2024“概论”课练习

1.（X、Y 的数据宽度均为 16 位，计算结果用 16 进制的补码表示）已知$[X]_{补码}$ = 0x0029，$[Y]_{补码}$ = 0xFE6A，则$[X+Y]_{补码}$ = （ 0xFE93 ），$[X-Y]_{补码}$ = （ 0x1BF ）。

2. 寄存器 EAX,EBX 内存储的为带符号 32 位整数，若%EAX > %EBX > 0（做为带符号数的比较结果），则指令 cmpl %EAX, %EBX 执行后条件码中的符号标志位 SF=（ 1 ），溢出标志位 OF=（ 0 ）。（可以填“不确定”）

3. 在所有由五个“1”和三个“0”组成的 8 位二进制整数（补码形式）中，最小的数是（ -113 ），最大的数为（ 124 ）；请用十进制表示结果。

4. X86-64 位 Linux 系统下的 float 类型的数据对齐要求是（ 4 ）字节对齐，double 类型的是（ 8 ）字节对齐。

5. 假设存在一种 8 位浮点数（符合 IEEE 浮点数标准），1 个符号位，3 位阶码，4 位尾数。其数值被表示为 $V = (-1)^S \times M \times 2^E$形式。请在下表中填空。
Binary: 这一列请填入 **8 位二进制表示**；M: **十进制数表示**；E: **十进制整数表示**；
Value: 被表示的具体数值，**十进制表示**；“—”表示无需填入。

描述	Binary	M	E	Value
负 0	1 000 0000	0	**-2**	-0.0
正无穷	01110000	—	—	+ ∞
—	0 110 0110	1.375	3	11
最小的正规格化浮点数	0 001 0000	1	-2	0.25

6. 假设存在一种 6 位带符号整数（补码表示，规律与我们课堂上学的一样，只是位数有差别）。请填写下列表格（TMax 表示该类有符号整数的最大值，TMin 为最小值）:

数字描述	十进制表示	二进制补码表示
N/A	-6	111010
N/A	9	001001
TMin + TMin	0	000000

TMax+1	-32	100000

7. 有如下对应的 C 代码与汇编代码（x86-64），请对照着填上代码中缺失的部分。

```
struct matrix_entry{
    char a;
    char b;
    double d;
    int c;
};
struct matrix_entry matrix[5][①];
int return_entry(int i, int j){
        return matrix[i][j].c;
}
```

①（ 7 ）②（840 ）③（ 16 ）④（ %rdx ）

```
return_entry:
        movslq   %esi, %rsi
        movslq   %edi, %rdi
        leaq     (%rsi,%rsi,2), %rax
        leaq     0(,%rax,8), ④
        leaq     (%rdi,%rdi,4), %rax
        leaq     (%rdi,%rax,4), %rcx
        leaq     0(,%rcx,8), %rax
        movl     matrix+③(%rdx,%rax), %eax
        ret
        .comm    matrix, ②, #.comm 后的第二个参数
                 #表示变量 matrix 占据空间的大小，
                 #以字节为单位
```

8. 有如下对应的 C 代码与汇编代码（x86-64），请对照着填上代码中缺失的部分（数字请用十进制表示）

```
int VVV(int n, int m)
{
    int matrix1[n][m];
    return call_val(n, m, matrix1);
}
```

```
VVV:
        pushq    %rbp
        movq     %rsp, %rbp
        movslq   %esi, %rdx
        movslq   %edi, %rax
        ①        %rdx, %rax
        leaq     18(,%rax,4), %rax
        andq     $-16, %rax
        subq     %rax, ②
        movq     ③, ④
        call     call_val
        leave    # 这条指令等同于 movq %rbp, %rsp;   popq %rbp
        ret
```

①（ imulq ）②（ %rsp ）③（%rsp ）④（ %rdx）

9. 编程解决猴子吃桃问题：每天吃一半再多吃一个，第十天想吃时候只剩一个，问总共有多少。其对应的 C 语言程序如下左侧代码所示。

请在其对应的汇编代码（Linux X86-64）内填入缺失的内容（可以看到编译将递归优化掉了）。

```
int chitao(int i){
   if(i == 10){
      return 1;
   }else{
      return (chitao(i + 1) + 1) * 2;
   }
}
```

```
chitao:
        cmpl    $10, %edi
        movl    $1, ①
        je      .L4
        xorl    ②, %edx
.L3:
        addl    %eax, ③ //2
        addl    ④, %edx   //2
        addl    $1, %edi
        cmpl    $10, %edi
        jne     .L3
        addl    ⑤, %eax    //3
        ret
.L4:
        ret
```

①（ %eax ）②（ %edx ）③（ %eax ）④（ %eax ）⑤（ %edx ）

10. 在 x86-64 Linu 中的某些编译选项下，每个 C 函数对应的汇编代码的开头是 pushq %rbp; movq%rsp, %rbp 来维护栈指针，函数结尾是 movq %rbp, %rsp; popq %rbp; ret 指令来恢复栈指针并返回到调用者，并且函数运行过程中不会修改 rbp 寄存器。基于这种性质，可以在栈上面寻找各级调用者的返回地址和栈地址，按照这种思路实现了下列的 backtrace 函数。

```
1 backtrace:
2     pushq %rbp
3     movq %rsp, %rbp
4     pushq %rbx
5     subq $8, %rsp
6     movq %rbp, %rbx
7 .LOOP:
8     movq 8(%rbx), %rdi
9     movq (%rbx), %rbx
10    movq %rbx, %rsi
11    call print_stackframe
12    testq %rbx, %rbx
13    jne   .LOOP
14    movq -8(%rbp), %rbx
15    leave
16    ret
```

(1) backtrace 函数按照栈地址 _____ (从低到高/从高到低) 的顺序遍历各个调用者，代码中用 push 指令把寄存器 _____ 保存在栈上并在函数退出之前用 movq 指令恢复，这属于调用约定中 _____ (caller-saved/callee-saved)的寄存器。

(2) （单选）其中 print_stackframe 函数接收两个参数，它会打印出当前调用者的信息：其中第一个参数是该调用者的 _____，通过寄存器 _____传递；第二个参数是该

调用者的 _____，通过寄存器 _____ 传递。

A. rax **B.** rbp **C.**rbx **D.**rdi **E.**rsi **F.**返回地址

(3) 描述 backtrace 函数实现的原理，并画出 backtrace 执行过程中遍历的栈的示意图。

(1) 从低到高； rbx； callee-saved

(2) F； D； B； E

(3) 从当前的 rbp 开始，找到上一层调用者的 rbp 和返回地址，然后循环打出每一个调用者的返回地址和 rbp。画一个栈的示意图，要表示出返回地址和 rbp 的关系。

11. 有如下的 C 代码结构定义与函数。在 Linux X86-32 下，相应的汇编代码如右框图所示。请补齐代码。

```
typedef struct node {
    ____①____ x;
    ____②____ y;
    struct node *next;
    struct node *prev;
} node_t;
node_t N;

void func() {
  node_t *m;
  m = ____③____;
  m->y /= ____④____;
  return;
}
```

```
func:
        movl    N+16, %ecx
        movzbl  8(%ecx), %edx
        leal    7(%edx), %eax
        testb   %dl, %dl
        cmovns  %edx, %eax
        sarb    $3, %al
        movb    %al, 8(%ecx)
        ret
```

并已知 Linux X86-32 下各种基本数据类型的大小与对齐要求（下表），上面结构定义中缺失的数据类型只能从表中选取。

Type	Size (bytes)	Alignment (bytes)
char	1	1
short	2	2
unsigned short	2	2
int	4	4
unsigned int	4	4
double	8	4

1、double 2、char 3、n.prev 4、8

12. 在 Linux X86-32 架构下，有如下 C 代码及对应的汇编语言，请填充 C 代码缺失的部分。

```
loop:
    pushl %ebp
    movl %esp,%ebp
    movl 0x8(%ebp),%edx
    movl %edx,%eax
    addl 0xc(%ebp),%eax
    leal 0xffffffff(%eax),%ecx
    cmpl %ecx,%edx
    jae .L4
.L6:
    movb (%edx),%al
    xorb (%ecx),%al
    movb %al,(%edx)
    xorb (%ecx),%al
    movb %al,(%ecx)
    xorb %al,(%edx)
    incl %edx
    decl %ecx
    cmpl %ecx,%edx
    jb .L6
.L4:
    movl %ebp,%esp
    popl %ebp
    ret
```

```
void loop(char *h, int len)
{
    char *t;

    for (_____________; _____________; h++,t--) {

        _____________;

        _____________;

        _____________;
    }

    return;
}
```

```
void loop(char *h, int len){
   char *t;
   for  (t=h+len-1;t>h; h++,t--)
   {
       h[0] = h[0]^t[0];
       t[0] = h[0]^t[0];
       h[0] = h[0]^t[0];
   }
}
```

13. 有如下 3 个函数，请解释为什么函数 f1 的汇编代码中没有出现 call 和 ret 指令，即函数 f1 是如何调用函数 f2/f3 又是如何返回的？画出运行栈的 layout 示意图并辅以说明。

```
int f2(int a){
    return a*20;
}
int f3(int a){
    return a*30;
}
int f1(int a){
    if (a>3)
        return f2(a+a);
    else
        return f3(a+2);
}

Int main()
{
    int a=XXX;
    int b=f1(a);
}
```

```
0000000000400c30 <f1>:
  400c30:    83 ff 03          cmp    $0x3,%edi
  400c33:    7e 07             jle    400c3c <f1+0xc>
  400c35:    03 ff             add    %edi,%edi
  400c37:    e9 14 00 00 00    jmpq   400c50 <f2>
  400c3c:    83 c7 02          add    $0x2,%edi
  400c3f:    e9 1c 00 00 00    jmpq   400c60 <f3>
0000000000400c50 <f2>:
  400c50:    8d 04 bf          lea    (%rdi,%rdi,4),%eax
  400c53:    c1 e0 02          shl    $0x2,%eax
  400c56:    c3                retq
0000000000400c60 <f3>:
  400c60:    89 f8             mov    %edi,%eax
  400c62:    c1 e0 05          shl    $0x5,%eax
  400c65:    2b c7             sub    %edi,%eax
  400c67:    2b c7             sub    %edi,%eax
  400c69:    c3                retq
```

因为f1没有栈帧；图略。

14. 程序运行过程中发生被 0 除异常，这属于__同步__（异步或 同步）异常（exception）；程序运行过程中发生缺页中断，这是__同步__（异步 或 同步）异常（exception）；磁盘操作完成后向处理器发送中断属于__异步__（异步 或 同步）异常（exception）。

15.

```
union {
  bfloat16 f;
  unsigned short s;
}
```

bfloat16 是由谷歌提出的一种半精度浮点数，exp 位数是 8，frac 位数是 7，符号位数为 1；除了位宽度差别外，其它规格符合 IEEE 754 标准。如果存在一个如左所示的 C 语言 union（联合）数据类型（在 x86 机器上，且假设 bfloat16 表示该种半精度浮点数类型），那么当 f 被赋予 bfloat16 所能表示的最接近于 1 的数（且大于 1）后，s 的值为（ ），请用 16 进制表示。

0011 1111 1000 0001= 0x3f81

16. 读以下程序，并填空。

（1）下方程序输出（ 8 ）行。

```
void doit() {
 fork();
 fork();
 printf("hello\n");
 return;
}
```

```
int main() {
 doit();
 printf("hello\n");
 exit(0);
}
```

（2）下方程序输出（ 3 ）行。

```
void doit() {
 if (fork() == 0) {
   fork();printf("hello\n");  exit(0);
  }
  return;
}
```

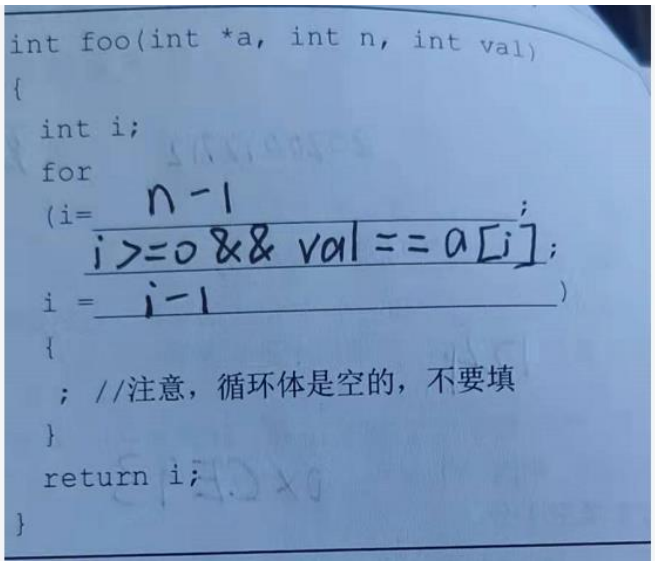
```
int main() {
 doit();
 printf("hello\n");
 exit(0);
}
```

（16.3）右侧程序输出的 counter 值是（ 2）。

```
int counter = 1;
int main()
{
  if (fork() == 0) {
    counter--;
    exit(0);
  }
  else {
    wait(NULL);
    counter++;
    printf("counter=%d\n",counter);
  }
  exit(0);
}
```

17. 读下面左侧的 X86-32 汇编程序，并填空与之对应的右侧 C 程序（共 6 分）。

```
foo:
  pushl %ebp
  movl %esp,%ebp
  movl 8(%ebp),%ecx
  movl 16(%ebp),%edx
  movl 12(%ebp),%eax
  decl %eax
  js .L3
.L7:
  cmpl %edx,(%ecx,%eax,4)
  jne .L3
  decl %eax
  jns .L7
.L3:
  movl %ebp,%esp
  popl %ebp
  ret
```



18. X86-32 机器上一个带符号 32 位整数，其十进制数值为-370，不经过任何转换直接将其通过网络传递到 Sparc 32 位机器上（Big Endian，大端表示），该带符号整数的值变为（ 0x8efeffff ），可以用 16 进制表示。

19. （下面的源代码用于跟踪家庭经营酒店中当前预订的房间情况。residents 数组中的每一元素都存储了预订房间的客户的名称（字符串）。FLOORS 代表酒店的楼层数、ROOMS 代表每层的房间数(这两个都是用#define 声明的常量)；LEN 则被定义为 12。下面的 C 代码就是用来记录预订某层某号房间(假设都是从 0 开始，连续编号)的客户名称。strcpy(char *dest, const char *src)把 src 所指向的字符串复制到 dest 中。对照相应的汇编代码(x86-32)，回答如下问题：

（1） 请问 ROOMS 的值是多少？

5

（2） 由于一个奇怪的错误，程序访问了 *residents[0][1][-2]*。请问实际访问的是哪个元素（将答案表示为整数三元组(-，-，-)；假设 FLOORS 和 ROOMS 都大于 1。

(0 ,0 ,10)

```
char residents[FLOORS][ROOMS][LEN];
void reserve_room
(int floor, int room, char *custname){
  strcpy(residents[floor][room],
         custname);
}
```

```
reserve_room:
  pushl %ebp
  movl %esp,%ebp
  movl 12(%ebp),%eax
  movl 16(%ebp),%edx
  pushl %edx
  movl 8(%ebp),%edx
  sall $4,%edx
  subl 8(%ebp),%edx
  leal (%eax,%eax,2),%eax
  leal
residents(,%eax,4),%eax
  leal (%eax,%edx,4),%edx
  pushl %edx
  call strcpy
  movl %ebp,%esp
  popl %ebp
```

（3）代码做了改进(如下图)，residents 被更改为指向字符串(客户名称)的指针的二维数组。新代码只为实际预订出去的房间分配了存储客户名称的内存；否则，仅存储一个 NULL 指针。简单起见，假设 malloc 不带来额外内存开销（malloc (k)分配长度为 k 字节的内存空间，并返回一个指向这一内存地址的指针；我们假设这个调用不带来额外内存开销，即其开销就是 k 字节）。后来，程序员来检查改进后方案所节省的内存——此时酒店预订率为 20%，**节省了 168 字节**。请问这家旅馆有几层楼？(即 FLOORS 值是什么)

```
char *residents[FLOORS][ROOMS];
void reserve_room(int floor, int room,
                  char  *custname)
{
  residents[floor][room] = malloc(LEN);
  strcpy(residents[floor][room], custname);
}
```

FLOORS = 6

20. 某些计算程序的运行时间特别长（比如长达几周），为了防止其运行过程中因为出现一些意外情况（如计算机掉电、维护等）而前功尽弃，需要周期性的保存进程状态（包括**处理器状态与虚存状态两部分**）到磁盘文件中，这样一旦出问题就可以从最近保存的状态中恢复运行而不是从头开始——确实有这样的软件工具，比如 libckpt 就是较有名的一个。为方便设计实现，我们做了如下假设：

（1）保存状态与恢复运行的计算机是同一台，且系统环境配置、该程序及其进程的所有设置都不变；操作系统内核维护的进程状态不予考虑；文件 IO 等状态不考虑；

（2）假设 Linux 系统提供了系统调用 enum_pages，可以顺序遍历本进程的所有用户空间虚页的属性，包括每一个虚页的起始/终止地址以及页属性，后者包括：是否可以合法访问、目前是否已分配了物理页面(cached or not)、只读或者可写（可写的必然可读）；

（3）所有的用户空间虚页从进程运行开始后其属性就没有变化过（除了“目前是否已分配了物理页面”）。

我们设计了自己的进程状态保存与恢复工具，以一个库文件的形式被目标程序链接。然后，目标进程周期性的调用这个工具提供的 save 函数将**两部分进程状态数据**保存到磁盘文件里；首先是处理器状态，这个与我们平时作业中“‘协程’任务切换”所做的工作类似，即将这一状态保存在进程的某个虚存空间中，虚存地址为 current_ctx；接着保存虚存数据，即通过 enum_pages 遍历本进程的所有用户空间虚页，选择性的保存之。状态恢复时（即重新运行该程序，进入 main 函数后调用这个工具提供的 restore 函数进行状态恢复），则读取保存的用户空间虚页内容，复制到当前进程相应的虚页内；然后再恢复处理器状态。

请回答以下问题：

（1） 保存虚存数据时，选择保存具有哪些（个）属性的虚存页的内容能使得保存的数据量尽量少，并回答为什么；

（2） 发现将状态写入磁盘文件的代码中并没有显式地对 current_ctx 的访问，这是为什么？

（3） 保存虚页内容到磁盘文件所涉及的数据量可能很大，比如几个 G，而保存的时候进程无法执行正常计算任务。为了提升性能（即保存数据时该进程仍然能够继续工作），你想出了一个简单有效的优化方法，其核心是使用了一个我们课上讲过的系统调用，请问这个系统调用是？这一优化方法能够工作的原理是？

（4） 恢复进程状态的时候，你首先进行了足够多次的函数递归调用，然后才开始恢复，请问这么做的理由是？

（1） “保存可写页面”这一个属性就对；或者，可写且可以合法访问；

（2） 因为 current_ctx 肯定是虚存空间（带可写属性）页面的一部分，自然已经由（1）保存了。

（3） fork. 第一、clone 了父进程一模一样的虚存；第二、具有 cow 属性（或者类似的意思）；

（4） 确保栈恢复的时候是从“深”（低地址）恢复到“浅”（高地址），以免覆盖“恢复”函数的栈帧（或者类似意思）。

21. 有如下 C 程序及对应的 X86-64 架构下的汇编代码（执行程序反汇编出来的）

```
#include <stdio.h>
int overflow(void);
int one = 1;

int overflow() {
      char buf[4];
      int val, i=0;
      while(scanf("%x", &val) != EOF)
            buf[i++] = (char)val;
      return 15213;
}
```

```
int main() {
     int val = overflow();
     val += one;
     if (val != 15213)
        printf("Boom!\n");
     else
        printf("????\n");

     exit(0);
}
```

说明：scanf(“%x”, &val)函数从标准输入读取由空格分割的代表一个十六进制整数的字符或字符序列，并将该字符或字符序列转换为一个 32 位 int，将转换结果赋给 val。scanf 返回 1 表示转换成功，返回 EOF（-1）则表示 stdin 已经没有更多的输入序列了（通常输入 ctrl+d 表示 EOF）。例如，在输入字符串“0 a ff”上调用 scanf 四次将有以下结果：

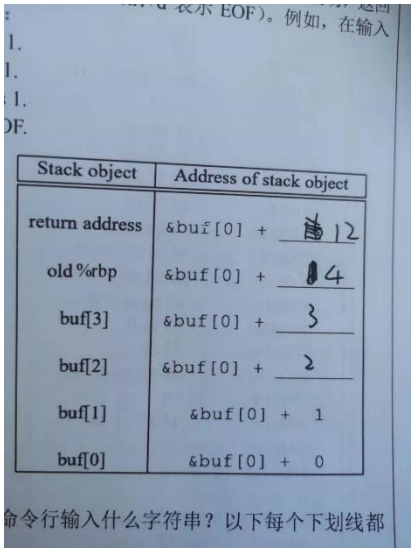
– 1st call to `scanf`: `val=0x0` and `scanf` returns 1.

– 2nd call to `scanf`: `val=0xa` and `scanf` returns 1.

– 3rd call to `scanf`: `val=0xff` and `scanf` returns 1.

– 4th call to `scanf`: `val=??` and `scanf` returns EOF.

(1) 请参照 buf[0]、buf[1]的地址表示方式表示表格内其余对象的地址)。



Stack object	Address of stack object
return address	&buf[0] + 12
old %rbp	&buf[0] + 4
buf[3]	&buf[0] + 3
buf[2]	&buf[0] + 2
buf[1]	&buf[0] + 1
buf[0]	&buf[0] + 0

(2) 为了让这个程序输出“？？？？”，你需要在命令行输入什么字符串？以下每个下划线都是一个 1 位或 2 位的十六进制数字，下划线之间是空格（6 分）。

0 0 0 0 0 0 0 0 0 0 0 0 39 或 26

最后数字是 39 or 26，长度 13；其他随意（不必是 0）

```
00000000004005d6 <overflow>:
   4005d6:        55                      push   %rbp
   4005d7:        48 89 e5                mov    %rsp, %rbp
   4005da:        48 83 ec 28             sub    $0x28, %rsp
   4005db:        53                      push   %rbx
   4005df:        bb 00 00 00 00          mov    $0x0, %ebx
   4005e4:        eb 0d                   jmp    4005f3 <overflow+0x1d>
   4005e6:        48 63 c3                movslq %ebx, %rax
   4005e9:        8b 55 dc                mov    -0x8(%rbp), %edx          #edx = val
   4005ec:        88 54 05 e0             mov    %dl, -0x4(%rbp, %rax, 1) #buf
   4005f0:        8d 5b 01                lea    0x1(%rbx), %ebx     # i++
   4005f3:        48 8d 75 dc             lea    -0x8(%rbp), %rsi
   4005f7:        bf d4 06 40 00          mov    $0x4006d4, %edi    # "%x"地址是 0x4006d4
   4005fc:        b8 00 00 00 00          mov    $0x0, %eax
   400601:        e8 ba fe ff ff          callq  4004c0 <scanf>
   400606:        83 f8 ff                cmp    $0xffffffff, %eax       #EOF 值为-1
   400609:        75 db                   jne    4005e6 <overflow+0x10>
   40060b:        b8 6d 3b 00 00          mov    $0x3b6d, %eax       #0x3b6d=15213
   400610:        5b                      pop    %rbx
   400614:        48 83 c4 28             add    $0x28, %rsp
   400615:        5d                      pop    %rbp
   400616:        c3                      retq

0000000000400617 <main>:
   400617:        55                      push   %rbp
   400618:        48 89 e5                mov    %rsp, %rbp
   40061b:        e8 b6 ff ff ff          callq  4005d6 <overflow>
   400620:        03 05 22 0a 20 00       add    0x200a22(%rip), %eax    # val += one
   400626:        3d 6d 3b 00 00          cmp    $0x3b6d, %eax
   40062b:        74 0c                   je     400639 <main+0x22>
   40062d:        bf d7 06 40 00          mov    $0x4006d7, %edi
   400632:        e8 69 fe ff ff          callq  4004a0 <puts>           #printf("Boom!\n")
   400637:        eb 0a                   jmp    400643 <main+0x2c>
   400639:        bf dd 06 40 00          mov    $0x4006dd, %edi
   40063e:        e8 5d fe ff ff          callq  4004a0 <puts >          #printf("????\n")
   400643:        bf 00 00 00 00          mov    $0x0, %edi
   400648:        e8 43 fe ff ff          callq  400490 <exit >   #exit 不使用 rbp
```

22. 阅读如下代码，请问程序结束后 test.txt 中的字符串长度与内容分别是什么。

a	b	c	d	e	v	w	x	y	z	1	2	3	4	5	

每格填一个字符（注意上面的格子数目与实际的字符串长度无关）。长度 15

```
int main(int argc, char *argv[])
{
    int fd1, fd2, fd3;
    char *fname = argv[1]; //输入文件名为 test.txt
    fd1 = Open(fname, O_CREAT|O_TRUNC|O_RDWR, S_IRUSR|S_IWUSR);
    //上面代码表示创建新文件并打开之（如果文件原来就存在，则清空后打开之）;
    //打开后该文件可读可写。
    Write(fd1, "abcde", 5);
```

```
    fd3 = Open(fname, O_APPEND|O_WRONLY, 0);//以追加写入的方式打开文件，即文件
                                            //每次写入的位置均为文件末尾
    Write(fd3, "fghij", 5);
    fd2 = Open(fname, O_APPEND|O_WRONLY, 0);
    dup2(fd1,fd2);
    Write(fd2, "vwxyz", 5);
    Write(fd3, "ef", 2);
    Write(fd1, "12345",5);
    return 0;
}
```

23. 理发店问题。理发店理有一位理发师、一把理发椅和 N 把供等候理发的顾客坐的椅子。如果没有顾客，理发师便在理发椅上睡觉；一个顾客到来时，他叫醒理发师理发；如果理发师正在理发时又有顾客来到，则如果有空椅子可坐，就坐下来等待，否则就离开理发店。以下是代码的框架，但其中没有加入同步、临界区保护这类的操作。

```
#define N 10
int customers = 0;
void* customer() {//顾客线程
   if (customers > N) {
      return NULL;
   }
   customers ++;
   getHairCut(); //得到服务
   return NULL;
}
void* barber() {//理发师线程
   while(1) {
      customers --;
      cutHair(); //理发师工作
   }
}
```

为解决同步问题，我们使用三个**信号量**（semaphores）。
mutex：控制对全局变量 customers 的访问；　customer：顾客的资源信号量；
barber：理发师的资源信号量。
（1）指出这三个信号量的初始值。
mutex =1　customer=0　barber=0
（2）仅使用以下语句将上述代码补充正确，每个语句不限使用次数。
P(&mutex);　V(&mutex);　P(&customer);　V(&customer);　P(&barber);　V(&barber);

补充后的代码是：

```
void* customer() {
    P(&mutex);
    if (customers > N) {
        V(&mutex);
        return NULL;
    }
    customers ++;
    V(&customer);
    V(&mutex);
    P(&barber);
    getHairCut();
    return NULL;
}
```

```
void* barber() {
    while(1) {
        P(&customer);
        P(&mutex);
        customers --;
        V(&barber);
        V(&mutex);
        cutHair();
    }
}
```

24、给定一个浮点格式，exp 域位宽为 k，frac 域位宽为 n，对于下列数给出其阶码 E、尾数 M、小数 f 和值 V 的公式。并请描述其位表示。

（1）5.0

（2）能够被准确描述的最大奇整数;

（3）最小的正规格化数。

❶ $V = 5.0 = 1.01_{(2)} * 2^2$，$E = 2$，$M = 1.01_2$，$F = 01000\ldots000(n-bits)$。

❷ 由于奇数 + 整数，尾数最后一位“1”的位置即为指数 E。故存在两种情况：“1”的长度为 n；“1”的长度不足 n（由于 exp 无法表示）。
讨论分界点：exp 最大为 $11\ldots110(k-bits)$，$bias = 2^{k-1}-1$，
$E = 2^k - 2 - bias = 2^{k-1}-1$。若 $n < 2^{k-1}-1$，则是第一种情况；若 $n \geq 2^{k-1}-1$，则是第二种情况。

a $M = 1.111\ldots1$，$F = 111\ldots111(n-bits)$，$E = n$，
$V = 111\ldots111_2(n+1-bits) = 2^{n+1}-1$。

b $E = 2^{k-1}-1$，$M = 1.111\ldots111000\ldots$，
$F = 111\ldots111000\ldots$（前 E 位为 1 后续为 0），
$V = 111\ldots111_2(2^{k-1}-bits) = 2^{2^{k-1}}-1$。

❸ $F = 00\ldots000$，$E = 1 - bias = 2 - 2^{k-1}$，$V = 1.0 * 2^E = 2^{2-2^{k-1}}$。

25. X86-64 体系结构中的条件跳转指令 jg 是用于符号数比较还是无符号数比较的？其产生跳转的成立条件是~(SF^OF)&~ZF 为真，请解释为何是这一条件。

（1）有符号跳转（当严格大于的时候跳转）。

（2）先解释 ZF：当 ZF 为 True，则说明相等，不跳转；再解释 SF 和 OF。~(SF ∧ OF) 相当于 SF == OF；分别讨论 SF=OF=0 和 SF=OF=1 的情况。

26. 编程解决猴子吃桃问题：每天吃一半再多吃一个，第十天想吃时候只剩一个，问总共有多少。

（1）C 语言程序如下，请在其对应汇编代码（Linux X86-64）内填入缺失内容。

```
int chitao(int i){
   if(i == 10){
      return 1;
   }else{
      return (chitao(i + 1) + 1) * 2;
   }
}
```

(1) %edi
(2) ret
(3) .L3
(4) %rax
(5) .L2
(6) $8

```
chitao:
        cmpl    $10,_____ ①
        je      _____③
        subq    $8, %rsp
        addl    $1, %edi
        call    chitao
        leal    2(%rax, ____④____), %eax
        jmp     _____⑤_____
.L3:
        movl    $1, %eax
        _____   ②
.L2:
        addq    ____⑥____, %rsp
        ret
```

27. 有如下对应的 C 代码与汇编代码（x86-64），请对照着填上代码中缺失的部分（数字请用十进制表示）

```
call_swap:
        subq    $24, %rsp
        movl    ①, 12(%rsp)
        movl    $91125, 8(%rsp)
        leaq    8(%rsp), %rsi
        leaq    12(%rsp), ④
        movl    $0, %eax
        call    swap
        addq    $24, %rsp
        ret
```

```
void call_swap()
{
  int zip1 = 15213;
  int zip2 = ③;
  return ②;
}
```

(1) $15213　(2) swap(&zip1, &zip2)　(3) 91125　(4) %rdi

28. 请对照下面的 C 语言代码与相应汇编（Linux X86-64），给出 M、N 的值。

```
copy_element:
        movslq  %edi, %rdi
        movslq  %esi, %rsi
        leaq    (%rsi,%rsi,2), %rax
        leaq    (%rsi,%rax,4), %rax
        addq    %rdi, %rax
        movl    mat2(,%rax,4), %edx
        leaq    0(,%rdi,8), %rax
        subq    %rdi, %rax
        addq    %rax, %rsi
        movl    %edx, mat1(,%rsi,4)
        ret
```

```
#define M   ?
#define N   ?
int mat1[M][N];
int mat2[N][M];
int copy_element(int i, int j)
{
      mat1[i][j] = mat2[j][i];
}
```

M=13　N=7

mat[i][j] 按行连续存储

```
for(int i=0; i<N; ++i)
sum1 += mat1[i][k]; %  访存不连续
sum2 += mat2[k][i]; %  访存连续
```

29. 一个 C 语言的 for 循环代码（部分）及其 64 位 Linux 汇编如下所示，请对照汇编填充 C 语言里的缺失部分。

```
int looper(int n, int *a) {
   int i;
   int x = _____;
   for(i = _____;
      _____;
      i++)
   {
      if (_____)
            x = _____;
      else_____;
   }
   return x;
}
```

```
looper:
        movl    $0, %eax
        movl    $0, %edx
        jmp     .L2
.L4:
        movslq  %edx, %rcx
        movl    (%rsi,%rcx,4), %ecx
        addl    $1, %eax
        cmpl    %eax, %ecx
        jle     .L3
        leal    (%rcx,%rcx), %eax
.L3:
        addl    $1, %edx
.L2:
        cmpl    %edi, %edx
        jl      .L4
        ret
```

```
int x=0;
for(i= 0; i<n; i++)
{
      if(a[i] > (++x))
            x=2*a[i];
      else
            ;
}
```

30. 选择哪（些）种是可能的程序输出.

```
int main () {
    if (fork() == 0) {
        if (fork() == 0) {
            printf("3");
        }
        else {
            pid_t pid; int status;
            if ((pid = wait(&status)) > 0) {
                printf("4");
            }
        }
    }
    else {
        printf("2");
        exit(0);
    }
    printf("0");
    return 0;
}
```

A. 32040	Y	N
B. 34002	Y	N
C. 30402	Y	N
D. 23040	Y	N
E. 40302	Y	N

进程 4 会等待 3 结束才输出，故 3 和 0 必须在 4 之前。

答案：A C D