装订线内

不要答题

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：2022 年 12 月 19 日 **小班号：**_____

题号	一	二	三	四	五	六	七	**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 14 页。

第一题 填空（30 分）

1. 对比指令系统体系结构（ISA）的两大类别，_____通常比_____的指令数量和寻址方式都少。

2. 这是一个 IEEE754 标准的单精度浮点数的十六进制形式：$(41\ 20\ 00\ 00)_{16}$。该数转换成十进制是_____。（提示：指数域有 8 比特）

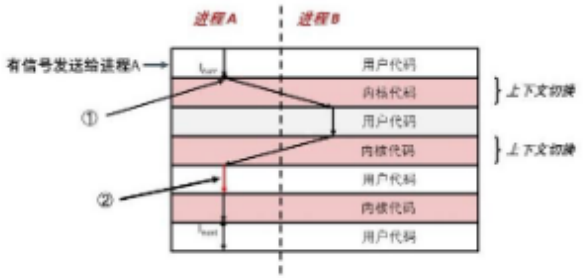
3. 在 x86-64/Linux 的约定中，函数传递参数的前两个分别放在_____和_____寄存器。

4. 为了构造可执行文件，静态链接器必须完成_____和_____这两个主要任务。

5. 在当前的 Linux 系统中，延迟绑定是通过两个数据结构之间的交互来实现的，这两个重要的数据结构分别是_____和_____。

6. a 和 b 表示当前目录中的目标模块或者静态库，而 a—>b 表示 a 依赖于 b，也就是说 b 定义了一个被 a 占用的符号。对于下面的每种场景，请给出最小的命令行（即一个含有最少数量的目标文件和库参数的命令），使得静态链接器能解析所有的符号引用。
   ① p.o —>libx.a —> liby.a
   _____
   ② p.o —>libx.a —>liby.a 且 liby.a —> libx.a —> p.o
   _____

7. 请根据下图所示流程完成填空。
   ① 处发生了什么？_____。
   ② 执行的代码是什么？_____。



8. 实现 Shell 程序时，在解析了命令行后，会检查第一个命令行参数是否是一个内置 Shell 命令。如果不是，则 Shell 程序会调用_____①_____和_____②_____函数完成命令行命令。

9. 在支持虚拟地址空间的系统中，CPU 取到虚拟地址后，发给_____①_____，由①将虚拟地址转换为物理地址。为了加快地址翻译速度，需要在①中引入_____②_____。

10. 假设一个小型系统：内存按字节寻址，内存访问按 1 个字节完成。虚拟地址空间为 2^{14}，物理内存空间为 2^{12}，页面大小为 2^{6}。TLB 是 4 路组相联，有 16 个条目。页表是一级页表。

TLB内如如下表所示：

Set	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid
0	03	–	0	09	0D	1	00	–	0	07	02	1
1	03	2D	1	02	–	0	04	–	0	0A	–	0
2	02	–	0	08	–	0	06	–	0	03	–	0
3	07	–	0	03	0D	0	0A	34	1	02	–	0

页表的前16项如下表所示：

VPN	PPN	Valid
00	28	1
01	–	0
02	33	1
03	02	1
04	–	0
05	16	1
06	–	0
07	–	0

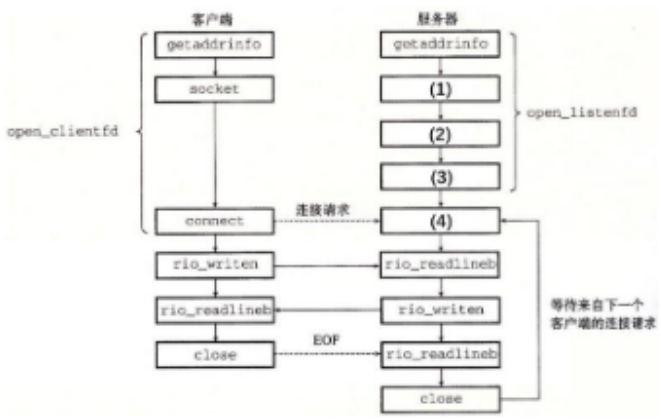
VPN	PPN	Valid
08	13	1
09	17	1
0A	09	1
0B	–	0
0C	–	0
0D	2D	1
0E	17	1
0F	0D	1

如果 CPU 执行取到一条虚拟地址为 0x0395，经地址翻译后，该虚拟地址对应的物理地址是_____。

11. Linux 中可以一个磁盘上的文件与虚拟内存中的区域关联起来，这一过程称为_____。

12. IP 地址可以用十六进制表示，也可以用点分十进制表示。请写出 IPv4 地址`0x24982c5f`的点分十进制表示：_____。

13. 下图给出了一个典型的客户端-服务器套接字交互流程。应当填在空(2)处的函数名称是：_____（accept / bind / listen / socket）。



14. 多线程的执行模型在某些方面和进程的执行模型是相似的。每个进程开始生命周期时都是单一线程，这个线程称为主线程（main thread）。在某一时刻，主线程创建一个_____，从这个时间点开始，两个线程就并发地运行。

15. 以提供互斥为目的的二元信号量常常也称为_____。

第二题 处理器（15 分）

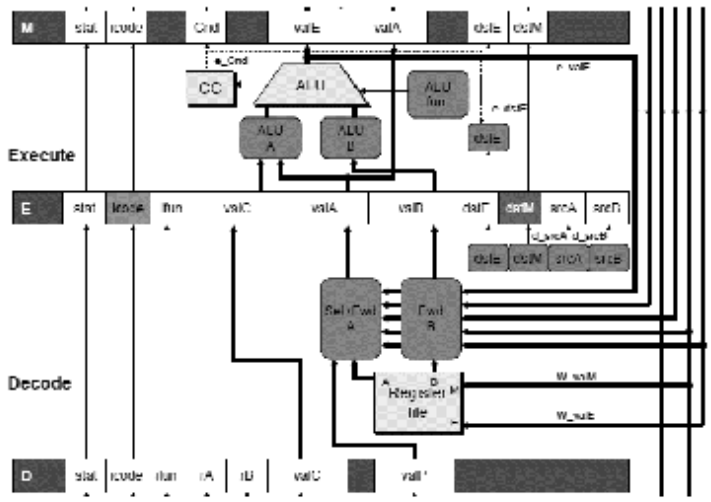
基于本课程教材中讲述的 Y86 流水线处理器结构，如果一条 mrmovq 指令的目标寄存器在紧随的指令里作为源操作数使用了，就会导致"load-use"冒险，是无法通过前递（Forwarding）技术解决的。

（1）要解决"load-use"冒险，首先要能检测出来。根据下面的处理器信号连接图，补全检测"load-use"冒险的 HCL 表达式。（每空 1 分）

_____ in { IMRMOVQ, IPOPQ } && E_dstM in {_____, _____ }

（2）检测到"load-use"冒险后，各流水级应该如何设置？在下表中填入 normal/stall/bubble。（每空 1 分）

F	D	E	M	W
stall				normal



（3）ret 指令因为在流水线很晚阶段才能获得转移目标地址，所以也要特殊处理。参考上面的处理器信号连接图（注意：图中信号可能不完整，但足够推测出所需的信号），补全检测 ret 指令处理时机的 HCL 表达式。（每空 1 分）

IRET in {_____,_____,_____ }

（4）检测到需要处理的 ret 指令后，各流水级应该如何设置？在下表中填入 normal/stall/bubble。（每空 1 分）

F	D	E	M	W
stall				normal

（5）执行 ret 指令时，其后会插入_____个周期的 bubble。（1 分）

（6）什么场景下，上面提到的 load-use 和 ret 需要处理的情况会同时出现？（2 分）

_____

第三题 链接（10 分）

本题基于下列 c 语言文件所编译生成的 main.o 和 addvec.o，编译和运行在 X86-64/Linux 下完成，编译过程未加优化选项。

```
//main.c
#include <stdio.h>
#include "vector.h"
int x[2] = {1, 2};
int z[2];
int main(int argc, char** argv){
  int y[2] = {3, 4};
  int n=2;
  addvec(x,y,z,n);
  printf("z=[%d %d]\n",z[0],z[1]);
  return 0;
}
```

```
//addvec.c
void addvec(int *x, int *y,
          int *z, int n) {
    int i;
    for (i = 0; i < n; i++)
       z[i] = x[i] + y[i];
}
```

（1）对于每个下表中给出的符号，请用"是"或"否"指出它是否在模块 main.o 的.symtab 节中有符号表条目。如果存在条目，则请指出该符号的符号类型（局部、全局或外部）。并进一步指出定义该符号的模块（main.o 或 addvec.o）、以及此符号在该模块中所处的节名或伪节名；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	节或伪节名
x				
y				
z				
n				
addvec				

（2）基于 main.o 和 addvec.o 生成可执行程序 m 的时候，会进行若干重定位。下述代码是对可执行程序 m 执行 objdump -D m 以后的片段结果，请根据相关信息进行填空。

```
0000000000400238 <.interp>:
  400238:      2f                   (bad)
  400239:      6c                   insb   (%dx),%es:(%rdi)
  40023a:      69 62 36 34 2f 6c 64    imul   $0x646c2f34,0x36(%rdx),%esp
  400241:      2d 6c 69 6e 75          sub    $0x756e696c,%eax
  400246:      78 2d                js     400275 <_init-0x153>
  400248:      78 38                js     400282 <_init-0x146>
  40024a:      36                   ss
  40024b:      2d 36 34 2e 73          sub    $0x732e3436,%eax
  400250:      6f                   outsl  %ds:(%rsi),(%dx)
  400251:      2e 32 00             xor    %cs:(%rax),%al
      ...

00000000004003e0 <printf@plt-0x10>:
  4003e0:      ff 35 0a 0c 20 00       pushq  0x200c0a(%rip)        # 600ff0 <_GLOBAL_OFFSET_TABLE_+0x8>
  4003e6:      ff 25 0c 0c 20 00       jmpq   *0x200c0c(%rip)        # 600ff8 <_GLOBAL_OFFSET_TABLE_+0x10>
  4003ec:      0f 1f 40 00             nopl   0x0(%rax)

00000000004003f0 <printf@plt>:
  4003f0:      ff 25 0a 0c 20 00       jmpq   *0x200c0a(%rip)        # 601000 <_GLOBAL_OFFSET_TABLE_+0x18>
  4003f6:      68 00 00 00 00          pushq  $0x0
  4003fb:      e9 e0 ff ff ff          jmpq   4003e0 <_init+0x18>
0000000000400400 <__libc_start_main@plt>:
  400400:      ff 25 02 0c 20 00       jmpq   *0x200c02(%rip)        # 601008 <_GLOBAL_OFFSET_TABLE_+0x20>
  400406:      68 01 00 00 00          pushq  $0x1
  40040b:      e9 d0 ff ff ff          jmpq   4003e0 <_init+0x18>
      ...
0000000000400528 <main>:
  400528:      55                   push   %rbp
  400529:      48 89 e5             mov    %rsp,%rbp
  40052c:      48 83 ec 20          sub    $0x20,%rsp
  400530:      89 7d ec             mov    %edi,-0x14(%rbp)
```

```
  400533:       48 89 75 e0           mov    %rsi,-0x20(%rbp)
  400537:       c7 45 fc 02 00 00 00    movl   $0x2,-0x4(%rbp)
  40053e:       8b 45 fc              mov    -0x4(%rbp),%eax
  400541:       89 c1                 mov    %eax,%ecx
  400543:       ba _____①_____          mov    $___________,%edx
  400548:       be _____②_____          mov    $___________,%esi
  40054d:       bf _____③_____          mov    $___________,%edi
  400552:       e8 _____④_____          callq  ___________
  400557:       8b 15 __⑤_________      mov    ___________ (%rip),%edx
  40055d:       8b 05 __⑥_________      mov    ___________ (%rip),%eax
  400563:       89 c6                 mov    %eax,%esi
  400565:       bf __⑦_________         mov    $___________,%edi
  40056a:       b8 00 00 00 00          mov    $0x0,%eax
  40056f:       e8 __⑧_________         callq  ___________ <printf@plt>
  400574:       b8 00 00 00 00          mov    $0x0,%eax
  400579:       c9                    leaveq
  40057a:       c3                    retq
  40057b:       90                    nop
000000000040057c <addvec>:
  40057c:       55                    push   %rbp
  40057d:       48 89 e5              mov    %rsp,%rbp
  400580:       48 89 7d e8           mov    %rdi,-0x18(%rbp)
  400584:       48 89 75 e0           mov    %rsi,-0x20(%rbp)
  400588:       48 89 55 d8           mov    %rdx,-0x28(%rbp)
  40058c:       89 4d d4              mov    %ecx,-0x2c(%rbp)
  40058f:       c7 45 fc 00 00 00 00    movl   $0x0,-0x4(%rbp)
  400596:       eb 4a                 jmp    4005e2 <addvec+0x66>
  400598:       8b 45 fc              mov    -0x4(%rbp),%eax
  40059b:       48 98                 cltq
  40059d:       48 8d 14 85 00 00 00    lea    0x0(,%rax,4),%rdx
  4005a4:       00
      ...
0000000000600fe0 <.got>:
      ...
0000000000600fe8 <_GLOBAL_OFFSET_TABLE_>:
  600fe8:       10 0e                 adc    %cl,(%rsi)
  600fea:       60                    (bad)
      ...
  600fff:       00 f6                 add    %dh,%dh
  601001:       03 40 00              add    0x0(%rax),%eax
  601004:       00 00                 add    %al,(%rax)
  601006:       00 00                 add    %al,(%rax)
  601008:       06                    (bad)
  601009:       04 40                 add    $0x40,%al
  60100b:       00 00                 add    %al,(%rax)
  60100d:       00 00                 add    %al,(%rax)
      ...
0000000000601020 <x>:
  601020:       01 00                 add    %eax,(%rax)
  601022:       00 00                 add    %al,(%rax)
  601024:       02 00                 add    (%rax),%al
      ...
0000000000601028 <y.2304>:
  601028:       03 00                 add    (%rax),%eax
  60102a:       00 00                 add    %al,(%rax)
  60102c:       04 00                 add    $0x0,%al
      ...
0000000000601038 <z>:
      ...
```

编号	重定位条目信息	应填入的重定位引用值
③	r.offset = 0x26　　r.symbol = x r.type = R_X86_64_32　　r.addend = 0	
④	r.offset = 0x2b　　r.symbol = addvec r.type = R_X86_64_PC32　　r.addend = -4	

(3) 基于你对代码的分析，生成该可执行程序的操作系统的动态链接器的路径是（　　）

A) /lib64/ld-linux-x86-64.so.2
B) /lib/ld-linux-x86-64.so.2
C) /lib/ld-x86-64-linux.so.2
D) /lib64/ld-x86-64-linux.so.2

(4) 程序中 printf 的 PLT 表条目是 PLT[_____]，GOT 表条目是 GOT[_____]。（填写数字即可）

第四题 关于异常控制流（10 分）

信号是实现进程间通信的一种重要方式。以下程序中，父进程借助信号把一个随机数传给子进程。

```
1.  #include "csapp.h"
2.  volatile int acc = 0;
3.  volatile int count = 0;
4.  void sigusr1_handler(int sig) {
5.      _________G_________
6.      acc = acc * 2 + __A__;
7.      Kill(getppid(), SIGCONT);
8.  }
9.  void sigusr2_handler(int sig) {
10.     _________H_________
11.     acc = acc * 2 + __B__;
12.     Kill(getppid(), SIGCONT);
13. }
14. void sigcont_handler(int sig) {}
15. int main() {
16.     sigset_t mask_all, mask_father, mask_child;
17.     Sigfillset(&mask_all);
18.     Sigfillset(&mask_father);
19.     Sigfillset(&mask_child);
20.     Sigdelset(&mask_father, SIGCONT);
21.     Sigdelset(&mask_child, SIGUSR1);
22.     Sigdelset(&mask_child, SIGUSR2);
23.     Signal(SIGUSR1, sigusr1_handler);
24.     Signal(SIGUSR2, sigusr2_handler);
25.     Signal(SIGCONT, sigcont_handler);
26.     __C__(SIG_SETMASK, __D__, NULL);
27.
28.     pid_t pid = Fork();
```

```
29.     if (pid == 0) {
30.         for (int i = 0; i < sizeof(int) * 8; i++)
31.             __E__(&mask_child);
32.         printf("child:0x%x\n", acc);
33.     }
34.     else {
35.         srand(time(NULL));
36.         int r = rand();
37.         printf("father:0x%x\n", r);
38.         for (int i = 0; i < sizeof(int) * 8; i++) {
39.             if (r & (__F__ << 31 - i))
40.                 Kill(pid, SIGUSR2);
41.             else
42.                 Kill(pid, SIGUSR1);
43.             __E__(&mask_father);
44.         }
45.         Wait(NULL);
46.     }
47. }
```

1. 假设 G, H 处都为空，该程序需要使得父进程和子进程的输出相同。(共 6 分，每空 1 分)

   (1) 程序填空:

   A:_____

   B:_____

   C:_____

   D:_____

   E:_____

   F:_____

   (2) 第 43 行的代码是必须的吗？如果"是"，请用一句话描述去掉了这一行代码会发生什么。(2 分)

2. 假设 G 处填写 `if (count % 3 == 0) return;` 在 H 处填写 `count++;`
   如果第 36 行生成的随机数是 0xaaaaaaaa，则子进程的输出是
   `child:0x`_____ (1 分)，简要说明理由。(1 分)

第五题 虚拟内存管理（10 分）

1、典型的 Linux 进程的虚拟内存布局如下图所示，请分别给出①②③④对应存放的内容。(每小题 1 分，共 4 分)

2、如上题给出的 Linux 进程的虚拟内存布局图，如果执行*(int*)main=0，会出现何种结果？如果在②和③之间的地址空间位置执行 mov $0x0a, count，会出现何种结果？（忽略 count 的实际意义，假设能正常通过编译）（每小题 1 分，共 2 分）

3、内存安全（共 4 分，找出一个错误给 1 分）

以下代码有 4 处内存安全错误，请指出所在代码行行号并简要说明。

注意：

1) 请仔细阅读注释；
2) 不需要考虑库函数的错误和库函数调用出错的情况（因此不需要考虑返回值检查）；
3) 说明时加上必要的代码行号，请使用简洁的语言；
4) 只需说明为什么会出错，无需给出修补方法。

```
5) #include <assert.h>
6) #include <stdio.h>
7) #include <stdlib.h>
8)
9) // There're 4 memory safety bugs
10)
11)// Assume all calls to library functions succeed
12)// For simplicity, our vector type can only hold ints.
13)typedef struct {
14)     int *data;     // Pointer to our array on the heap
15)     int  length;   // How many elements are in our array
16)     int  capacity; // How many elements our array can hold
17)} Stack;
18)
19)// Get a chunk of heap memory that can accomodate `size` in
   t number
20)int *get_heap_memory(int size) {
21)     int *new_data = (int *)malloc(size);
22)     assert(new_data != NULL);
23)     return new_data;
24)}
```

```
25)
26)Stack *stack_new() {
27)     Stack stack;
28)     stack.data     = get_heap_memory(1);
29)     stack.length   = 0;
30)     stack.capacity = 1;
31)     return &stack;
32)}
33)
34)void stack_push(Stack *stack, int n) {
35)     if (stack->length == stack->capacity) {
36)         int new_capacity = stack->capacity * 2;
37)
38)         int *new_data = get_heap_memory(new_capacity);
39)
40)         for (int i = 0; i < stack->length; ++i) {
41)             new_data[i] = stack->data[i];
42)         }
43)
44)         free(stack->data);
45)         stack->data     = new_data;
46)         stack->capacity = new_capacity;
47)     }
48)
49)     stack->data[stack->length] = n;
50)     stack->length += 1;
51)}
52)
53)int main() {
54)     Stack *stack = stack_new();
55)     stack_push(stack, 107);
56)
57)     int *n = &stack->data[0];
58)     stack_push(stack, 110);
59)     printf("%d\n", *n);
60)
61)     free(stack);
62)}
```

错误：

(1) _____

(2) _____

(3) _____

(4) _____

学完《网络编程》一章，小明迫不及待地想搭建自己的游戏服务器。他根据课本上代码写出的《网络乒乓》服务器代码如下。他在Class Machine上运行代码，TCP等协议都采用默认设置。请你根据代码回答问题。

```c
#include "csapp.h"
#include "stdlib.h"
#include "stdio.h"
#define GOAL 100

int main(int argc, char **argv)
{
    int listenfd, connfd;
    socklen_t clientlen;
    struct sockaddr_storage clientaddr;
    char client_hostname[MAXLINE], client_port[MAXLINE], buf[MAXLINE];
    if (argc != 2) {
        fprintf(stderr, "usage: %s <port>\n", argv[0]);
        exit(0);
    }
    listenfd = Open_listenfd(argv[1]); // Open listen socket
    int ball_cnt = 0;
    while (1){
        clientlen = sizeof(struct sockaddr_storage);
        connfd = Accept(listenfd, (SA *)&clientaddr, &clientlen); // Accept connection
        Getnameinfo((SA *)&clientaddr, clientlen, client_hostname, MAXLINE,
        client_port, MAXLINE, 0);
        printf("Connected to (%s, %s)\n", client_hostname, client_port);

        rio_t rio;
        Rio_readinitb(&rio, connfd);
        Rio_readlineb(&rio, buf, MAXLINE);
        int recv_num = atoi(buf);
        if (recv_num == ball_cnt){
            if (ball_cnt == GOAL){
                sprintf(buf, "You win!\n");
                Rio_writen(connfd, buf, strlen(buf));
                ball_cnt = 0;
            }
            else {
                sprintf(buf, "%d\n", ball_cnt+1);
                Rio_writen(connfd, buf, strlen(buf));
                ball_cnt += 2;
            }
        }
        Close(connfd); // Close the connection
    }
    exit(0);
}
```

1. 函数`Open_listenfd`要建立的是可靠的TCP连接。因此，里面有一行代码设置了`hints.ai_socktype = _____;`（`SOCK_STREAM` / `SOCK_DGRAM`）。（1分）
2. 函数`Getnameinfo`的用途是查询输入的_____（主机名对应的IP地址 / IP地址对应的主机名）。（1分）
3. 小明的朋友们对他的游戏都不感兴趣，小明只好自己写了一个客户端来玩自己的游戏。他的客户端代码如下：

```c
#include "csapp.h"
#define GOAL 100

int main(int argc, char **argv){
    int clientfd;
    char *host, *port, buf[MAXLINE];
    rio_t rio;

    if (argc != 3){
        fprintf(stderr, "usage: %s <host> <port>\n", argv[0]);
        exit(0);
    }
    host = argv[1];
    port = argv[2];

    for (int ball_cnt=0; ball_cnt<=GOAL; ball_cnt+=2){
        clientfd = Open_clientfd(host, port);
        Rio_readinitb(&rio, clientfd);
        sprintf(buf, "%d\n", ball_cnt);
        Rio_writen(clientfd, buf, strlen(buf));
        Rio_readlineb(&rio, buf, MAXLINE);
        Fputs(buf, stdout);
    }
    exit(0);
}
```

他在同一台主机上运行服务器和客户端。他的操作流程是：

```
cd pingpong
nohup ./server 50000 > log.out &
./client _____ _____
```

程序正常运行完毕，“You win！”出现在了他的屏幕上。小明查看log.out，里面的第一行是`Connected to (localhost, 33580)`。请填入小明操作流程中缺失的两处信息。（2分）

4. 小明认为设置`#define GOAL 100`难度太低了，于是将服务器与客户端中的代码都改成了`#define GOAL 100000`。他像之前一样运行服务器与客户端，结果发现客户端开始报错，并且始终没有输出“You win”。请你解释为什么修改成100000会让程序不能正常运行。（2分）

5. 请在客户端中添加一行代码，使得 `#define GOAL 100000` 时也能正常运行。你添加的位置是现在的第_____行之后，添加的代码内容是_____。（2分）
6. 小红和小方听说了小明的高难度游戏，非常感兴趣。在小明合理配置服务器使得服务器能被访问到后，小红和小方同时在各自的电脑上运行客户端程序，与小明的服务器互动。假设小红、小方的电脑配置相同，网络延迟大小相同，并且整个过程中没有发生过重传。请推测，一段时间后，小红和小方：（）（2分）

- A. 有且只有一个人的屏幕上会出现“You win!”。
- B. 两个人的屏幕上都不会出现“You win!”。
- C. 两个人的屏幕上都会出现“You win!”。

```
cd pingpong
nohup ./server 50000 > log.out &
./client ______ ______
```

程序正常运行完毕，“You win！”出现在了他的屏幕上。小明查看 log.out，里面的第一行是`Connected to (localhost, 33580)`。请填入小明操作流程中缺失的两处信息。（2 分）

4、小明认为设置`#define GOAL 100`难度太低了，于是将服务器与客户端中的代码都改成了`#define GOAL 100000`。他像之前一样运行服务器与客户端，结果发现客户端开始报错，并且始终没有输出“You win”。请你解释为什么修改成 100000 会让程序不能正常运行。（2 分）

5、请在客户端中添加一行代码，使得`#define GOAL 100000`时也能正常运行。你添加的位置是现在的第_____行之后，添加的代码内容是_____。（2 分）

6、小红和小方听说了小明的高难度游戏，非常感兴趣。在小明合理配置服务器使得服务器能被访问到后，小红和小方同时在各自的电脑上运行客户端程序，与小明的服务器互动。假设小红、小方的电脑配置相同，网络延迟大小相同，并且整个过程中没有发生过重传。请推测，一段时间后，小红和小方：（　　）（2 分）

* A. 有且只有一个人的屏幕上会出现“You win!”。
* B. 两个人的屏幕上都不会出现“You win!”。
* C. 两个人的屏幕上都会出现“You win!”。

第七题 并发（15 分）

1、第一类读者-写者问题（读者优先：总是给读者优先权，只要写者当前没有进行写操作，读者就能获得访问权；这种情况存在于读者很多，写者不经常更新的时候使用）

```
int readcnt;    /* Initially 0 */
sem_t mutex, w; /* Both initially 1 */

void reader(void)
{
  while (1) {
    /* 1 */
    readcnt++;
    if (readcnt == 1) /* First in */
        /* 2 */
    /* 3 */

    /* Reading happens here */

    /* 4 */
    readcnt--;
    if (readcnt == 0) /* Last out */
             /* 5 */
    /* 6 */
  }
}

void writer(void)
{
```

```
  while (1) {
    /* 7 */

    /* Writing here */

    /* 8 */
  }
}
```

2、第二类读者-写者问题（写者优先：写者具有优先权，将后来的读者延迟到所有等待的或活动的写者都完成为止；这种情况存在于经常更新的系统，而读者的目的是获取最新的数据）

```
int readcnt, writecnt;      // Initially 0
sem_t rmutex, wmutex;         // Initially 1
sem_t r, w;                       // Initially /* 1 */
void reader(void)
{
  while (1) {
    /* 2 */
    P(&rmutex);
    readcnt++;
    /* 3 */
    V(&rmutex);
    /* 4 */

    /* Reading happens here */

    P(&rmutex);
    readcnt--;
    /* 5 */
    V(&rmutex);
  }
}

void writer(void)
{
  while (1) {
    P(&wmutex);
    writecnt++;
    /* 6 */
    V(&wmutex);

    P(&w);
    /* Writing here */
    V(&w);

    P(&wmutex);
    writecnt--;
    /* 7 */
    V(&wmutex);
  }
}
```