
    xorq    %r9, %rsi
    cdecle  %rdi
    addq    %rdi, %rax
    andq    %rdi, %rdi
    jne     Lo
    ret
```

周期数计算从执行 bar 第一条指令开始，直到其返回指令 ret 完全通过流水线为止。在调用 bar 之前，处理器已经执行过足够多的 nop 指令。

当函数 bar 的参数 x=0 时，需要执行_____个周期；x=3 时需要执行_____个周期。（每空 2 分，共 4 分）

得分

第三题（15 分）

Part A: 分析下面这段代码的行为，假设 file.txt 文件存在，且不需要考虑 open()函数未能正常打开文件的情况。

```
#include <csapp.h>
int main(){
    int fd[3];
    fd[0] = open("./file.txt", O_RDWR | O_TRUNC, 0);
    fd[1] = open("./file.txt", O_RDWR | O_TRUNC, 0);
     write(fd[0], "Harry", 5);

     if (fork() == 0) {
         dup2(fd[0], fd[1]);
         write(fd[1], "Voldemort", 9);
         fd[2] = open("./file.txt", O_RDWR, 0);
         printf("%d\n", fd[2]);
         exit(0);
     }
     wait(NULL);

     fd[1] = dup(fd[0]);
     write(fd[0], "Hermione", 8);
     write(fd[1], "Ron", 3);

     fd[2] = open("./file.txt", O_RDWR, 0);
     printf("%d\n", fd[2]);

}
```

1） 运行这段代码，在终端上可以看到两行输出，第一行的内容为_____，第二行的内容为_____。(每空 1 分)

2）运行这段代码后，file.txt 文件的内容为_____。（3 分）

Part B:分析下列代码的行为，其功能是利用信号处理函数输出数列内容。

```
#include <csapp.h>
#define N 10
int flag = 0;
void handler1() {
    for (int i = 0; i < N; i += 2){
        printf("%d ", i)
        fflush(stdout);
```

```
    }
    flag++;
}
void handler2() {
    for (int i = 1; i < N; i += 2){
        printf("%d ", i);
        fflush(stdout);
    }
    flag++;
}
int main() {
    pid_t pid = fork();

    if (pid == 0) {
        signal(SIGUSR1, handler1);
        signal(SIGUSR2, handler2);
        while (flag < 2)
            sleep(1);
    }
    else{
        kill(pid, SIGUSR1);
        kill(pid, SIGUSR2);
        wait(NULL);
    }
    return 0;
}
```

1） 运行这段代码后，发现程序没有输出，且陷入死循环。请简要指出其原因，并给出解决方案。（2 分）

2） 在解决上述问题之后，第一次运行得到“1357902468”；但再次运行得到的却是“1302468579”，与期望不符。请结合“编写信号处理函数”的知识，补全 handler1()函数中缺失的代码，以保证运行结果的正确性。（3 分）

```
void handler1(){
    sigset_t mask, prev_mask;
    ____A____(____B____);
    ____C____(____D____, &mask, &prev_mask);
    for (int i = 0; i < N; i += 2){
        printf("%d ", i)
        fflush(stdout);
    }
    flag++;
    ____E____(____F____, &prev_mask);
}
```

基于下列选项选择序号填写：

① Sigfillset;　② Sigpromask;　③ Sigemptyset;
④ SIG_BLOCK;　⑤ SIG_UNBLOCK;　⑥ SIG_SETMASK;
⑦ &mask;　⑧ &prev_mask;　⑨ NULL;

A:_____
B:_____
C:_____
D:_____
E:_____
F:_____

Part C: 分析下面这段代码的行为，假设 file.txt 文件存在且**初始内容为空**，并且不需要考虑 open()函数未能正常打开文件的情况。

```
#include <csapp.h>

int main() {
    char *numbers = "0123456789";

    char buffer[100];
    buffer[0] = '\0';

    pid_t father_pid = getpid();

    for (int i = 0; i < 10; ++i){
        if (fork() != 0) {
            strncat(buffer, numbers + i, 1);
        }
    }

    int fd = open("./file.txt", O_RDWR , 0);
    write(fd, buffer, strlen(buffer));

    while(wait(NULL) != -1){}

    if (getpid() == father_pid) {
        close(fd);
        fd = open("./file.txt", O_RDONLY, 0);
        char content[100];
        read(fd, content, 100);
        printf("%c\n", content[3]);
        printf("%c\n", content[6]);
    }
    exit(0);
}
```

如果你不清楚 string.h 中 strncat()函数的行为，可以参考下列说明：

strncat()函数用于将一个字符串追加到另一个字符串的末尾。它接受一个目标字符串 dest，一个源字符串 src 和最大追加的字符数 n。在 src 的长度至少为 n 时，这个函数将 src 的前 n 个字符拼接到 dest 字符串的后面。例如：

```
char *dest = "Astarion";
char *src = "Shadowheart";
strncat(dest, src, 6);
printf("%s\n", dest);
```

这段代码将输出 AstarionShadow。

1）运行上述代码，在终端上会看到两行输出，第一行的内容可能为_____，第二行的内容可能为_____。（你需要列出所有可能出现的输出，每种可能性用逗号隔开，每空 1 分）

2） 在运行这段代码之后，file.txt 中的内容有_____种可能的结果。（3 分）（注意，字符串级别相等的内容属于同一种可能的结果）

得分

第四题（20 分）

Part A: 虚拟内存(11 分)

考虑如下的一种小型系统：内存的最基本单位为 Byte，任何数据类型的长度都是 Byte 的整数倍，且按照小端法表示。该系统中，单个页表项占据 2B，地址编码为 16 位，页大小为 128B，页表为一级页表。目前，该系统支持的虚拟地址空间大小为 8KB,实际拥有的物理内存为 2KB。系统中的 TLB 为 8 组 2 路组相联，共有 16 个条目，替换策略为 LRU。TLB 被系统单独储存，不占用物理内存。

在运行了一段时间后，这个系统内的 TLB 布局如下：

Set	Tag	PPN	Valid	Tag	PPN	Valid
0	2	6	1	5	E	0
1	0	D	0	3	7	1
2	1	3	1	5	A	1
3	6	B	0	4	F	1
4	0	8	1	7	B	1
5	5	E	0	2	4	0
6	1	4	0	0	C	1
7	3	8	0	6	0	0

页表的前 16 个页表项布局如下（页表项经过翻译，以便于阅读）：

VPN	PPN	Valid	VPN	PPN	Valid
00	-	0	08	-	0
01	-	0	09	E	1
02	-	0	0A	3	1
03	2	1	0B	-	0
04	8	1	0C	1	1
05	-	0	0D	-	0
06	C	1	0E	D	1
07	-	0	0F	9	1

1） 该系统的一个完整的页表中共有_____个页表项，该系统需要____个物理页来存储页表结构。（每空 1 分）

2） 该系统的 CPU 执行指令时，取到一条虚拟地址为 0x0721。经过翻译，该虚拟地址对应的物理地址是_____（以十六进制方式书写，如 0x012）。（2 分）

3） 不考虑 2）中操作对 TLB 和页表的修改。该系统的 CPU 执行指令时，依次连续访问了以下若干虚拟地址：0x11A5，0x1419，0x1C01，0x0863，0x1C7F。每次 TLB 不命中时，都会将翻译结果的 PPN 替换入 TLB。假设这些操作过程中未发生缺页异常，则共产生了_____次对物理内存的访问（不考虑对指令地址的翻译和访问；不考虑高速缓存，即，无论物理内存中高速缓存命中与否，均只看作一次访问）。（3 分）

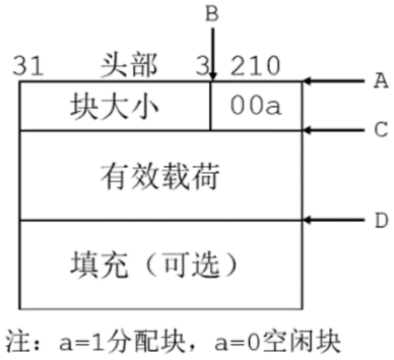
4） 不考虑 2）和 3）中操作对 TLB 和页表的修改。该系统的 CPU 执行指令时，取到一条虚拟地址为 0x00D0，程序运行过程中一定会发生_____（单选，填写选项，下同）。该系统的 CPU 执行指令时，取到一条虚拟地址为 0x2226，程序运行过程中一定会发生_____。（每空 1 分）

A．正常访存　　B.缺页异常　　C．段错误

5） 在真实环境中，为了节省内存等目的，往往采用多级页表的方式。考虑一个 IA32 体系下的系统，单个页表项占据 4B，地址编码为 32 位，页大小为 4KB，使用了两级页表，每个页表均占据一页。设一级页表基址为 0x00E77000，则虚拟地址 0x19260817 对应的一级页表项地址为_____（以十六进制方式书写）。假设该页表项的值为 0x0D000721，则该虚拟地址对应的二级页表项地址为_____（以十六进制方式书写）。（每空 1 分）

Part B: 动态内存分配(9 分)

小明开发了一种简单的动态内存分配器，仅采取无脚部的隐式空闲链表组织方式，没有额外的数据结构，内存分配块采取 8 字节对齐，其内存块如图 1 所示，块大小采用小端法存储。分配算法采用首次适配，堆空间足够大。



注：a=1分配块，a=0空闲块

图1

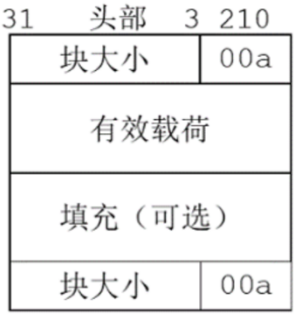


图2

1） 在如上动态内存分配器的机器上（初始堆为空），小张执行 `void *p = malloc(5);` 后，p 实际指向的位置为图 1 中的_____位置（选填”A”、”B”、”C”或”D”），此时 A 处的 1 个字节的内容为_____（用十六进制表示，如 0xFF）。（每空 1 分）

2） 小明设计的分配器采取的空闲块合并方式为推迟合并，假设某次合并前，堆空间共有 n 个独立的内存块，则该合并过程的时间复杂度至少为_____。（单选，填写选项）（1 分）

A．$O(1)$ B．$O(logn)$ C．$O(n)$ D．$O(n^2)$

3） 为了提高空闲块合并的效率，小明为每个内存块在结尾处都添加了脚部，如图 2 所示，空闲块合并策略更改为立即合并，假设某次释放某已分配内存块后，堆空间共有 n 个独立的内存块，则此时的合并操作的时间复杂度至少为_____。（单选，填写选项）

A．$O(1)$ B．$O(logn)$ C．$O(n)$ D．$O(n^2)$

此时使用 malloc 申请内存时，单个内存块的内存利用率最低为_____。（用最简分数表示，内存利用率=申请块大小/实际分配块大小）（每空 1 分）

4） 为了进一步提升内存利用率，小明只对空闲块添加脚部，而不对分配块添加脚部。则此时对已分配块执行释放操作时，_____有足够空间存放脚部。（1 分）

A．一定 B．不一定

5） 小明改进后的合并方法如下：每当释放一个内存块时，取出该内存块头部之前的四个字节的数据，检查最低位是否为 0，如果最低位为 0，再利用取出的字节的

其他部分读出块大小，并根据大小找到对应的头部位置，如果此时找到的头部与取出的四个字节内容完全相同，则合并这个块。小张认为小明此时的合并方式存在问题，于是构造了如下代码进行攻击，假设初始堆为空：

```
#include <stdlib.h>
int main() {
    void *p1 =  calloc(28);
    void *p2 =  calloc(32);
    *((char*)p1 + 24 ) = 0x10;
    *((char*)p1 + ① ) =  ② ;
    free(p2);
}
```

为了使小明的内存分配器合并空闲块出错，上述代码段①、②处应当分别填写_____、_____，此时在执行完 free(p2) ;后，小明的内存分配器会错误地认为从地址 ((char*)p1 + _____) 开始的堆内存都是空闲的。（3 分，代码两空全部正确得 1 分，最后一空 2 分）

得分

第五题（20 分）

Part A (12 分)

北京大学南门外新开设了一家小型的电影院，由于电影院规模较小，只有两个影厅——影厅 A 与影厅 B，并且工作人员只有一名同时充当售票员的经理。这个影院有一些奇怪的规定。

- 每个影厅正在放映的电影是自动续播的，只有当最后一个观众主动离开当前的影厅时结束当前影片的放映。
- 两个影厅同时放映电影，选择观看两个影厅中任意一个正在放映的影片的观众买票后可立即进入，且后续不能更换影厅。
- 两个影厅内的总观众人数不能超过 100 人。购票人数到达上限后，购票通道会立即关闭。直到有观众离开影院，后续观众才可以继续购票。
- 如果有推销商来找经理推销新影片或者高级放映设施，为了能够及时接待推销商，经理会立即关闭购票入口，新到来的观众不能购票入场。
- 每次经理只能与一名推销商会谈，后续到达的推销商需要等待。经理与已经到达的所有推销商都会谈过后，电影院才能重新开业接待观众。

1．这名可怜的经理希望北京大学的同学能帮他开发出一套系统，实现上面的逻辑，从而减轻他的工作量。学过 ICS 的小 A 一眼就看出这个问题可以建模为读者-写者问题，并且写出了代码。

1） 补全代码（5 分）

```
#include “csapp.h”
#define MaxAudienceCount 100

int AudienceCount = 0; // 影院的观众数
int PromoterCount = 0; // 推销商的数量
信号量 MovieA = 1;
信号量 MovieB = 1;
信号量 Promote = 1;
信号量 MaxAudience = MaxAudienceCount;
信号量 AudienceMutex = 1;  // 变量 AudienceCount 的互斥锁
信号量 PromoterMutex = 1;  // 变量 PromoterCount 的互斥锁

AudienceA(){
    while(1){
        P([  A  ]);
        P(MovieA);
        P([  B  ]);
        AudienceCount++;
        if (AudienceCount == 1){
```

```
            P([  C  ]);
        }
        V([  B  ]);
        V(MovieA);

        // Watch the movie


        P(AudienceMutex);
        AudienceCount--;
        if (AudienceCount == 0){
            V(Promote);
        }
        V(AudienceMutex);
        V(MaxAudience);
    }
}

AudienceB(){
    while(1){
        P([  A  ]);
        P([  D  ]);
        P([  B  ]);
        AudienceCount++;
        if (AudienceCount == 1){
            P([  C  ]);
        }
        V([  B  ]);
        V([  D  ]);

        // Watch the movie


        P(AudienceMutex);
        AudienceCount--;
        if (AudienceCount == 0){
            V(Promote);
        }
        V(AudienceMutex);
        V(MaxAudience);
    }
}

Promoter(){
    while(1){
        P(PromoterMutex);
        PromoterCount++;
        if (PromoterCount == 1){
```

```
            P(MovieA);
            P(MovieB);
        }
        V(PromoterMutex);


        P([  E  ]);
        // Try to promote
        V([  E  ]);

        P(PromoterMutex);
        PromoterCount--;
        if (PromoterCount == 0){
            V(movieA);
            V([  D  ]);
        }
        V(PromoterMutex);
    }
}
```

A:_____ B:_____ C:_____

D:_____ E:_____

（选择下述编号填入字母表示的空缺内，注意不同字母也可能填入相同的数字编号）

① MovieA

② MovieB

③ Promote

④ MaxAudience

⑤ AudienceMutex

2） 在这个实现的逻辑下，观众有可能饥饿吗？_____（1 分）

A． 可能

B． 不可能

2． 让我们再次回归到最基本的第二类读者-写者问题的另一种解法，如下述代码所示。学习了并发之后的你迅速提取出了有效信息并补全了代码。（6 分）

ActiveR 表示正在读的读者的数量，WaitingR 表示正在等待的读者的数量；ActiveW 表示正在写的写者的数量，WaitingW 表示正在等待的写者的数量。

```
int ActiveR = 0;
int WaitingR = 0;
int ActiveW = 0;
int WaitingW = 0;
信号量 mutex = 1;
信号量 waitingToRead = [  A  ];
```

```
信号量 waitingToWrite = [  B  ];

Reader() {
    P(mutex);
    while ((ActiveW + WaitingW) > 0) {
        WaitingR++;
        [  C  ];
        [  D  ];
        P(mutex);
        WaitingR--;
    }
    ActiveR++;
    V(mutex);

    // Perform actual read-only access
    P(mutex);
    ActiveR--;
    if (ActiveR == 0 && WaitingW > 0) {
        V(waitingToWrite);
    }
    V(mutex);
}

Writer () {
    P(mutex);
    while (([  E  ]) > 0) {
        WaitingW++;
        [  C  ];
        P(waitingToWrite);
        P(mutex);
        WaitingW--;
    }
    ActiveW++;
    V(mutex);

    // Perform actual write-only access
    P(mutex);
    ActiveW--;
    if (WaitingW > 0) {
        V(waitingToWrite);
    } else if (WaitingR > 0) {
        int n = WaitingR;
        while (n--) {
            V([  F  ]);
        }
    }
    V(mutex);
}
```

A:_____ B:_____ C:_____

D:_____ E:_____ F:_____

（选择下述编号填入字母表示的空缺内，注意不同字母也可能填入相同的数字编号）

① 0

② 1

③ ActiveR + ActiveW

④ ActiveR + WaitingR

⑤ ActiveW + WaitingW

⑥ P(mutex)

⑦ V(mutex)

⑧ P(waitingToRead)

⑨ V(waitingToRead)

Part B (8 分)

完全人格，首在体育。小北同学近日遇到了考入某体育强校的李华同学，但二者就谁是体育世一大产生了争执，因此决定通过拔河比赛一决胜负。为了确保公平性，小北和李华各自找来了自己的学长作为裁判进行比赛计分。作为计算机专业的同学，你希望编写一个运行在 Linux 内核上的 C 程序模拟这个场景，以分析小北是否遥遥领先。

为了模拟上述场景，我们用进程表示选手团和裁判团，每个进程中又用两个线程去模拟两位选手和两位裁判。每个“选手”线程通过信号将自己视角下的比赛情况发给对应的“裁判”线程，“裁判”线程将比赛结果记录在文件里以便最后登记分数。

补充说明：

- 在 POSIX 语义下，多线程进程接收信号时将随机指定一个线程进行接收
  - 但在 Linux 系统下，通过在主线程中 BLOCK 某信号、在对等线程中 UNBLOCK 该信号，可以强制指定该对等线程接收该信号
  - 本题利用这种方法，使两个线程 judger1 和 judger2 分别处理进程收到的 SIGUSR1 和 SIGUSR2 信号
- volatile 修饰关键词要求编译器对于指定的变量的每次读写都必须使用内存中的值
- register 修饰关键词要求编译器必须将指定的变量分配在寄存器上
- 题目在 Ubuntu 24.04.1 上使用 gcc 13.3.0 (Ubuntu 13.3.0-6ubuntu2~24.04)进行编译，编译命令为 gcc input.c csapp.c -o input。题目环境的内核版本为 5.15.167.4。解决本题无需课程内容以外知识，提供本信息仅为避免你臆想不必要的情况。

input.c

```
#include "csapp.h"
void signal_handler(int signo) {
    if (signo == SIGUSR1) printf("A\n");
    if (signo == SIGUSR2) printf("B\r");
}

void *judger(void *num) {
    ...
    //在省略的代码中使用 pthread_sigmask 分别取消对 SIGUSR1 和 SIGUSR2 信号的阻塞
    //(int)num 为 1 则取消 SIGUSR1 的阻塞，(int)num 为 2 则取消 SIGUSR2 的阻塞
    for (;;);
}

int judger_pid = 0;

void *player(void *arg) {
    int num = (int)(uint64_t)arg;
    volatile static int flag = 0;
    register int val = 0;
    for (int i = 1; i <= 10; i++) {
        val = flag; // *2
        if (num == 1) {
            if (val <= 0) ++val;
            flag = val;
            printf("Player %d: %d %c\n", num, val,
                  "_A"[val > 0]);
            if (val > 0) kill(judger_pid, SIGUSR1);
        } else {
            if (val >= 0) --val;
            flag = val;
            printf("Player %d: %d %c\n", num, val,
                  "_B"[val < 0]);
            if (val < 0) kill(judger_pid, SIGUSR2);
        }
    }
}

int main() {
    ...
    // 在省略的代码中使用 pthread_sigmask 阻塞 SIGUSR1 和 SIGUSR2 信号
    // 并设置 SIGUSR1 和 SIGUSR2 的信号处理函数为 signal_handler
    pid_t player_pid;
    if (player_pid = fork()) { // *1
        freopen("game.out", "wb", stdout); //题目保证硬盘空间足够
        setvbuf(stdout, NULL, _IOLBF, 1024);
        // 裁判团
```

```
        pthread_t judger1, judger2;
        pthread_create(&judger1, NULL, judger, (void *)1);
        pthread_create(&judger2, NULL, judger, (void *)2);
        waitpid(player_pid, NULL, 0);
        sleep(1); // 题目保证此时所有 pending 信号均处理完成
        _exit(0);
    } else {
        sleep(1); // 题目保证此时信号处理程序已按要求生效
        // 玩家团
        judger_pid = getppid();
        pthread_t player1, player2;
        pthread_create(&player1, NULL, player, (void *)1);
        pthread_create(&player2, NULL, player, (void *)2);
        pthread_join(player1, NULL);
        pthread_join(player2, NULL);
        return 0;
    }
}
```

1） 在 player 函数中，使用了_____个共享变量？(同一个变量多次出现算 1 个）（1 分）

2） 看到代码后，你很快发现：在信号处理程序里使用了不安全的 printf 函数！老师在课堂上讲过著名的“灵魂出窍”死锁案例，而题目的程序同时在 main 函数和信号处理程序都使用了 printf。

题目中的程序_____（会 / 不会）出现死锁？原因包括：printf _____（是 / 不是）异步信号安全的；printf _____（是 / 不是）线程安全的；在题目的程序中， signal_handler 中的 printf_____（可能/不会）被打断。（4 分）

3） 下面是上面代码某次运行时输出的结果：

stdout

```
Player 1: 1   A
Player 1: 1   A
Player 1: 1   A
Player 1: 1   A
Player 1: 1   A
Player 1: 1   A
Player 1: 1   A
Player 1: 1   A
Player 2: 0   _
Player 2: 0   _
Player 2: -1  B
Player 1: 1   A
Player 1: 0   _
Player 2: -1  B
```

game.out

```
A
A
A
A
B
A
```

```
Player 2: -1  B
Player 2: -1  B
Player 2: -1  B
Player 2: -1  B
Player 2: -1  B
Player 2: -1  B
```

① 下列哪一项不是导致本题程序多次运行的结果不同的原因？_____（1 分）

A．操作系统调度进程具有不确定性

B．操作系统调度线程具有不确定性

C．操作系统下发信号的时机具有不确定性

D．操作系统处理系统调用具有不确定性

② 下列哪一选项会导致上面的运行结果中，game.out 的 A 比 stdout 的少？_____（1 分）

除此之外，还有哪一选项会导致上面的运行结果中，game.out 的 B 比 stdout 的少？_____（1 分）

A．多个到达的信号可能只被处理一次

B．后面的字符可能在缓冲区内覆盖前面的字符

C．缓冲区内的字符没有刷新到文件，程序就退出了

D．多次相同的 kill 可能只有一次进入内核

E．父进程中的两个线程产生 race Condition

F．子进程中的两个线程产生 race Condition

G．由于磁盘太慢等原因，产生写不足