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：2019 年 12 月 30 日**小班号：**__**小班教师：**____

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 16 页。

得分

注：所有回答请填写在答题纸上,并注明大题和小题的题号。

第一题 单项选择题（每小题 2 分，共 40 分）

注：请在答题纸上按照下表样式画出表格并填写答案。

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. 已知浮点运算表达式：t = a + b; x = t + c; y = d + t; 下列表达式中，x, y 不能都得到相同的计算结果的是：

   A. x = a + b + c; y = d + a + b;

   B. x = a + b + c; y = d + (a + b);

   C. x = c + (a + b); y = a + b + d;

   D. x = (a + b) + c; y = d + (a + b);

2. 在本课程的 PIPE 流水线中，下列情况会出现数据冒险的是：

   A. 当前指令会改变下一条指令的目的操作数

   B. 当前指令会改变下一条指令的源操作数

   C. 下一条指令会改变当前指令的目的操作数

   D. 下一条指令会改变当前指令的源操作数

3.
```
int x = -2019;
int a = (x / 8) * 8;
int b = ((x + 7) >> 3) << 3;
int c = -(((~x)+1) >> 3) << 3;
```

   a, b, c 大小关系是什么？

   A. a=b, b=c

   B. a=b, b>c

   C. a<b b<c

   D. a<b, b=c

4. 下述关于 RISC 和 CISC 的讨论，哪个是错误的

   A. RISC 指令集包含的指令数量通常比 CISC 的少

   B. RISC 的寻址方式通常比 CISC 的寻址方式少

   C. RISC 的指令长度通常短于 CISC 的指令长度

   D. 手机处理器通常采用 RISC，而 PC 采用 CISC

5. 下述关于各类存储介质的讨论，哪个是错误的？

A．磁盘访问的延时和数据所在位置有关，连续数据访问的性能高于随机数据访问

B．DRAM 是动态随机存储器，连续数据访问和随机数据访问延时是一样的

C．对 SSD 设备进行数据擦除操作的延时，远高于进行数据读取的延时

D．SRAM 是静态随机存储器，它的存储密度比 DRAM、SSD 都要低

6. 在链接时，对于下列哪些符号需要进行重定位？

(1) 不同 C 语言源文件中定义的函数

(2) 同一 C 语言源文件中定义的全局变量

(3) 同一函数中定义时不带 static 的变量

(4) 同一函数中定义时带有 static 的变量

A．(1)(3)　　B．(2)(4)　　C．(1)(2)(4)　　D．(1)(2)(3)(4)

7. 关于 x86-64 系统中的异常，下面那个判断是正确的：

A．除法错误是异步异常，Unix 会终止程序；

B．键盘输入中断是异步异常，异常服务后会返回当前指令执行；

C．缺页是同步异常，异常服务后会返回当前指令执行；

D．时间片到时中断是同步异常，异常服务后会返回下一条指令执行；

8. 考虑 4 个具有如下开始和结束时间并运行在不同处理器上的进程，

进程	开始时间	结束时间	运行处理器
A	5	7	P0
B	2	4	P1
C	3	6	P0
D	1	8	P1

下面那个判断是正确的。

A．进程 A 和 B 不是并发运行的，但是并行运行的；

B．进程 A 和 C 是并发运行的，并且是并行运行的；

C．进程 B 和 D 是并发运行的，但不是并行运行的；

D．进程 A 和 D 是并发运行的，并且是并行运行的；

9. 进程管理相关函数的调用和返回行为，下列那些函数都是可能返回多于一次的？

A．longjmp 和 fork

B．execve 和 longjmp

C．fork 和 setjmp

D．setjmp 和 execve

10. 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

11. 已知某系统页面长 2KB，页表项 8 字节，采用多层分页策略映射 48 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

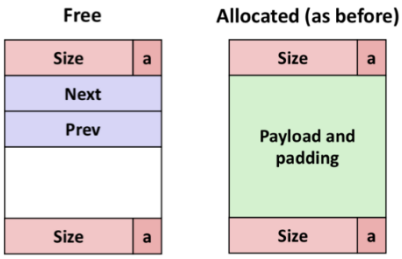
B. 4 层

C. 5 层

D. 6 层

12. 现在有一个用户程序执行了如下调用序列

```
void *p1 = malloc(16);
void *p2 = malloc(32);
void *p3 = malloc(32);
void *p4 = malloc(48);
free(p2);
void *p5 = malloc(4);
free(p3);
void *p6 = malloc(56);
void *p7 = malloc(10);
```



内存分配器内部使用**显式空闲链表**实现，按照地址从低到高顺序来维护空闲链表节点顺序。分配块格式参照上图。每个分配块 16 字节对齐，头部和脚部的大小都是 4 字节。分配算法采用**首次适配**算法，将适配到的空闲块的第一部分作为分配块，剩余部分变成新的空闲块，并采用**立即合并**策略。假设初始空闲块的大小是 1KB。那么以上调用序列完成后，分配器管理的这 1KB 内存区域中，**内部碎片**的总大小是____，链表里**第一个**空闲块的大小是_____。

A. 58Byte，848Byte

B. 26Byte，32Byte

C. 74Byte，48Byte

D. 74Byte，16Byte

13. 在一个具有 TLB 和高速缓存的系统中，假设地址翻译使用四级页表来进行，且不发生缺页异常，那么在 CPU 访问某个虚拟内存地址的过程中，至少会访问（ ）次物理内存，至多会访问（ ）次物理内存。

A. 0，4

B. 0，5

C. 1，4

D. 1，5

14. 采用迭代的 Echo 服务器服务多个客户端，假设 client1 先完成连接，并且没有结束，那么 client2 向服务器发出连接请求后，会被阻塞在下列哪个函数上？

A. connect

B. read

C. write

D. socket

15. 下列关于全球 IP 因特网的各种概念叙述中，哪一个是错误的？

A. 一个套接字是连接的一个端点，对应了一个套接字地址

B. IPv4 的因特网中，一个 IP 地址是一个 32 位无符号整数

C. 因特网主机的字节顺序与 TCP/IP 定义的网络字节顺序是一致的

D. IP 协议提供了一种不可靠的从一台主机向其他主机的递送机制

16. 下列关于 Web 服务器向客户端提供动态内容（遵从 CGI 标准）的叙述中，哪一个是错误的？

A. 客户端提出的 GET 请求的参数在 URI 中传递，并用“&”字符分隔文件名和多个参数

B. 接到客户端的请求，Web 服务器创建一个子进程执行相应的 CGI 程序

C. 通过设置环境变量，Web 服务器将 URI 中传递的参数传递给子进程

D. CGI 程序将动态内容重定向到与客户端关联的已连接描述符 connfd

17. 并发程序执行时会遇到下列各种情况：

① 程序执行结果的正确性依赖于一个线程要在另一个线程到达 y 点之前到达其控制流的 x 点

② 程序员使用 P 和 V 操作顺序不当

③ 外部事件和/或系统调度决策阻止了任务进度

④ 不合理的资源分配策略导致程序难以向前运行

⑤ 程序的执行结果取决于系统的调度决策

⑥ 在信号处理函数中调用 printf()

请问下列哪些情况表示可能产生死锁现象？

A. ①、②和⑤

B. ②、③和④

C. ③、④和⑥

D. ②、④和⑥

18. 称一个函数为线程安全的（thread-safe），当且仅当该函数被多个并发线程反复调用时会一直产生正确的结果。下列哪些函数不是线程安全的？

A. 利用 P、V 操作保护了共享变量的函数

B. 保持跨越多个调用的状态的函数

C. 包括了调用者传递存放结果变量的地址的函数

D. 可重入函数

19. 在以下基于线程的并发 Echo 服务器代码中，哪一个代码片段会潜在引发不正确的程序执行结果？

A.
```
int main(int argc, char **argv)
{                                /*假设有配对的”}”
    int listenfd, connfd;
```

```
        socklen_t clientlen;
        struct sockaddr_storage clientaddr;
        pthread_t tid;
```

B.
```
listenfd = Open_listenfd(argv[1]);
```

C.
```
while (1) {
    connfd = Accept(listenfd, (SA *) &clientaddr,
&clientlen);
    Pthread_create(&tid, NULL, thread, &connfd);
}
```

D.
```
void *thread(void *vargp)
{
  int connfd = *((int *)vargp);
  Pthread_detach(pthread_self());
  echo(connfd);
  Close(connfd);
  return NULL;
}
```

20. 下列关于信号量及 P、V 操作的叙述中，哪一个是不正确的？

A. 信号量是一个非负整数，对信号量只能执行 P 操作和 V 操作

B. 当信号量的值为 0 时，调用 P 操作的线程被挂起

C. P(s)的实现代码中语句：`while (s == 0) wait(); s--;` 应该由内核保证其执行的不可分割性

D. 在保护临界区的解决方案中使用 `sem_wait()`和 `sem_post()`比使用 `pthread_mutex_lock()`和 `pthread_mutex_unlock()`性能好

得分

第二题（10 分）

现有如下汇编代码段，相关寄存器的初始值也在右侧表格中给出：

```
.L2:
    movq      (%rdi), %rbx
    addq      %rbx, %rax
    addq      $8, %rdi
    addq      $1, %rcx
    cmpq      $8, %rcx
    jl        .L2
```

%rdi	0x8008
%rax	0
%rbx	0
%rcx	0

（1） 请在下面 C 代码中，补充缺失的部分。

```
for(i=0; i<_____ ; i++)
{
    sum=sum+a[i];
}
```

（2） 现有一个类似课上 SEQ 的处理器来运行上述汇编代码，在理想情况下，其每个周期可以处理完一条汇编代码，请计算：理想情况下，执行完上述代码段（即结束循环）一共需要多少周期？

（3） 现在考虑数据缓存（data cache）对该处理器的影响。

该处理器配置了一个简单的直接映射（direct-mapped，E=1）数据缓存，总容量 32KB，每个缓存块的大小是 64Byte。对于 movq 指令，如果被访问的数据在该缓存中，则 1 个周期可以处理完成。否则，则需要 2 个周期才能完成。在运行上述汇编代码之前，数据缓存初始状态为空，其他设置同上一问。请再次计算：完成上述汇编代码段一共需要多少周期？

（4） 为了提升处理器的性能，增加了一个加法器。因此，在**没有数据依赖**、并且**不改变指令顺序**的情况下，新的处理器可以同时处理两个 addq 指令、或者一个 addq 加一个 compq 指令。注意：movq 和跳转指令仍旧需要单独执行，其他设置同上一问。请再次计算：完成上述汇编代码段一共需要多少周期？

（5） 为了进一步减少处理时间，程序员将代码进行循环展开，新的代码如下：

```
.L2:
    movq      (%rdi), %rbx
    movq      8 (%rdi), %rdx
    addq      %rbx, %rax
```

```
addq      %rdx, %rax
addq      $16, %rdi
addq      $2, %rcx
cmpq      $8, %rcx
jl        .L2
```

假设在不影响程序正确性的情况下，可以调整上述指令的顺序，请问最短需要多少个周期可以完成该代码段（注：其他设置和上一问相同）。

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，编译和运行在 x86-64/Linux 下完成，编译过程未加优化选项。

```
//m.c
void myswap();
char buf[2] = {1, 2};
void *bufp0;
void *bufp1;
char temp;
int main(){
......//略去题目无关代码
myswap(temp);
......//略去题目无关代码
   return 0;
}
```

```
//foo.c
extern int buf[];
char *bufp0;
......//略去题目无关代码
char *bufp1 = &buf[1];
void myswap(char temp){
static int count = 0;
......//略去题目无关代码
bufp0 = &buf[0];
temp = *bufp0;
......//略去题目无关代码
*bufp0 = *bufp1;
*bufp1 = temp;
......//略去题目无关代码
count++;
}
```

（1） 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，则请指出在模块 foo.o 中的符号类型（局部、全局或外部）、它在 foo.o 模块中所处的节（.data、.text、.bss 或 COMMON）以及其强弱信息（强、弱、非强非弱）；如果不存在条目，则请将该行后继空白处标记为“/”（请将下表画到答题纸上）。

符号	.symtab 条目?	符号类型	节	强弱
bufp1	是	全局	.data	强
buf				
bufp0				
temp				
count				

（2） 我们使用 REF(myswap.m)->DEF(myswap.foo)表示链接器将把模块 m 中对符号 myswap 的任意引用与模块 foo 中对 swap 的定义关联起来。对于下面的示例，用这种方法来说明链接器如何解析每个模块对多重定义符号的引用。如果有链接时错误，请标记“错误”。如果链接器从定义中任意选择一个，请标记“未知”。

a) REF(bufp0.m) ->DEF(_____)

b) REF(bufp1.m) ->DEF(_____)

（3） 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
48 89 e5            mov    %rsp,%rbp
48 83 ec 10         sub    $0x10,%rsp
......
b8 00 00 00 00      mov    $0x0,%eax
e8 00 00 00 00       callq  1a <main+0x1a>
①
......
c9                  leaveq
c3                  req
```

```
    0000000000......<myswap>:
    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 mov    0x0(%rip),%rax
④
0f b6 00             movzbl (%rax),%eax
88 45 fc             mov    %al,-0x4(%rbp)
......
48 8b 05 00 00 00 00 mov    0x0(%rip),%rax
⑤
48 8b 15 00 00 00 00 mov    0x0(%rip),%rdx
⑥
0f b6 12             movzbl (%rdx),%edx
88 10                mov    %dl,(%rax)
48 8b 05 00 00 00 00 mov    0x0(%rip),%rax
⑦
0f b6 55 fc          movzbl -0x4(%rbp),%edx
88 10                mov    %dl,(%rax)
......
8b 05 00 00 00 00    mov    0x0(%rip),%eax
⑧
83 c0 01             add    $0x1,%eax
89 05 00 00 00 00    mov    %eax,0x0(%rip)
⑨
90                   nop
5d                   pop    %rbp
c3                   retq
```

```
0000000000000f28 <main>:
f28:  55                  push   %rbp
f29:  48 89 e5            mov    %rsp,%rbp
f2c:  48 83 ec 10         sub    $0x10,%rsp
......略去部分和答题无关的信息......
f38:  b8 00 00 00 00      mov    $0x0,%eax
f3d:   e8 _____①_____      callq  1000
<myswap>
......略去部分和答题无关的信息......
ffd:  c9                   leaveq
ffe:  c3                   retq
......略去部分和答题无关的信息......
0000000000001000 <myswap>:
1000:  55                  push   %rbp
1001:  48 89 e5            mov    %rsp,%rbp
......略去部分和答题无关的信息......
100d:  48 c7 05 ?? ?? ?? ?? ?? ?? ?? ??
movq    ②  ,0x????(%rip)
1018:                        48            8b
05 ?? ?? ?? ?? mov    0x????(%rip),%rax
101f:  0f b6 00             movzbl (%rax),%eax
1018:  88 45 fc                mov    %eax,-
0x4(%rbp)
......略去部分和答题无关的信息......
1035:     48   8b   05   ??   ??   ??   ??   mov
0x????(%rip),%rax
103c:   48  8b  15  ??  ??  ??  ??  mov      ⑥
(%rip),%rdx
1043:   0f  b6  12                        movzbl
(%rdx),%edx
1046:  88 10                mov    %dl,(%rax)
1048:     48   8b   05   ??   ??   ??   ??   mov
0x????(%rip),%rax
104f:   0f  b6  55  fc                movzbl -
0x4(%rbp),%edx
1053:  88 10                mov    %dl,(%rax)
......略去部分和答题无关的信息......
1057:    8b  05  ??  ??  ??  ??           mov
0x????(%rip),%eax
105d:  83 c0 01             add    $0x1,%eax
1060:  89 05 ?? ?? ?? ??    mov    %eax,  ⑨
(%rip)
1066:  90                   nop
1067:  5d                   pop    %rbp
1068:  c3                   retq
......略去部分和答题无关的信息......
000000000000a000 <buf>:
......略去部分和答题无关的信息......
000000000000a050 <bufp1>:
......略去部分和答题无关的信息......
000000000000cb24 <count.1837>:
......略去部分和答题无关的信息......
000000000000cb28 <bufp0>:
......略去部分和答题无关的信息......
```

在上图中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。（请将下表的第 1 列和第 3 列画到答题纸上）

编号	重定位条目信息		应填入的重定位引用值
①	r.offset = 0x16 r.type = R_X86_64_PC32	r.symbol = 本题不提供 r.addend = -4	
②	r.offset = 0x14 r.type = R_X86_64_32	r.symbol = buf r.addend = +0	
⑥	r.offset = 0x3f r.type = R_X86_64_PC32	r.symbol = bufp1 r.addend = -4	
⑨	r.offset = 0x62 r.type = R_X86_64_PC32	r.symbol = 本题不提供 r.addend = -4	

得分

第四题（10 分）

分析以下C程序，其中f1.txt和f2.txt为已有用户有读写的文件，初始文件内容为空。

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <sys/types.h>
4.  #include <sys/stat.h>
5.  #include <fcntl.h>
6.  #include <unistd.h>
7.  #include <sys/types.h>
8.  #include <sys/wait.h>
9.
10. int main()
11. {
12.   int fd1,fd2,fd3,fd4;
13.   int pid;
14.   int c=1;
15.   fd1=open("./f1.txt",O_WRONLY,0);
16.   fd2=open("./f1.txt",O_WRONLY,0);
17.
18.   printf("fd1=%d,fd2=%d;\n",fd1,fd2);
19.
20.   write(fd1,"EECSPKU",7);
21.   write(fd2,"2019",4);
22.
23.   close(fd2);
24.
25.   fd3=open("./f2.txt",O_WRONLY,0);
26.   fd4=dup(fd3);
27.
28.   printf("fd3=%d,fd4=%d;\n",fd3,fd4);
29.
30.   pid=fork();
31.   if((pid==0)) {
32.     c--;
33.     write(fd3,"PKU",3);
34.     write(fd4,"ICS",3);
35.     printf("c= %d\n",c);
36.   }
37.   else {
38.     waitpid(-1,NULL,0);
39.     c++;
40.     write(fd3,"2019",4);
```

```
41.     close(fd3);
42.     close(fd4);
43.     printf("c= %d\n",c);
44.   }
45.
46.   if(c)
47.    write(fd1,"CS",2);
48.   c++;
49.   close(fd1);
50.   printf("c=%d\n",c);
51. }
```

当程序正确运行后，填写输出结果：

（1） 程序第 18 行：fd1=__①__，fd2=__②__；

（2） 程序第 28 行：fd3=__①__，fd4=__②__；

（3） 程序第35、43、50行输出c的值依次分别为：_____；

（4） 文件 f1.txt 中的内容为：_____；

（5） 文件 f2.txt 中的内容为：_____。

得分

第五题（10 分）

某 32 位机器有 24 位地址空间，采用二级页表，一级页表中有 64 个 PTE，二级页表中有 1024 个 PTE，一级页表 4KB 对齐，PTE 最高一位是有效位，没有 TLB 和 Cache。惠楚思程序员在该机器上执行了如下代码。

```
1. int* a=calloc(10000, sizeof(int));
2. int* b = a+5000;
3. fork();
4. *b=0x80C3F110;
```

（1） 假设编译后 b 的值保存在寄存器中，请问该机器页面大小为①__字节，VPN1 长度为②____，VPN2 长度为③____。

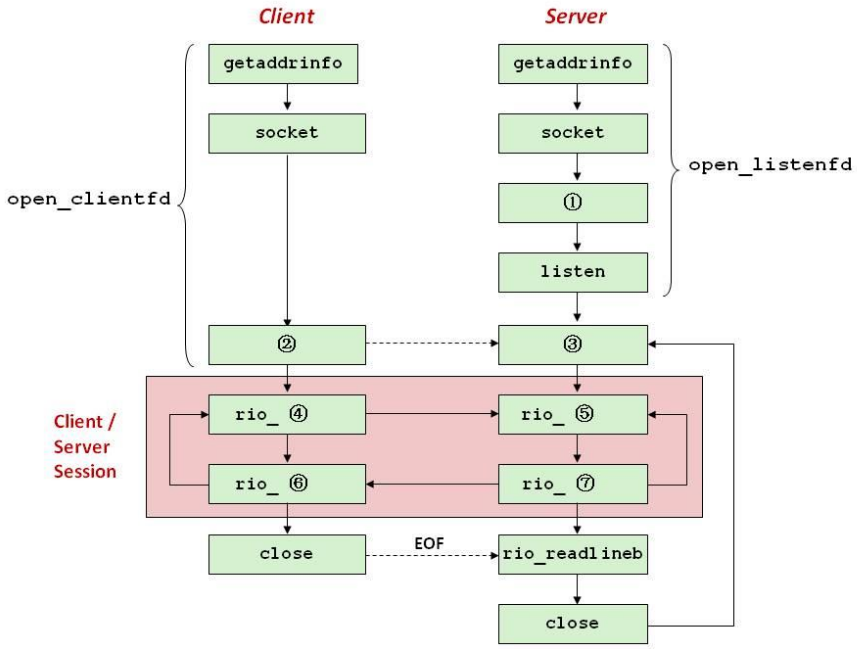
（2） 由于该执行结果不符合预期，惠楚思需要对程序执行过程进行还原。程序执行过程中硬件记录了部分内存访问记录，但访问记录并不完整。目前已知在父进程执行第四行的时候，进行了若干次物理内存的读写操作，其中第一次读内存操作从地址 0x67F0E8 读出 0x80AA32C4，另外有两次将 0x80C3F110 写入内存的操作，写入的地址分别是 0xC3F50D 和 0xAA3AD0，但写入顺序并不清楚。程序结束后变量 b 中保存的值是①_____。在解析“*b”的过程中，一级页表的起始地址为②_____；二级页表的起始地址为③_____；物理页面的起始地址为④_____。（地址均用 16 进制表示）

得分

第六题（10 分）

下图是一个基于 echo 服务器的 client-server 框架

（1） 请给图中的编号填写相应的函数名。



（2） 请补全下面 server 端 open_listenfd 函数中缺失的操作（Line 21, Line 26 和 Line 34）

```
Line 1:  int open_listenfd(char *port)
Line 2:  {
Line 3:     struct addrinfo hints, *listp, *p;
Line 4:     int listenfd, optval=1;
Line 5:     /* Get a list of potential server addresses */
Line 6:     memset(&hints, 0, sizeof(struct addrinfo));
Line 7:     hints.ai_socktype = SOCK_STREAM;
Line 8:     hints.ai_flags = AI_PASSIVE | AI_ADDRCONFIG;
Line 9:     hints.ai_flags |= AI_NUMERICSERV;
Line 10:    Getaddrinfo(NULL, port, &hints, &listp);
```

```
Line 11:
Line 12:    for (p = listp; p; p = p->ai_next) {
Line 13:       /* Create a socket descriptor */
Line 14:      if ((listenfd = socket(p->ai_family, p->ai_socktype,
Line 15:                             p->ai_protocol)) < 0)
Line 16:        continue;  /* Socket failed, try the next */
Line 17:       /* Eliminates "Address already in use" error from
bind */
Line 18:       Setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR,
Line 19:                (const void *)&optval , sizeof(int));
Line 20:
Line 21:       if (_⑧_(listenfd, p->ai_addr, p->ai_addrlen) == 0)
Line 22:         break;  /* Success */
Line 23:       Close(listenfd);
Line 24:    }
Line 25:    /* Clean up */
Line 26:    Freeaddrinfo(___⑨___);
Line 27:    if (!p) /* No address worked */
Line 28:       return -1;
Line 29:
Line 30:    if (listen(listenfd, LISTENQ) < 0) {
Line 31:       Close(listenfd);
Line 32:       return -1;
Line 33:    }
Line 34:    return __⑩___;
Line 35: }
```

得分

第七题（10 分）

解决第一类读者-写者问题的代码如下：

```
int readcnt;    /* Initially 0 */
sem_t mutex, w; /* Both initially 1 */
void reader(void)
{
(1)    while (1) {
(2)      P(&mutex);
(3)      readcnt++;
(4)      if (readcnt == 1) /* First in */
(5)        P(&w);
(6)      V(&mutex);
(7)         /* Reading happens here, need 7 time unit */
(8)      P(&mutex);
(9)      readcnt--;
(10)     if (readcnt == 0) /* Last out */
(11)        V(&w);
(12)     V(&mutex);
       }
}

void writer(void)
{
(13)  while (1) {
(14)    P(&w);
(15)       /* Writing here, need 8 time unit */
(16)    V(&w);
     }
}
```

假设有 5 个读者或写者到来，到达时刻和所需的时间如下：

线程	到达时刻	读写时间
R1	0	7
W1	1	8
R2	2	7
W2	4	8
R3	5	7

注意，为简单起见，只考虑读写时间，忽略所有其他时间（包括代码中其他语句的执行时间、线程切换、调度等等），亦假设没有更多的读者或写者或其他线程。

（1）在时刻 3 时，R1、W1 和 R2 所处的位置，请标出对应的代码行号。此刻，readcnt 和 w 的值分别是多少？R1:_；W1:_；R2:_；readcnt:_；w:_。

（2）在时刻 6 时，W2 和 R3 所处的位置，请标出对应的行号。此刻，readcnt 和 w 的值分别是多少？W2:_；R3:_；readcnt:_；w:_。

（3）如果不使用 P(&mutex)和 V(&mutex)，程序执行会不会出错？如果出错，会出什么错？