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：2021 年 1 月 11 日 **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，共 16 页。

第一题 单项选择题（每小题 2 分，共 50 分）

注：选择题答案请按照如下格式整理到答题纸上

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										
题号	21	22	23	24	25					
回答										

1. 下列关于 x86-64 汇编，说法**错误**的是：

A. Bomblab 中 bomb 程序的代码段基址每次运行可能不同，因此其 switch 语句跳转表存储的不是目标代码的绝对地址，而是相对于某基址的偏移量。

B. 本门课程主要讲述的是 ATT 格式的汇编，其 mov 等双操作数指令一般以第一个操作数作为源而第二个操作数作为目标，这种操作数顺序通常与 Intel 格式的汇编相反。

C. 函数栈帧的长度在编译时被确定，因此 x86-64 中通用寄存器%rbp 不会再作为桢指针使用，但%rbp 依然是被调用者保存的寄存器。

D. 多数同学 Archlab 书写 Y86-64 汇编时，在过程调用中修改了%rbx 等部分寄存器而未事先保存，且未在过程调用结束时恢复，这种行为在 x86-64 汇编下是不可取的。

2. 假设内存空间为 64 字节，考虑两个高速缓存系统 A 与 B。它们均有 s=E=b=2，但高速缓存 A 要求组索引位在地址中间，而高速缓存 B 要求组索引位在地址的最高位，如图。设替换策略均采用 LRU，现用考虑如下访存序列(10 进制)：

1, 7, 8, 3, 4

则下列说法中**正确**的个数为：

①高速缓存 A 的容量为 32 字节，但 Tag 与 Valid 位的存在使得其实际占用的字节数是容量的 1.5 倍。

②若初始时缓存均为空，用该序列分别访存，其在高速缓存 A 中的 hit 次数多于 B。

③对于高速缓存 A，两个地址相邻的缓存块在相同的组中；而对于高速缓存 B，则在不同的组中。

④实际的缓存系统通常采用 A 的格式而不采用 B，是因为空间局部性较差时其能更好地利用缓存。

A. 0　　B. 1　　C. 2　　D. 3

3. 在int 类型值为32 位、用补码表示、采用算术右移，unsigned 类型值也为32 位的机器上，声明

int x, y;

unsigned ux = (unsigned)x;

unsigned uy = (unsigned)y;

则下列表达式中，当x 与y 取遍所有可能值时，可能为假的是

A. (x<y)==(-x>-y)

B. ~x+~y+1==~(x+y)

C. (ux-uy)==-(unsigned)(y-x)

D. ((x>>3)<<3)<=x

4. 下列哪种类型转换既可能导致溢出、又可能导致舍入?

A. int 转 float　B. float 转 int　C. int 转 double　D. float 转 double

5. 下面对指令系统的描述中，错误的是:（　）

A. 通常 CISC 指令集中的指令数目较多，有些指令的执行周期很长；而 RISC 指令集中指令数目较少，指令的执行周期较短。

B. 通常 CISC 指令集中的指令长度不固定；RISC 指令集中的指令长度固定。

C. 通常 CISC 指令集支持多种寻址方式，RISC 指令集支持的寻址方式较少。

D. 通常 CISC 指令集处理器的寄存器数目较多, RISC 指令集处理器的寄存器数目较少。

6. Y86 指令 rmmovl 的 SEQ 实现如下图所示，其中①和②分别为:

	rmmovl rA, D(rB)
Fetch	icode:ifun ← M_1[PC] rA:rB ← M_1[PC+1] valC ← M_4[PC+2] valP ← ①
Decode	valA ← R[rA] valB ← R[rB]
Execute	valE ← ②
Memory	M_4[valE] ← valA
Write back	
PC update	PC ← valP

A． PC + 4,　valB + 4　　B． PC + 4,　valB + valC

C． PC + 6,　valB + 4　　D． PC + 6,　valB + valC

7. 下列关于链接的描述，正确的是:

A．打桩（interpositioning)机制需要能够访问程序的源代码。

B. 链接器生成的文件不再有ABS、UNDEF 和COMMON 伪节。

C. 对于GlibC和GCC生成的程序，其入口点位于__libc_start_main函数的第一条指令处。

D．用户使用GCC进行共享库的编译必须使用-fpic选项指示生成位置无关代码。

8. 文件 f1.c 和 f2.c 的 C 源代码如下图所示：

```
//f1.c
#include<stdio.h>
extern float x;
extern void f();
int main()
{
   f();
   printf("%d\n",(int)x);
   return 0;
}
```

```
//f2.c
int x = 0;
void f()
{
   x++;
}
```

已知这两个文件在同一个目录下，在该目录下用“gcc -Og -o f f1.c f2.c”编译，然后用“./f”运行，这个过程中会出现的情况是：

A. 编译错误
B. 输出0
C. 输出1
D. 链接错误

9. C 语言中的代码如下：

```
fork() && fork();
printf("-");
fork() || fork();
printf("-");
```

这段代码一共输出（ ）个“-”字符。

A. 12
B. 18
C. 20
D. 32

10. 当一个网络数据包到达一台主机时，会触发以下哪种异常：

A. 系统调用
B. 信号
C. 中断
D. 缺页异常

11. 在键盘上输入 Ctrl+C 会导致内核发送一个（ ）信号到（ ）进程组中的每个进程

A. SIGINT，前台
B. SIGTSTP，前台
C. SIGINT，后台
D. SIGTSTP，后台

12. 以下关于缺页异常和系统调用的描述，不正确的是:
A. 两个异常都是同步异常
B. 两个异常的触发都是由于执行某一条指令
C. 两个异常在正常退出时，都需要判断是否有非阻塞的待处理信号
D. 两个异常在正常退出后，都需要重新执行触发异常的指令

13. 对于 Linux 系统，下列说法错误的是:
A. 在单核 CPU 上，所有看似并行的逻辑流实际上是并发的。
B. 用户模式不可访问/proc 文件系统中包含的内核数据结构的内容。
C.在 execve 函数参数的 argv 数组加入"> 1.txt"，不能自动实现 IO 重定向。
D. 即便在信号处理程序中调用 printf 前阻塞所有信号，也不一定安全。

14. 假设在 64 位系统下页大小被设置为 16KB，那么采用类似 Core i7 的地址翻译过程，四级页表最多可以索引多少位地址空间?
A. 64
B. 58
C. 54
D. 52

15. 为了支持 16G 的虚拟地址空间，采用 3 级页表，页大小为 1KB，页表项大小依旧为 4 字节。现在映射总大小为 1MB 的虚拟地址，其分布未知但分布的最小单位限定为 1 字节，请问实现上述映射的页表结构占用的内存至少至多分别为多大?
A. 6KB，4113KB
B. 6KB，65793KB
C. 6KB，1052688KB
D. 3KB，4113KB

16. 某 64 位系统，物理内存地址长度为 52 位，虚拟内存地址长度为 43 位，已知页的大小为 8KB，采用多级页表进行地址翻译，每级页表都占一页，则需要几级页表:
A. 2 级　B. 3 级　C. 4 级　D. 5 级

17. 在页表条目中，以下哪个条目是由 MMU 在读和写时进行设置，而由软件负责清除:
A. P 位，子页或子页表是否在物理内存中
B. G 位，是否为全局页（在任务切换时，不从 TLB 中驱逐出去）
C. CD 位，能/不能缓存子页或子页表
D. D 位，修改位，是否对子页进行了修改操作

18. 关于写时复制（copy-on-write）技术的说法，不正确的是:
A. 写时复制既可以发生在父子进程之间，也可以发生在对等线程之间
B. 写时复制既需要硬件的异常机制，也需要操作系统软件的配合
C. 写时复制既可以用于普通文件，也可以用于匿名文件
D. 写时复制既可以用于共享区域，也可以用于私有区域

19. 垃圾收集器的根节点不包括以下哪个节里的数据:

A. text

B. data

C. bss

D. stack

20. 按照课程中的描述，套接字 socket 接口是一组函数，它们与 UNIX I/O 函数一起使用可以构建网络应用。下列关于 socket 的描述中，哪一个是错误的?

A. 因特网的套接字地址存放在 socketaddr_in 结构中

B. 在使用套接字接口函数时，要将对应协议的特定结构指针转换成通用结构 sockaddr

C. 当客户端通过 connect()发起连接请求时，由服务器端的服务进程通过 listen()捕获该请求并建立监听套接字 listenfd

D. 对于迭代 echo 服务器，当 client1 连接到该服务器且尚未断开连接时，client2 若想要连接该服务器会阻塞在从服务器读结果的操作上

21. 下列关于全球 IP 因特网的描述中，哪一个是不正确的?

A. IPV4 协议中，一个 IP 地址是一个 32 位无符号整数

B. IP 地址结构中存放的地址总是以大端法存放的

C. 网络字节顺序是采用大端法的

D. 在 Linux 系统上，用 hostname -i 命令可以确定自己主机的域名

22. 同一个进程内的不同线程共享以下几项?

a) 函数体内部的静态变量

b) 堆

c) 文件描述符表

d) 条件码

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

23. 令 L_i 表示加载共享变量到寄存器的指令，U_i 表示更新寄存器的指令，S_i 表示将更新值存回到共享变量的指令，指令 L_1、S_1、U_3、S_3 操作共享变量 cnt，指令 L_2、S_2、L_4、S_4 操作共享变量 num，假定每次只允许交换相邻的两条指令，则将下列指令序列变为正确的至少需要交换多少次:

L_2、L_1、U_1、L_3、L_4、U_4、U_3、S_1、U_2、S_4、S_3、S_2

A.10 B.11 C.12 D.13

24. 以下程序在输出有限行之后就终止了，请问最有可能的原因是（假定所有函数都正常执行）

```
#include "csapp.h"
void *thread(void *dummy){
        while (1){
            printf("hello, world!\n");
            Sleep(1);
         }
}
int main(){
        pthread_t tid;
        Pthread_create(&tid, NULL, thread, NULL);
        Sleep(3);
}
```

A. 主线程结束必然引发所有对等线程结束
B. 主线程结束时调用了_exit 导致进程结束
C. 主线程结束后，内核发送 SIGKILL 杀死进程
D. 主线程结束后，内核观察到对等线程运行时间过长，将其杀死

25. 有四个信号量，初值分别为：a=1，b=1，c=1，d=1

线程 1:	线程 2:	线程 3:
① P(a);	⑨ P(c);	⑮ P(d);
② P(b);	⑩ P(b);	⑯ P(a);
③ P(d);	⑪ V(c);	⑰ V(d);
④ P(c);	⑫ V(b);	⑱ P(b);
⑤ V(d);	⑬ P(a);	⑲ V(a);
⑥ V(a);	⑭ V(a);	⑳ V(b);
⑦ V(b);		
⑧ V(c);		

上面的程序执行时，下列哪一个执行轨迹执行后已经出现死锁?

A. ⑨①⑩②⑪⑮
B. ①⑨⑩②⑮⑯
C. ⑮①⑨⑩⑯②
D. ①②③④⑨⑮

第二题（10 分）

有一个五级流水线处理器，和课程中的 PIPE 处理器类似，包含：取指（F)、译码（D)、执行（E)、访存（M)、写回（W）五级，

1）在理想设计情况下，流水线每一级的电路处理时间如下表所示，每两级流水线阶段之间插入的寄存器耗时为 1ns

	F	D	E	M	W
处理时间（ns）	8	5	9	8	4

请问流水线运行的时钟周期，最快为 _____ ns

2）需要执行如下 Y86-64 汇编代码，寄存器的初始状态如右下表格所示：

```
0x000    t:  mrmovq    (%rdi), %rbx
0x002        addq      %rbx, %rax
0x004        addq      $8, %rdi
0x006        addq      $1, %rcx
0x008        xorq      $8, %rcx
0x00a        jne       t
```

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

流水线支持课上讲的数据前递（data forwarding)，分支预测设计为“**始终跳转**”（predicted as taken)，请问执行上述代码，需要 _____ 个时钟周期?

3）现需要考虑实际执行时，存储层次对于流水线执行时间的影响：

- 该处理器包含一个 L1 指令缓存（I-cache）和一个 L1 数据缓存（D-cache)；
- I-cache 是一个直接映射（direct-mapped，E=1）缓存，总容量 32KB，每个缓存块的大小是 32Byte，代码执行前 I-cache 全部为空；
- D-cache 也是一个直接映射（direct-mapped，E=1）缓存，总容量 32KB，每个缓存块的大小是 32Byte，代码执行前 D-cache 全部为空；
- 取指（F）阶段，如果指令在 I-cache 中，则需要 1 个周期，否则需要 10 个周期；
- 访存（M）阶段，如果数据在 D-cache 中，则需要 1 个周期，否则需要 10 个周期；

请问执行上述代码 I-cache 缓存命中___次、D-cache 缓存命中_____次?

在存储层次的影响下，执行完代码需要 _____ 个时钟周期?

第三题（10 分）

本题基于下列 m.c 及 swap.c 文件所编译生成的 m.o 和 swap.o，编译和运行在 Linux/X86-64 下使用 GCC 完成，编译过程未加优化选项。

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

```
//swap.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++;
}
void func(){
    ......//略去题目无关代码
}
```

1）对于每个 swap.o 中定义和引用的符号，请用“是”或“否”指出它是否在模块 swap.o 的.symtab 节中存在符号表条目。如果存在条目，则请指出定义该符号的模块（swap.o 或 m.o）、符号类型（局部、全局或外部）以及该符号在所属的模块（“即该符号在该模块中被定义”）中所处的节；如果不存在条目，则请将该行后继空白处标记为“/”。

符号	.symtab 有条目?	符号类型	定义符号的模块	节
buf				
bufp0				
count				
func				
temp				

2）下图左边给出了 m.o 和 swap.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
b8 00 00 00 00          mov    $0x0,%eax
e8 00 00 00 00          callq  e <main+0xe>     ①
......
0f b6 05 00 00 00 00    movzbl 0x0(%rip),%eax  ②
0f be c0                movsbl %al,%eax
89 c7                   mov    %eax,%edi
b8 00 00 00 00          mov    $0x0,%eax
e8 00 00 00 00          callq  26 <main+0x26>  ③
......
b8 00 00 00 00          mov    $0x0,%eax
5d                      pop    %rbp
c3                      retq
```

```
0000000000......<myswap>:
55                 push   %rbp
48 89 e5           mov    %rsp,%rbp
89 f8              mov    %edi,%eax
88 45 fc           mov    %al,-0x4(%rbp)
......
48 c7 05 00 00 00 00 00 00 00 00          ⑤④
                     movq   $0x0,0x0(%rip)
......
48 8b 05 00 00 00 00  mov    0x0(%rip),%rax   ⑥
0f b6 00               movzbl (%rax),%eax
88 45 fc               mov    %al,-0x4(%rbp)
......
48 8b 05 00 00 00 00 mov    0x0(%rip),%rax    ⑦
48 8b 15 00 00 00 00 mov    0x0(%rip),%rdx    ⑧
0f b6 12             movzbl (%rdx),%edx
88 10                mov    %dl,(%rax)
48 8b 05 00 00 00 00 mov    0x0(%rip),%rax    ⑨
0f b6 55 fc          movzbl -0x4(%rbp),%edx
88 10                mov    %dl,(%rax)
......
8b 05 00 00 00 00     mov    0x0(%rip),%eax   ⑩
83 c0 01              add    $0x1,%eax
89 05 00 00 00 00     mov    %eax,0x0(%rip)   ⑪
90                    nop
5d                    pop    %rbp
c3                    retq

0000000000......<func>:
......
```

```
0000000000001000 <main>:
1000: 55                  push  %rbp
1001: 48 89 e5           mov   %rsp,%rbp
1004: b8 00 00 00 00      mov   $0x0,%eax
1009: e8      ①           callq 302e <func>
......
1010:  0f  b6  05          ②              movzbl
0x??????(%rip),%eax
1017: 0f be c0            movsbl %al,%eax
101a: 89 c7               mov    %eax,%edi
101c: b8 00 00 00 00      mov    $0x0,%eax
1021: e8      ③           callq  10e4 <myswap>
......
10db: b8 00 00 00 00       mov    $0x0,%eax
10e0: 5d                   pop    %rbp
10e1: c3                   retq

00000000000010e4<myswap>:
10e4: 55                   push   %rbp
10e5: 48 89 e5             mov    %rsp,%rbp
10e8: 89 f8                mov    %edi,%eax
10ea: 88 45 fc             mov    %al,-0x4(%rbp)
......
10f4: 48 c7 05      ⑤          ④
                   movq    ④ , ⑤ (%rip)
......
1104: 48 8b 05      ⑥       mov   ⑥ (%rip),%rax
110b: 0f b6 00              movzbl (%rax),%eax
110e: 88 45 fc              mov    %al,-0x4(%rbp)
......
1229: 48 8b 05      ⑦       mov   ⑦ (%rip),%rax
1230: 48 8b 15      ⑧       mov   ⑧ (%rip),%rdx
1237: 0f b6 12              movzbl (%rdx),%edx
123a: 88 10                 mov    %dl,(%rax)
123c: 48 8b 05      ⑨       mov   ⑨ (%rip),%rax
1243: 0f b6 55 fc           movzbl -0x4(%rbp),%edx
1247: 88 10                 mov    %dl,(%rax)
......
124b: 8b 05        ⑩        mov    ⑩ (%rip),%eax
1251: 83 c0 01              add    $0x1,%eax
1254: 89 05        ⑪        mov    %eax, ⑪ (%rip)
125a: 90                    nop
125b: 5d                    pop    %rbp
125c: c3                    retq

000000000000302e<func>:
......
   ......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......
000000000000a250 <bufp1>:
   ......略去部分和答题无关的信息......
0000000000dedd38 <count.1837>:
   ......略去部分和答题无关的信息......
0000000000dedd40 <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⑪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。

编号	重定位条目信息	应填入的重定位引用值 一律填写 32 位 16 进制数
①	r.offset = 0x0a　　r.symbol = 本题不提供 r.type = R_X86_64_PC32　　r.addend = -4	
③	r.offset = 0x22　　r.symbol = 本题不提供 r.type = R_X86_64_PC32　　r.addend = -4	
④	r.offset = 0x17　　r.symbol = 本题不提供 r.type = R_X86_64_32　　r.addend = 0	
⑨	r.offset = 0x15b　　r.symbol = 本题不提供 r.type = R_X86_64_PC32　　r.addend = -4	
⑪	r.offset = 0x172　　r.symbol = 本题不提供 r.type = R_X86_64_PC32　　r.addend = -4	

第四题（10 分）

C 语言中的代码如下：

```
int counter = 0;

void handler(int sig) {
        counter++;
}

int main(int argc, char *argv[])
{
        int fd1, fd2, fd3;
        char c1, c2, c3;
        char *fname = argv[1];
        int parent;

        signal(SIGUSR1, handler);

        fd1 = open(fname, O_RDONLY);
        read(fd1, &c1, 1);
        parent = getpid();
        if (fork()) {
                wait();
                fd2 = open(fname, O_RDONLY);
        } else {
                kill(parent, SIGUSR1);
                fd2 = dup(fd1);
        }
        read(fd2, &c2, 1);

        parent = getpid();
        if (fork()) {
                wait();
                fd3 = open(fname, O_RDONLY);
        } else {
                kill(parent, SIGUSR1);
                fd3 = dup(fd2);
        }

        read(fd3, &c3, 1);
        printf("%c %c %c %d\n", c1, c2, c3, counter);
        return 0;
}
```

当 fname 文件内容为 abcde 时，这段代码的打印结果为____行，输出结果如下：（不考虑出错的情况）

第五题（10 分）

考虑一对 n*n 矩阵相乘问题：C=AB。矩阵乘法函数通常是用 3 个嵌套的循环来实现的，分别用索引 i、j 和 k 来标识。如果改变循环的次序，我们就能得到矩阵乘法的 6 个在功能上等价的版本，如下图所示：

`for (i = 0; i < n; i++)`
`  for (j = 0; j < n; j++) {`
`  sum = 0.0;`
`  for (k = 0; k < n; k++)`
`  sum += A[i][k]*B[k][j];`
`  C[i][j] += sum;`
`  }`
	`for (j = 0; j < n; j++)`
	`  for (i = 0; i < n; i++) {`
	`  sum = 0.0;`
	`  for (k = 0; k < n; k++)`
	`  sum += A[i][k]*B[k][j];`
	`  C[i][j] += sum;`
	`  }`

1）ijk 版本	**2）jik 版本**

`for (j = 0; j < n; j++)`
`  for (k = 0; k < n; k++) {`
`  r = B[k][j];`
`  for (i = 0; i < n; i++)`
`  C[i][j] += A[i][k]*r;`
`  }`
	`for (k = 0; k < n; k++)`
	`  for (j = 0; j < n; j++) {`
	`  r = B[k][j];`
	`  for (i = 0; i < n; i++)`
	`  C[i][j] += A[i][k]*r;`
	`  }`

3）jki 版本	**4）kji 版本**

`for (k = 0; k < n; k++)`
`  for (i = 0; i < n; i++) {`
`  r = A[i][k];`
`  for (j = 0; j < n; j++)`
`  C[i][j] += r*B[k][j];`
`  }`
	`for (i = 0; i < n; i++)`
	`  for (k = 0; k < n; k++) {`
	`  r = A[i][k];`
	`  for (j = 0; j < n; j++)`
	`  C[i][j] += r*B[k][j];`
	`  }`

5）kij 版本	**6）ikj 版本**

本题目具有以下假设：

- 每个数组都是一个 double 类型的 n*n 的数组，sizeof(double)==8
- 虚拟存储分页管理，页大小为 4K 字节，且 A 的起始地址为页对齐，ABC 连续存放
- **TLB 为全相联结构**
- **初始时 TLB 为空，数据全部在硬盘中**
- 一级 dTLB 有 64 表项，采用 LRU（Least-Recently-Used）置换策略
- 分析过程只考虑矩阵操作本身的数据空间，且 i/j/k/n/r/sum 等变量都存放在寄存器中
- 分析过程不考虑进程切换和异常处理程序的影响
- **不考虑对页表本身的访问**

n 的取值	**缺页次数**	**一级 dTLB 失效次数**					
		1-ijk	**2-jik**	**3-jki**	**4-kji**	**5-kij**	**6-ikj**
8							
16							
64							
512							

第六题（10 分）

1. 火锅洞洞主终于要请吃火锅啦！由于防疫规定，洞主决定分批邀请，Alice、Bob、……、Zach 这 26 位同学顺理成章地成为了第一批受到邀请的同学。**他们围坐在一张圆桌旁，按照首字母顺序排列**（即 Alice 的右边是 Bob，Bob 的右边是 Carol，……，Zach 的右边是 Alice）。吃饭之前，洞主想要验证他们的身份，他提出了这样的要求：
    1) 每位同学有一个**编号**，即**字母序号减 1**，也就是说，Alice 是 0 号，Bob 是 1 号，以此类推。
    2) 每位同学需要同时拿着**旁边两位同学**的手机去找洞主（**不必带上自己的手机**），按照洞主的要求在树洞里发信息以验证身份。
    3) **可以同时有多位同学**找洞主验证（也就是说，Alice 需要拿着 Zach 和 Bob 的手机找洞主，并且在同一时间，Bob 也可以拿着 Alice 和 Carol 的手机找洞主。但是，如果 Alice 还没同时拿到 Zach 和 Bob 的手机，就**只能坐在座位上等待**。）

所幸，在座有多位字母君学过 ICS，他们决定采用**信号量**解决问题。

1.1 Alice 给出了这样的代码，其中 thread 函数的参数指明了字母君的编号：

```
1.  #include "csapp.h"
2.  #define N 26
3.  #define UP(i) ((i + 1) % N)
4.  #define DOWN(i) ((i - 1 + N) % N)
5.  sem_t sem[N];
6.
7.  void *thread(void *i){
8.      int num = (int)i;
9.      P(&sem[UP(num)]);
10.     P(&sem[DOWN(num)]);
11.
12.     /* verify identity */
13.
14.     V(&sem[UP(num)]);
15.     V(&sem[DOWN(num)]);
16. }
17. int main(){
18.     for (int i = 0; i < N; i++)
19.         Sem_init(&sem[i], 0, 1);
20.     pthread_t tid[N];
21.     for (int i = 0; i < N; i++)
22.         Pthread_create(&tid[i], NULL, thread, (void *)i);
23.     Pthread_exit(0);
24. }
```

Bob 一看，皱起了眉头。请问这段代码可能会发生什么问题（用一个专用名词表述即可）？

1.2 Carol 把 thread 函数换成了下面这样：

```
1.  void *thread(void *i){
2.      int num = (int)i;
3.      for (int i = 0; i < N; i++)
4.          P(&sem[i]);
5.      /* verify identity */
6.      for (int i = 0; i < N; i++)
7.          V(&sem[i]);
8.  }
```

Dave 指出，这段代码固然正确，但是效率太低了，他认为只要把第 3、4 行的代码换成一个 P 操作，把第 6、7 行的代码换成一个 V 操作，就能实现同样的效果。请问 Dave 的方法是：

1.2.1 P 操作：P(&sem[_____]);

1.2.2 V 操作：V(&sem[_____]);

1.3 Eve 看了 Dave 的方案还是摇头，这样执行效率太低了，没能利用**同时可以有多位**同学找洞主验证的条件。借助“**互斥锁加锁顺序规则**”，他给出了自己的代码。你能把代码补全吗？

```
1.  void *thread(void *i)
2.  {
3.      int num = (int)i;
4.      if (UP(num) < DOWN(num)){
5.          P(&sem[__A__]);
6.          P(&sem[__B__]);
7.      }
8.      else{
9.          P(&sem[__C__]);
10.         P(&sem[__D__]);
11.     }
12.     /* verify identity */
13.     V(&sem[UP(num)]);
14.     V(&sem[DOWN(num)]);
15. }
```

A:_____ B:_____ C:_____ D:_____

以上 4 空均填序号：

① UP(num)

② DOWN(num)

③ num

2. Hungary Alice 发现自己没吃上第一场的火锅，一怒之下，给上面的 26 位字母君出了一道难题：**用二元信号量实现信号量**（也就是值可以是任意非负整数的信号量）。精通 ICS 的你自然认为这是小儿科，迅速补全了以下代码。

```
#include "csapp.h"
typedef struct{
    int value;
    sem_t mutex;
    sem_t zero;
} my_sem_t;
void my_sem_init(my_sem_t *sem, unsigned int value){
    sem->value = value;
    Sem_init(&sem->mutex, 0, __A__);
    Sem_init(&sem->zero, 0, __B__);
}
void my_P(my_sem_t *sem){
    P(&sem->mutex);
    sem->value--;
    if (sem->value __C__ 0)
    {
        __D__;
        __E__;
    }
    else
        V(&sem->mutex);
}
void my_V(my_sem_t *sem){
    P(&sem->mutex);
    sem->value++;
    if (sem->value __F__ 0)
    {
        V(&sem->mutex);
        V(&sem->zero);
    }
    else
        V(&sem->mutex);
}
```

A:_____ B:_____ C:_____

D:_____ E:_____ F:_____

D 和 E 请填序号:

① P(&sem->mutex)

② P(&sem->zero)

③ V(&sem->mutex)

④ V(&sem->zero)