第一章　计算机系统漫游

一、　选择题

1. 计算机操作系统抽象表示时(　　)是对处理器、主存和 I/O 设备的抽象表示。

A. 进程　　B. 虚拟存储器　　C. 文件　　D. 虚拟机

2. 操作系统通过提供不同层次的抽象表示来隐藏系统实现的复杂性，其中(　　)是对实际处理器硬件的抽象。

A. 进程　　B. 虚拟存储器　　C. 文件　　D. 指令集架构（ISA）

二、　填空题

3. C 语言程序中的常量表达式的计算是由_____完成的。

4. 将 hello.c 编译生成汇编语言的命令行_____。

三、　简答题

5. 简述 C 编译过程对非寄存器形式的 int 全局变量与 int 局部变量处理的区别。包括存储区域、赋初值、生命周期、指令中寻址方式等。

四、 分析题

6. 在终端中的命令行运行显示“Hello World”的执行程序 hello，结合进程创建、加载、缺页中断、到存储访问（虚存）……等等，论述 hello 是怎么一步步执行的。

第二章　信息的表示与处理

一、　选择题

1. C 语句中的有符号常数，在（　　）阶段转换成了补码

A.编译　　B.连接　　C.执行　　D.调试

7. 计算机常用信息编码标准中，字符 0 的编码不可能是 16 进制数（　）

A.30　　B.30 00　　C.00　　D.00 30

8. C 语言中 float 类型的数据 0.1 的机器数表示，错误的是（　　）

A．规格化数　B.不能精确表示　C.与 0.2 有 1 个二进制位不同 D. 唯一的

9. 下列 16 进制数值中，可能是 Linux64 系统中 char*类型的指针值是（　　）

A.e4f9　　B.b4cc2200　　C.b811e5ffff7f0000　　D.30

10. 关于 IEEE　float 类型的数据+0.0 的机器数表示，说法错误的是（　）

A．是非规格化数　　B.不能精确表示　　C.+0.0 与-0.0 不同　　D. 唯一的

11. C 语句中的有符号常数，在（ ）阶段转换成了补码

A.编译　　B.连接　　C.执行　　D.调试

12. 计算机信息常用编码中，字符 0 的编码不可能是 16 进制数（　）

A.30　　B.30 00　　C.00　　D.00 30

13. C 语言中 float 数据 0.1 的机器数表示错误的是（　）

A．规格化数　B.不能精确表示　C.与 0.2 有 1 个二进制位不同 D. 唯一的

14. 程序中的 2 进制、10 进制、16 进制数，在（　　）时变成 2 进制

A.汇编时　　B.连接时　　C.执行时　　D.调试时

15. C 语言程序中的整数常量、整数常量表达式是在（　）阶段变成 2 进制补码的。

(A）预处理　　（B）编译　　（C）连接　　（D）执行阶段

16. C 语言中不同类型的数值进行强制类型转换时，下列说法错误的是（　）

A．从 int 转换成 float 时，数值可能会溢出

B．从 int 转换成 double 后，数值不会溢出

C．从 double 转换成 float 时，数值可能会溢出，也可能舍入

D．从 double 转换成 int 时，数值可能溢出，可能舍入

二、　填空题

17. 64 位系统中 int 数 -2 的机器数二进制表示_____。

18. Intel 桌面 X86-64 CPU 采用_____端模式。

19. C 语言中 short 类型-2 的机器数二进制表示为_____。

20. C 语言中的 double 类型浮点数用_____位表示。

21. 判断整型变量 n 的位 7 为 1 的 C 语言表达式是_____。

22. 整型变量 x=-2,其在内存从低到高依次存放的数是 (16 进制表示)_____。

23. 若字节变量 x 和 y 分别为 0x10 和 0x01，则 C 表达式 x&&~y 的字节值是_____。

24. 按照“向偶数舍入”的规则，二进制小数 101.110_2 舍入到最接近的 1/2（小数点右边 1 位）后的二进制为_____。

25. C程序中定义 int x=-3，则&x 处依次存放（小端模式）的十六进制数据为_____。

三、 判断题

26. （ ）C 语言中从 int 转换成 float 时，数字不会溢出，但可能舍入。

27. （ ）C 语言程序中，有符号数强制转换成无符号数时，其二进制表示将会做相应调整。

28. （ ）C 语言中对整型指针 p，当 p=null 时，表达式 p&&*p++会间接引用空指针。

29. （ ）C 语言中，关系表达式：127 > (unsigned char)128U 是成立的。

30. （ ）x 和 y 是 C 中的整型变量，若 x 大于 0 且 y 大于 0，则 x+y 一定大于 0。

31. （ ）CPU 无法判断参与加法运算的数据是有符号或无符号数。

32. （ ）C 浮点常数 IEEE754 编码的缺省舍入规则是四舍五入。

33. （ ）对 unsigned int x，（x*x）>=0 总成立。

34. （ ）CPU 无法判断加法运算的和是否溢出。

35. （ ）C 浮点常数 IEEE754 编码的缺省舍入规则是向上舍入。

36. （ ）C 语言中的有符号数强制转换成无符号数时位模式不会改变。

37. （ ）C 语言中从 double 转换成 float 时，值可能溢出，但不可能被舍入。

38. （ ）C 语言中 int 的个数比 float 个数多。

39. （ ）C 语言中数值从 int 转换成 double 后，数值虽然不会溢出，但有可能是不精确的。

四、 简答题

40. 写出 float f=-1 的 IEEE754 编码。（请按步骤写出转换过程）

41. 请在数轴上画出非负 float 数的各区间的密度分布，并标示各区间是规格化还是非规格化数、浮点数密度、最小值、最大值。

42. 请结合 ieee754 编码，说明怎样判断两个浮点数是否相等？

第三章　程序的机器级表示

一、　选择题

1. 递归函数程序执行时，正确的是（　　）

A．使用了堆　　B.可能发生栈溢出　　C.容易有漏洞　D.必须用循环计数器

2. 一台主流配置的 PC 上，调用 f(35)所需时间大概是(　)

A.几毫秒　　B.几秒　　C.几分钟　　D.几小时

```
int f(int x){
    int s = 0;
    printf("%d_",x);
    while(x++ >0)
        s += f(x);
    return max(s,1);
}
```

3. 下列叙述正确的是（　　）

A.一条 mov 指令不可以使用用两个内存操作数

B.在一条指令执行期间，CPU 不会两次访问内存

C.CPU 不总是执行 CS::RIP 所指向的指令，例如遇到 call、ret 指令时

D.X86-64 指令"mov$1,%eax"不会改变%rax 的高 32 位

4. 条件跳转指令 JE 是依据（ ）做是否跳转的判断

A. ZF　　B. OF　　C. SF　　D. CF

5. 在 x86-64 中，有初始值%rax =0x1122334455667788，执行下述指令后 rax 寄存器的值是(　)

```
movl $0xaa11,  %rax
```

A．0xaal1　　B.0x112233445566aa11

C. 0x112233440000aa11　　D.0x11223344ffffaa11

6. x86-64 中，某 C 程序定义了结构体

```
struct SS {
        double v;
        int i;
        short s;
```

} aa[10];

则执行 sizeof(aa)的值是（　）

A．14　　B.80　　C.140　　D. 160

二、　填空题

7. 64 位 C 语言程序中第一个参数采用_____传递。

8. 64 位 C 语言程序在函数调用时第二个整型参数采用寄存器_____传递。

9. C 语言 64 位系统中参数传递采用_____。

10. C 语言的常量表达式的计算是由_____完成的

11. C 语言程序定义了结构体 struct noname{char c; int n; short k; char *p;};若该程序编译成 64 位可执行程序，则 sizeof(noname)的值是_____。

三、　判断题

12. （　）C 的标准 IO 函数都是带缓冲的，Unix 的 IO 函数不带缓冲。

13. （　）X86-64 CPU 中的寄存器一定都是 64 位的。

四、　简答题

14. 从汇编的角度阐述：函数 int sum(int x1,int x2,int x3,int x4,int x5,int x6,int x7,int x8)，调用和返回的过程中，参数、返回值、控制是如何传递的？并画出 sum 函数的栈帧（X86-64 形式）。

15. 简述缓冲区溢出攻击的原理以及防范方法。

16. 下列 C 程序存在安全漏洞，请给出攻击方法。如何修复或防范？

```
int getbuf(char *s) {
    char    buf[32];
    strcpy( buf, s );
}
```

五、 分析题

17. 某 C 程序(64 位)的 main 函数参数 argv 地址为 0x0000413433323110，其内容如下：

0x0000413433323110：30 31 32 33 34 41 00 00 33 31 32 33 34 41 00 00

0x0000413433323120：35 31 32 33 34 41 00 00 00 00 00 00 00 00 00 00

0x0000413433323130：31 43 00 30 00 32 42 00 38 00 31 31 32 32 00 30

0x0000413433323140：32 33 00 61 41 00 31 00 32 00 33 00 31 00 00 31

请写出程序名:_____，本程序的参数个数_____，按顺序写出各个参数为_____

_____

18. 有下列 C 函数:

```
long arith(long x, long y, long z)
{
  long t1 =_____（1）_____;
  long t2 =_____（2）_____;
  long t3 =_____（3）_____;
  long t4 = _____（4）_____;
 _____（5）_____;
}
```

函数 arith 的汇编代码如下：

```
arith:
xorq    %rsi,%rdi
leaq    (%rdi,%rdi,4),%rax
leaq    (%rax,%rsi,2),%rax
subq    %rdx,%rax
retq
```

请填写出上述 C 语言代码中缺失的部分

（1）_____（2）_____（3）_____

（4）_____（5）_____

19. 已知内存和寄存器中的数值情况如下：

内存地址	值
0x100	0xff
0x104	0xAB
0x108	0x13
0x10c	0x11

寄存器	值
%rax	0x100
%rcx	0x1
%rdx	0x3

请填写下表，给出对应操作数的值：

操作数	值
%rax	
(%rax)	
9(%rax,%rdx)	
0xfc(,%rcx,4)	
(%rax,%rdx,4)	

20. 有下列 C 函数:

```
long arith(long x, long y, long z)
{
   long t1 =_____（1）_____;
   long t2 =_____（2）_____;
   long t3 =_____（3）_____;
   long t4 =_____（4）_____;
   _____（5）_____;
}
```

函数 arith 的汇编代码如下：

```
arith:
orq     %rsi,%rdi
sarq    $3,%rdi
notq    %rdi
movq    %rdx,%rax
subq    %rdi,%rax
retq
```

请填写出上述 C 语言代码中缺失的部分

（1）_____（2）_____（3）_____

（4）_____（5）_____

21. 某 C 函数(函数体只有一条 C 语句)的 64 位与 32 位的反汇编结果分别如下：

```
4005d6: push   %rbp                    804849b:  push   %ebp
4005d7: mov    %rsp,%rbp               804849c:  mov    %esp,%ebp
4005da: mov    %rdi,-0x8(%rbp)         804849e:  mov    0x8(%ebp),%eax
4005de: mov    -0x8(%rbp),%rax         80484a1:  mov    (%eax),%eax
4005e2: mov    (%rax),%rax             80484a3:  lea    0x4(%eax),%ecx
4005e5: lea    0x4(%rax),%rcx          80484a6:  mov    0x8(%ebp),%edx
4005e9: mov    -0x8(%rbp),%rdx         80484a9:  mov    %ecx,(%edx)
4005ed: mov    %rcx,(%rdx)             80484ab:  mov    (%eax),%eax
4005f0: mov    (%rax),%eax             80484ad:  pop    %ebp
4005f2: pop    %rbp                    80484ae:  ret
4005f3: retq
```

请写出函数 f 的返回值类型_____,参数 p 的类型_____

函数体的唯一一条 C 语句_____。

第四章　处理器体系结构

一、　选择题

1. Y86-64 的 CPU 顺序结构设计与实现中，分成（　　）个阶段

A.5　　B.6　　C.7　　D.8

2. 关于 Intel 的现代 X86-64 CPU 正确的是（　）

A．属于 RISC　B. 属于 CISC　C. 属于 MISC　D. 属于 NISC

3. 为了使计算机运行得更快，现代 CPU 采用了许多并行技术，将处理器的硬件组织成若干个阶段并让这些阶段并行操作的技术是（　　），该技术的 CPI 一般不小于 1。

A. 流水线　B.超线程　C.超标量　D.向量机

4. Y86-64 的指令编码长度是（　）个字节

A.1～10　　B.32　　C.64　　D.128

5. 在 Y86-64 指令集体系结构中，程序员可见的状态不包括（　　）

A．程序寄存器　B.高速缓存　C.条件码　D.程序状态

6. X86-64 中，通过寄存器传递整型参数时，第一个参数用寄存器（　　）访问

A.%rdi　　B.%edi　　C.%rsi　　D.%edi

7. Intel 桌面 CPU I7 没有采用如下现代 CPU 设计技术（　）

A. 流水线　B.超线程　C.超标量　D.向量机

8. 在 Y86-64 CPU 中有 15 个从 0 开始编码的通用寄存器，在对指令进行编码时，对于仅使用一个寄存器的指令，简单有效的处理法是（ ）

A.用特定的指令类型代码

B.用特定的指令功能码

C.用特定编码 0xFF 表示操作数不是寄存器

D.无法实现

9. 下列 Y86-64 硬件结构中，程序员不可见的是（　）

A．程序寄存器　B.算逻运算单元（ALU）　C.程序计数器 D. 内存

10. Y86-64CPU 顺序结构设计中，在更新 PC 时与指令 jmp 地址来源相同的指令是（　）

A．pushq　　B.call　　C.cmovxx　　D. ret

二、　判断题

11. （　）现代超标量 CPU 指令的平均周期接近于 1 个但大于 1 个时钟周期。

12. （　）Y86-64 的顺序结构实现中，寄存器文件读时是作为时序逻辑器件看待。

13. （　）现代超标量 CPU 指令的平均周期通常小于 1 个时钟周期。

14. （　）Y86-64 的顺序结构实现中，寄存器文件写时是作为组合逻辑器件看待。

15. （　）Y86-64 的顺序结构实现中，寄存器是时序逻辑器件。

三、 简答题

16. 简述 Y86-64 流水线 CPU 中的冒险的种类与处理方法。

17. 参照 Y86-64 流水线 CPU 的实现，说明流水线如何工作

四、 分析题

18. 请写出 Y86-64 CPU 顺序结构设计与实现中，POP 指令在各阶段的微操作。

19. 请写出 Y86-64 CPU 顺序结构设计与实现中，mrmovq 指令在各阶段的操作。

20. 为Y86-64 CPU增加一指令"iaddq　V,rB"，将常量数值V加到寄存器rB。参考irmovq、OPq 指令，请设计 iaddq 指令在各阶段的微操作。

指令	irmovq V,rB	OPq rA, rB	iaddq V,rB
取指	icode:ifun←M1[PC] rA:rB←M1[PC+1] valC←M8[PC+2] valP←PC+10	icode:ifun←M1[PC] rA:rB←M1[PC+1] valP←PC+2	
译码	valB←0	valA←R[rA] valB←R[rB]	
执行	valE←valB+valC	valE←valB OP valA Set CC	
访存			
写回	R[rB]←valE	R[rB]←valE	
更新 PC	PC←valP	PC←valP	

21. 写出 Y86-64CPU 顺序结构设计中 addq 指令各阶段的微操作。为 Y86-64 CPU 增加一条指令"mraddq D(rB)，rA"，能够将内存数据加到寄存器 rA。请参考 mrmovq、addq 指令，合理设计 mraddq D(rB)，rA 指令在各阶段的微操作，或给出设计思想。(10 分)

22. 计算机的 FPU 是采用堆栈架构实现的（其运算在栈顶附近的数据进行），中间层语言如 MSIL、JavaByteCode 也采用类堆栈 CPU。请按照 Y86-64 的顺序结构实验原理，设计一个 S86-64 的 Stack CPU，完成堆栈的压栈与出栈等基本操作。CPU 要求的指令系统如下：

halt: 00

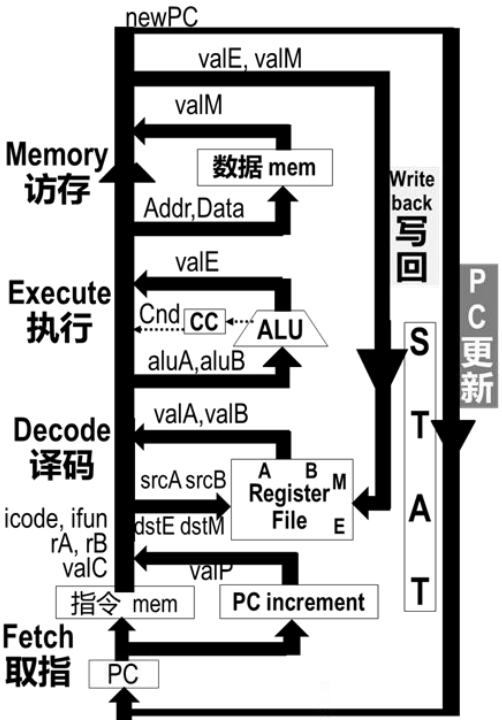
nop: 10

push imm：20 64 位立即数

push rA： 3|rA

pop rA： 4|rA

注：先期不用考虑堆栈的初始化、空、满的判断、运算的支持等等，以后可逐步扩展指令与标志位等等。且 S86-64 的寄存器与 Y86-64 一样，硬件结构与指令执行的阶段可根据需要进行优化。



（1） 请写出 POP rA 指令在各阶段的微操作。

（2） 画出访存阶段的硬件结构图

（3） 用 HCL 语言写出存储器地址与数据的控制逻辑。

第五章　优化程序性能

一、　选择题

1. C 语言程序如下，叙述正确的是（　）

```
#include <stdio.h>
#define DELTA sizeof(int)
int main(){
    int i;
    for (i = 40;  i - DELTA >= 0; i -= DELTA)
    printf("%d ",i);
}
```

A. 程序有编译错误

B. 程序输出 10 个数：40 36 32 28 24 20 16 12 8 4 0

C. 程序死循环，不停地输出数值

D. 以上都不对

2. 利用 GCC 生成代码过程中，不属于编译器优化的结果是（　）

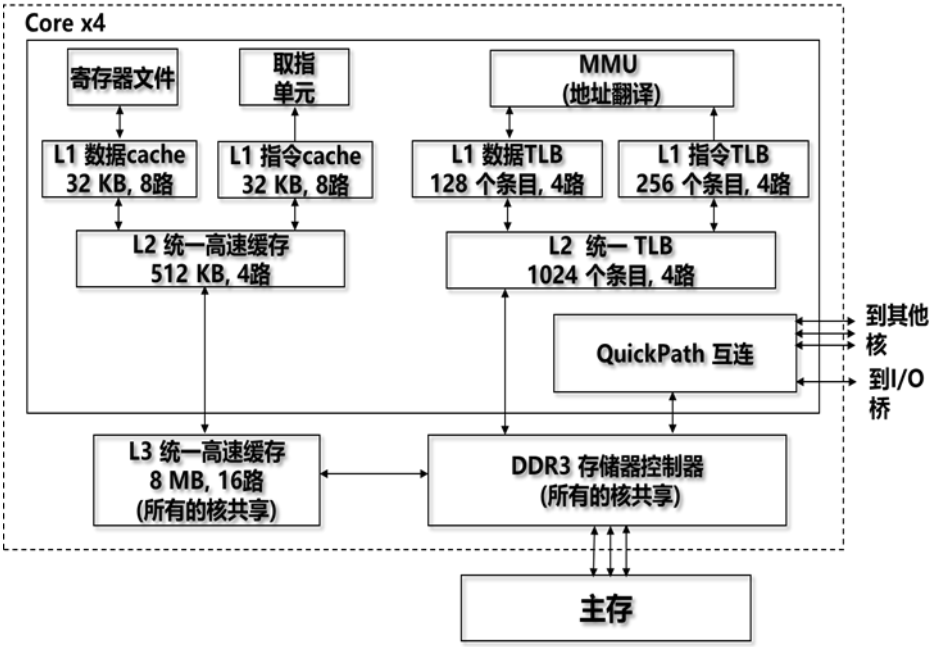
A．用移位操作代替乘法指令　B.消除循环中的函数调用

C. 循环展开　D.使用分块提高时间局部性

二、　简答题

3. 列举几种程序优化的方法，并简述其原理。

三、 分析题



4. 程序优化: 矩阵 c[n,n] = a[n,n] * b[n,n] ，采用题首 I7 CPU。块 64B。

```
for(int i=0;i<n;i++){
    for(int j=0;j<n;j++){
        c[i,j]=0;
        for(int k=0; k<n;k++)
            c[i,j]+=a[i,k]*b[k][j];
    }
}
```

5. 程序优化: 矩阵 c[n,n] = a[n,n] * b[n,n] ，采用题首 I7 CPU。块 64B。

```
for(int i=0;i<n;i++)
    for(int j=0;j<n;j++)
    {
        c[i,j]=0;
     for(int k=0; k<n;k++)
            c[i,j]+=a[i,k]*b[k][j];
      }
```

请给出基于编译、CPU、存储器的三种优化方法，并编写程序。

6. 现代超标量 CPU X86-64 的 Cache 的参数 s=5，E=1，b=5，若 M=N=64, 请优化如下程序，并说明优化的方法（至少 CPU 与 Cache 各一种）。

```
void trans(int M, int N, int A[M][N], int B[N][M])
{
    for (int i = 0; i < M; i++)
        for (int j = 0; j < N; j++)
            B[j][i] =A[i][j];
}
```

7. 优化如下程序，给出优化结果并说明理由。（10 分）

```
int sum_array(int a[M][N][N])  //M、N 足够大
{
    int i, j, k, sum = 0;
    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++)
            for (k = 0; k < M; k++)
            sum += a[k][i][j];
    return sum;
}
```

8. 向量元素和计算的相关程序如下，请改写或重写计算函数 vector_sum，进行速度优化，并简要说明优化的依据。

```
/*向量的数据结构定义 */
typedef struct{
    int len;          //向量长度，即元素的个数
    float *data;      //向量元素的存储地址
} vec;
/*获取向量长度*/
int vec_length(vec *v){return v->len;}
/* 获取向量中指定下标的元素值，保存在指针参数 val 中*/
int get_vec_element(*vec v, size_t idx, float *val){
    if (idx >= v->len)
        return 0;
        *val = v->data[idx];
         return 1;
}
/*计算向量元素的和*/
void vector_sum(vec *v, float *sum){
    long int i;
    *sum = 0;//初始化为 0
    for (i = 0; i < vec_length(v); i++) {
        float val;
        get_vec_element(v, i, &val);      //获取向量 v 中第 i 个元素的值，存入 val 中
        *sum = *sum + val;                //将 val 累加到 sum 中
    }
}
```

第六章　存储器层次结构

一、　选择题

1. 位于存储器层次结构中的最顶部的是(　　)。

A. 寄存器　　B. 主存　　C. 磁盘　　D. 高速缓存

2. CPU 一次访存时，访问了 L1、L2、L3 Cache 所用地址 A1、A2、A3 的关系（　）

A.A1>A2>A3　　B.A1=A2=A3　　C.A1<A2<A3　　D.A1=A2<A3

3. 下列各种存储器中存储速度最快的是(　　)。

A. 寄存器　　B. 主存　　C. 磁盘　　D. 高速缓存

4. 采用缓存系统的原因是（ ）

A. 高速存储部件造价高

B.程序往往有比较好的空间局部性

C. 程序往往有比较好的时间局部性

D.以上都对

5. UNIX I/O 的 read、write 函数无法读/写指定字节的数据量，称为“不足值”问题，叙述正确的是(　　)

A.读磁盘文件时遇到 EOF，会出现“不足值”问题

B.写磁盘文件也会出现“不足值”问题

C.读磁盘文件不会有这个问题

D.以上均不对

6. CPU 寄存器作为计算机缓存层次结构的最高层，决定哪个寄存器存放某个数据的是（　）

A. MMU　　B.操作系统内核　　C.编译器　　D.CPU

二、　填空题

7. 某 CPU 主存地址 32 位，高速缓存总大小为 4K 行，块大小 16 字节，采用 4 路组相连，则标记位的总位数（每行标记位数*总行数）是_____。

8. 存储器层次结构中，高速缓存（Cache）是_____的缓存。

三、　判断题

9. （　）全相联 Cache 不会发生冲突不命中的情况。

10. （　）直接映射 Cache 一定会发生冲突不命中的情况。

11. （　　）Cache 的大小对程序运行非常重要，必要的时候可以通过操作系统提高 Cache 的大小。

12. （　　）CPU 在同一次访问 Cache L1、L2、L3 时使用的地址是一样的。

四、　简答题

13. 简述程序的局部性原理，如何编写局部性好的程序？

14. 结合下面的程序段，解释局部性。

```
int cal_array_sum(int *a,int n){
    int sum = 0;
    for (int i = 0; i < n; i++)
        sum += a[i];
    return sum;
}
```

五、　分析题

15. 某 CPU 的 L1 cache 容量 32kb，64B/块，采用 8 路组相连，物理地址 47 位。试分析其结构参数 B、S、E 分别是多少？地址 0x00007f6635201010 访问该 L1 时，其块偏移 CO、组索引 CI、标记 CT 分别多少？（5 分）

第七章　链接

一、　选择题

1. 当函数调用时，(　　)可以在程序运行时动态地扩展和收缩。

A. 程序代码和数据区　　B. 栈　　C. 共享库　　D. 内核虚拟存储器

2. 在 Linux 系统中利用 GCC 作为编译器驱动程序时，能够将汇编程序翻译成可重定位目标程序的程序是（　　）

A．cpp　　B.ccl　　C.as　　D. ld

3. 连接时两个文件同名的弱符号，以（　　）为基准

A．连接时先出现的　B．连接时后出现的　C.任一个　D.连接报错

4. 连接过程中，赋初值的局部变量名，正确的是（　　）

A.强符号　　B.弱符号　　C.若是静态的则为强符号　　D.以上都错

5. C 语句中的全局变量，在（　　）阶段被定位到一个确定的内存地址

A.编译　　B.链接　　C.执行　　D.调试

6. 链接时两个同名的强符号，以哪种方式处理？（　　）

A．链接时先出现的符号为准　B．链接时后出现的符号为准

C.任一个符号为准　　D. 链接报错

7. 链接过程中，带 static 属性的全局变量属于（　）

A.全局符号　　B.局部符号　　C.外部符号　　D.以上都错

8. 以下关于程序中链接“符号”的陈述，错误的是（ ）

A.赋初值的非静态全局变量是全局强符号

B.赋初值的静态全局变量是全局强符号

C.未赋初值的非静态全局变量是全局弱符号

D.未赋初值的静态全局变量是本地符号

9. 关于动态库的描述错误的是（ ）

A.可在加载时链接，即当可执行文件首次加载和运行时进行动态链接。

B.更新动态库，即便接口不变，也需要将使用该库的程序重新编译。

C.可在运行时链接，即在程序开始运行后通过程序指令进行动态链接。

D.即便有多个正在运行的程序使用同一动态库，系统也仅在内存中载入一份动态库。

10. 若将标准输出重定向到文本文件 file.txt，错误的是（ ）

A.需要先打开重定位的目标文件"file.txt"

B.设"file.tx"t 对应的 fd 为 4,内核调用 dup2(1,4)函数实现描述符表项的复制

C.复制"file.txt"的打开文件表项、并修正 fd 为 1 的描述符

D.修改"file.txt"的打开文件表项的引用计数

11. 关于局部变量，正确的叙述是（ ）

A.普通（auto）局部变量也是一种编程操作的数据，存放在数据段

B.非静态局部变量在链接时是本地符号

C.静态局部变量是全局符号

D.编译器可将 rsp 减取一个数为局部变量分配空间

12. Linux 系统中将可执行目标文件（.out 文件）装入到存储空间时，没有装入到.text 段-只读代码段的是（ ）

A.ELF 头 B. .init 节 C. .rodata 节 D. .symtab 节

13. 链接过程中，赋初值的静态全局变量属于（ ）

A.强符号 B.弱符号 C.可能是强符号也可能是弱符号 D.以上都不是

二、 填空题

14. 链接器经过_____和重定位两个阶段，将可重定位目标文件生成可执行目标文件。

15. 若 p.o->libx.a->liby.a 且 liby.a->libx.a->p.o 则最小链接命令行_____。

16. 可重定位目标文件中代码地址从_____开始。

三、 判断题

17. （ ）链接时，若一个强符号和多个弱符号同名，则对弱符号的引用均将被解析成强符号。

四、 简答题

18. 简述 C 编译过程对非寄存器实现的 int 全局变量与非静态 int 局部变量处理的区别。包括存储区域、赋初值、生命周期、指令中寻址方式等。

19. 什么是共享库（动态连接库）？简述动态链接的实现方法。

20. 什么是静态库？使用静态库的优点是什么？

五、 分析题

21-23 两个 C 语言程序 main.c、test.c 如下所示：

```
/* main.c */
#include <stdio.h>
int a[4]={-1,-2,2, 3};
extern int val;
int sum();
int main(int argc, char * argv[ ] )
{
    val=sum();
    printf("sum=%d\n",val);
}
```

```
/* test.c */
extern int a[ ];
int val=0;
int sum( )
{
    int i;
    for (i=0; i<4; i++)
        val += a[i];
    return val;
}
```

用如下两条指令编译、链接，生成可执行程序 test：

gcc -m64 -no-pie -fno-PIC -c test.c main.c

gcc -m64 -no-pie -fno-PIC -o test test.o main.o

运行指令 objdump -dxs main.o 输出的部分内容如下：

```
Contents of section .data:
0000 ffffffff feffffff 02000000 03000000    ................
Contents of section .rodata:
0000 73756d3d 25640a00                      sum=%d..
    …
Disassembly of section .text:
    0000000000000000 <main>:
       0:   55                          push   %rbp
       1:   48 89 e5                    mov    %rsp,%rbp
       4:   48 83 ec 10                 sub    $0x10,%rsp
       8:   89 7d fc                    mov    %edi,-0x4(%rbp)
       b:   48 89 75 f0                 mov    %rsi,-0x10(%rbp)
       f:   b8 00 00 00 00              mov    $0x0,%eax
      14:   e8 00 00 00 00              callq  19 <main+0x19>
                15: R_X86_64_PC32     sum-0x4
      19:   89 05 00 00 00 00           mov    %eax,0x0(%rip)    # 1f <main+0x1f>
                1b: R_X86_64_PC32     val-0x4
      1f:   8b 05 00 00 00 00           mov    0x0(%rip),%eax    # 25 <main+0x25>
                21: R_X86_64_PC32     val-0x4
      25:   89 c6                       mov    %eax,%esi
      27:   bf 00 00 00 00              mov    $0x0,%edi
                28: R_X86_64_32   .rodata
      2c:   b8 00 00 00 00              mov    $0x0,%eax
      31:   e8 00 00 00 00              callq  36 <main+0x36>
                32: R_X86_64_PC32     printf-0x4
      36:   b8 00 00 00 00              mov    $0x0,%eax
      3b:   c9                          leaveq
      3c:   c3                          retq
```

objdump -dxs test 输出的部分内容如下（■是没有显示的隐藏内容）：

```
SYMBOL TABLE:
0000000000400400 l         d .text   0000000000000000          .text
00000000004005e0 l         d .rodata      0000000000000000     .rodata
0000000000601020 l         d .data  0000000000000000           .data
0000000000601040 l         d .bss   0000000000000000           .bss
0000000000000000           F *UND*     0000000000000000    printf@@GLIBC_2.2.5
0000000000601044 g        O .bss   0000000000000004           val
0000000000601030 g        O .data  0000000000000010           a
00000000004004e7 g        F .text   0000000000000039          sum
0000000000400400 g        F .text   000000000000002b          _start
0000000000400520 g        F .text   000000000000003d          main
Contents of section .rodata:
 4005e0 01000200 73756d3d 25640a00                  ....sum=%d..
…
Contents of section .data:
 601020 00000000 00000000 00000000 00000000    ................
 601030 ffffffff feffffff 02000000 03000000    ................
…
00000000004003f0 <printf@plt>:
  4003f0:       ff 25 22 0c 20 00       jmpq    *0x200c22(%rip) # 601018 <printf@GLIBC_2.2.5>
```

```
  4003f6:    68 00 00 00 00        pushq   $0x0
  4003fb:    e9 e0 ff ff ff        jmpq    4003e0 <.plt>

Disassembly of section .text:
0000000000400400 <_start>:
  400400:    31 ed           xor     %ebp,%ebp
....
00000000004004e7 <sum>:
  4004e7:    55                  push   %rbp          #①
  4004e8:    48 89 e5            mov    %rsp,%rbp #②
  4004eb:    c7 45 fc 00 00 00 00 movl   $0x0,-0x4(%rbp) #③
  4004f2:    eb 1e                jmp    400512 <sum+0x2b>
  4004f4:    8b 45 fc             mov    -0x4(%rbp),%eax
  4004f7:    48 98                cltq
  4004f9:    8b 14 85 30 10 60 00 mov    0x601030(,%rax,4),%edx
  400500:    8b 05 3e 0b 20 00    mov    0x200b3e(%rip),%eax #601044 <val>
  400506:    01 d0                add    %edx,%eax
  400508:    89 05 36 0b 20 00    mov    %eax,0x200b36(%rip) #601044 <val>
  40050e:    83 45 fc 01          addl   $0x1,-0x4(%rbp)
  400512:    83 7d fc 03          cmpl   $0x3,-0x4(%rbp)#④
  400516:    7e dc                jle    4004f4 <sum+0xd>#⑤
  400518:    8b 05 26 0b 20 00    mov    0x200b26(%rip),%eax # 601044 <val>
  40051e:    5d                   pop    %rbp
  40051f:    c3                   retq

0000000000400520 <main>:
  400520:    55                  push   %rbp
  400521:    48 89 e5            mov    %rsp,%rbp
  400524:    48 83 ec 10         sub    $0x10,%rsp
  400528:    89 7d fc            mov    %edi,-0x4(%rbp)
  40052b:    48 89 75 f0         mov    %rsi,-0x10(%rbp)
  40052f:    b8 00 00 00 00      mov    $0x0,%eax
  400534:    e8(    ①    )       callq   4004e7 <sum>
  400539:    89 05(      ②    )   mov    %eax, ■■■■■(%rip) #601044<val>
  40053f:    8b 05(     ③     )  mov    ■■■■■(%rip),%eax    #601044<val>
  400545:    89 c6               mov    %eax,%esi
  400547:    bf（    ④    ）     mov      ■■■■■,%edi
  40054c:    b8 00 00 00 00      mov     $0x0,%eax
  400551:    e8（    ⑤    ）     callq    4003f0 <printf@plt>
  400556:    b8 00 00 00 00      mov     $0x0,%eax
  40055b:    c9                  leaveq
  40055c:    c3                  retq
  40055d:    0f 1f 00            nopl    (%rax)
```

21. 阅读的 sum 函数反汇编结果中带下划线的汇编代码（编号①-⑤），解释每行指令的功能和作用（5 分）

22. 根据上述信息，链接程序从目标文文件 test.o 和 main.o 生成可执行程序 test，对 main 函数中空格①-⑤所在语句所引用符号的重定位结果是什么？以 16 进制 4 字节数值填写这些空格，将机器指令补充完整（写出任意 2 个即可）。（5 分）

23. 在 sum 函数地址 4004f9 处的语句"mov 0x601030(,%rax,4),%edx"中，源操作数是什么类型、有效地址如何计算、对应 C 语言源程序中的什么量(或表达式)？其中，rax 数值对应 C 语言源程序中的哪个量(或表达式)？如何解释数字 4？（5 分）

24-25 两个 C 语言程序 main2.c、addvec.c 如下所示：

```
/* main2.c */
/* $begin main2 */
#include <stdio.h>
#include "vector.h"

int x[2] = {1, 2};
int y[2] = {3, 4};
int z[2];

int main()
{
addvec(x, y, z, 2);
printf("z = [%d %d]\n", z[0], z[1]);
return 0;
}
/* $end main2 */
```

```
/* addvec.c */
/* $begin addvec */
int addcnt = 0;

void addvec(int *x, int *y,
int *z, int n)
{
int i;

addcnt++;

for (i = 0; i < n; i++)
z[i] = x[i] + y[i];
}
/* $end addvec */
```

用如下两条指令编译、链接，生成可执行程序 prog2：
gcc -m64 -no-pie -fno-PIC -c addvec.c main2.c
gcc -m64 -no-pie -fno-PIC -o prog2 addvec.o main2.o
运行指令 objdump -dxs main2.o 输出的部分内容如下：
Disassembly of section .text:

```
0000000000000000 <main>:
0:   48 83 ec 08             sub    $0x8,%rsp
 4:  b9 02 00 00 00          mov    $0x2,%ecx
 9:  ba 00 00 00 00          mov    $0x0,%edx
     a: R_X86_64_32 z
 e:  be 00 00 00 00          mov    $0x0,%esi
     f: R_X86_64_32 y
13:  bf 00 00 00 00          mov    $0x0,%edi
     14: R_X86_64_32     x
18:  e8 00 00 00 00          callq  1d <main+0x1d>
     19: R_X86_64_PC32 addvec-0x4
1d:  8b 0d 00 00 00 00       mov    0x0(%rip),%ecx        # 23 <main+0x23>
     1f: R_X86_64_PC32 z
23:  8b 15 00 00 00 00       mov    0x0(%rip),%edx        # 29 <main+0x29>
     25: R_X86_64_PC32 z-0x4
29:  be 00 00 00 00          mov    $0x0,%esi
     2a: R_X86_64_32     .rodata.str1.1
2e:  bf 01 00 00 00          mov    $0x1,%edi
33:  b8 00 00 00 00          mov    $0x0,%eax
38:  e8 00 00 00 00          callq  3d <main+0x3d>
     39: R_X86_64_PC32 __printf_chk-0x4
3d:  b8 00 00 00 00          mov    $0x0,%eax
```

```
42:  48 83 c4 08              add     $0x8,%rsp
46:  c3                       retq
```

objdump -dxs prog2 输出的部分内容如下（■是没有显示的隐藏内容）：

```
SYMBOL TABLE:
0000000000400238 l     d  .interp  0000000000000000              .interp
0000000000400254 l     d  .note.ABI-tag
0000000000000000 l     df *ABS*    0000000000000000              main2.c
0000000000601038 g          *ABS*  0000000000000000              _edata
000000000060103c g     O .bss 0000000000000008              z
0000000000601030 g     O .data     0000000000000008              x
0000000000000000       F *UND* 0000000000000000              addvec
0000000000601018 g          .data  0000000000000000              __data_start
00000000004007e0 g     O .rodata   0000000000000004              _IO_stdin_used
0000000000601028 g     O .data     0000000000000008              y
00000000004006f0 g     F .text 0000000000000047              main

00000000004005c0 <addvec@plt>:
  4005c0:   ff 25 42 0a 20 00          jmpq    *0x200a42(%rip)          # 601008

    <_GLOBAL_OFFSET_TABLE_+0x20>
  4005c6:   68 01 00 00 00             pushq   $0x1
  4005cb:   e9 d0 ff ff ff             jmpq    4005a0 <_init+0x18>

00000000004005d0 <__printf_chk@plt>:
  4005d0:   ff 25 3a 0a 20 00          jmpq    *0x200a3a(%rip)          # 601010

    <_GLOBAL_OFFSET_TABLE_+0x28>
  ....

00000000004006f0 <main>:
 4006f0:      48 83 ec 08             sub     $0x8,%rsp
 4006f4:      b9 02 00 00 00          mov     $0x2,%ecx
 4006f9:      ba ①_ _ _ _             mov     ■■■■,%edx
 4006fe:      be ②_ _ _ _             mov     ■■■■,%esi
 400703:      bf ③_ _ _ _             mov     ■■■■,%edi
 400708:      e8 ④_ _ _ _             callq   4005c0 <addvec@plt>
 40070d:      8b 0d ⑤_ _ _ _          mov     ■■■■(%rip),%ecx          # 601040 <z+0x4>
 400713:      8b 15 ⑥_ _ _ _          mov     ■■■■(%rip),%edx          # 60103c <z>
 400719:      be e4 07 40 00          mov     $0x4007e4,%esi
 40071e:      bf 01 00 00 00          mov     $0x1,%edi
 400723:      b8 00 00 00 00          mov     $0x0,%eax
 400728:      e8 ⑦ _ _ _ _            callq   4005d0 <__printf_chk@plt>
 40072d:      b8 00 00 00 00          mov     $0x0,%eax
 400732:      48 83 c4 08             add     $0x8,%rsp
 400736:      c3                      retq
```

24. 请指出 addvec.c main2.c 中哪些是全局符号？哪些是强符号？哪些是弱符号？以及这些符号经链接后在哪个节？（5 分）

25. 根据上述信息，main 函数中空格①--⑦所在语句所引用符号的重定位结果是什么？以 16 进制 4 字节数值填写这些空格，将机器指令补充完整（写出任意 3 个即可）。（5 分）

① _____ ②_____

③_____ ④ _____

⑤_____ ⑥ _____

⑦ _____

第八章　异常控制流

一、　选择题

1. 每个信号类型都有一个预定义的默认行为，可能是（　）

A.进程终止　B.进程挂起直到被 SIGCONT 重启　C.进程忽略该信号　D.以上都是

2. C 程序执行到整数或浮点变量除以 0 可能发生（　）

A.显示除法溢出错直接退出　B.程序不提示任何错误

C.可由用户程序确定处理办法　D.以上都可能

3. 同步异常不包括（　）

A.终止　B.陷阱　C.停止　D.故障

4. 进程上下文切换不会发生在如下（　）情况

A.当前进程时间片用尽　B.外部硬件中断

C.当前进程调用系统调用　D.当前进程发送了某个信号

5. Linux 下显示当前目录内容的指令为(　)

A.dir　B.man　C.ls　D.cat

6. 一个子进程终止或者停止时，操作系统内核会发送（　）信号给父进程。

A．SIGKILL　B.SIGQUIT　C.SIGSTOP　D.SIGCHLD

7. 进程从用户模式进入内核模式的方法不包括（　）

A.中断　B.陷阱　C.复位　D.故障

8. 内核为每个进程维持一个上下文，不属于进程上下文的是（　）

A.寄存器　B.进程表　C.文件表　D.调度程序

9. Linux 进程终止的原因可能是(　)

A.收到一个信号　B.从主程序返回　C.执行 exit 函数　D.以上都是

10. 进程上下文切换发生在如下（　）情况

A.当前进程时间片用尽　B.外部硬件中断

C.当前进程调用系统调　D.当前进程发送了某个信号

11. 内核为每个进程保存上下文用于进程的调度，不属于进程上下文的是（　）

A.全局变量值　B.寄存器　C.虚拟内存一级页表指针　D.文件表

12. 不属于同步异常的是（　）

A.中断　B.陷阱　C.故障　D.终止

13. 异步信号安全的函数要么是可重入的（如只访问局部变量）要么不能被信号处理程序中断，包括 I/O 函数（　）

A. printf　　B. sprint　　C. write　　D. malloc

14. 进程从用户模式进入内核模式的方法不包括（ ）

A.中断　　B.陷阱　　C.复位　　D.故障

15. 关于异常处理后返回的叙述，错误的叙述是（　　）

A.中断处理结束后，会返回到下一条指令执行

B.故障处理结束后，会返回到下一条指令执行

C.陷阱处理结束后，会返回到下一条指令执行

D.终止异常，不会返回

16. 下列异常中经异常处理后能够返回到异常发生时的指令处的是（　）

A. 键盘中断　　B.陷阱　　C. 故障　　D. 终止

17. 导致进程终止的原因不包括（　）

A. 收到一个信号　B.执行 wait 函数　C. 从主程序返回　D.执行 exit 函数

18. 下列不属于进程上下文的是（　　）

A.页全局目录 pgd　　B.通用寄存器　　C.内核代码　　D.用户栈

19. 下列函数中属于系统调用且在调用成功后，不返回的是(　)

A.fork　　B.execve　　C.setjmp　　D.longjmp

20. 三个进程其开始和结束时间如下表所示，则说法正确的是(　　)

进程	开始时刻	结束时刻
P1	1	5
P2	2	8
P3	6	7

A. P1、P2、P3 都是并发执行　　B.只有 P1 和 P2 是并发执行

C. 只有 P2 和 P3 是并发执行　　D.P1 和 P2、P2 和 P3 都是并发执行

二、　填空题

21. 程序执行到 A 处继续执行后，想在程序任意位置还原到执行到 A 处的状态，需要通过_____进行实现。

22. 进程创建函数 fork 执行后返回_____次。

23. 非本地跳转中的 setjmp 函数调用一次，返回_____次。

24. 进程加载函数 execve，如调用成功则返回_____次。

25. 子程序运行结束会向父进程发送_____信号。

26. 向指定进程发送信号的 linux 命令是_____。

三、 判断题

27. （ ）Linux 系统调用中的功能号 n 就是异常号 n 。

28. （ ）fork 的子进程中与其父进程同名的全局变量始终对应同一物理地址。

29. （ ）进程一旦终止就不再占用内存资源。

30. （ ）execve 加载新程序时会覆盖当前进程的地址空间，但不创建新进程。

31. （ ）异常处理程序运行在内核模式下，对所有的系统资源都有完全的访问权限。

32. （ ）子进程即便运行结束，父进程也应该使用 wait 或 waitpid 对其进行回收。

33. （ ）相比标准 I/O，Unix I/O 函数是异步信号安全的，可以在信号处理程序中安全地使用。

34. （ ）当执行 fork 函数时，内核为新进程创建虚拟内存并标记内存区域为私有的写时复制，意味着新进程此时获得了独立的物理页面。

35. （ ）进程是并发执行的，所以能够并发执行的都是进程。

四、 简答题

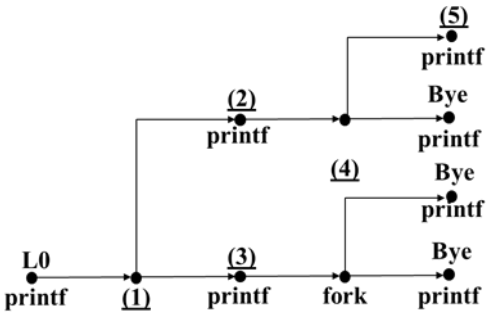
36. Linux 如何处理信号？应当如何编写信号处理程序？谈谈你的理解。

37. 结合 fork，execve 函数，简述在 shell 中加载和运行 hello 程序的过程。

五、 分析题

38. C 程序 fork2 的源程序与进程图如下：

```
void fork2(){
    printf("L0\n");
    fork();
    printf("L1\n");
    fork();
    printf("Bye\n");
}
```



请写出上述进程图中空白处的内容

（1）_____（2）_____（3）_____

（4）_____（5）_____

39. 设一个 C 语言源程序 p.c,编译连接后生成执行程序 p，反汇编如下：

C 程序

```
#include <stdio.h>
unsigned short b[2500];
unsigned short k;
void main()
{
    b[1000]=1023;
    b[2000]=2049%k;
```

反汇编程序的 main 部分（还有系统代码）如下:

main 的地址为 0x80482C0: （short 占 2 字节）

```
1  movw   $0x3ff,0x80497d0
2  movw   0x804a324,%cx          ;k=>cx
3  mov    $0x801,%eax
4  xorw   %dx,%dx
5  div    %ecx                   ;2049/k
6  movw   %dx,0x804a324
```

```
    b[10000]=20000;          7   movw    $0x4e20,0x804de20
}                            8   ret
```

现代 Intel 桌面系统，采用虚拟页式存储管理，每页 4K，p 首次运行时系统中无其他进程。请阅读如上 C 与汇编程序，结合进程与虚拟存储管理的知识，分析：

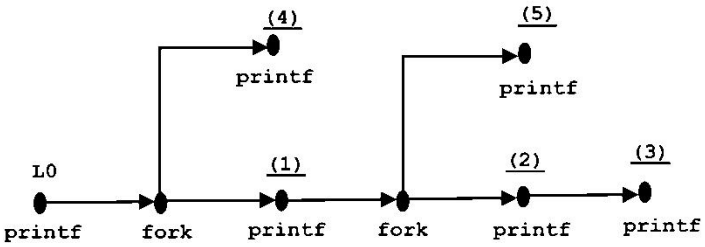
（1） 上述程序的执行过程中，在取指令时发生的缺页异常次数为_____.

（2） 写出已恢复的故障指令序号与故障类型_____

（3） 写出没有恢复的故障指令序号与故障类型_____

40. C 程序 forkB 的源程序与进程图如下：

```
void forkB( ){
    printf("L0\n");
    if(fork( )!=0);{
        printf("L1\n");
        if(fork( )!=0){
            printf("L2\n");
        }
    }
    printf("Bye\n");
}
```



请写出上述进程图中空白处的内容

（1）_____（2）_____（3）_____

（4）_____（5）_____

41. 一个 C 程序的 main()函数如下：

```
int main( ){
    if(fork()==0){
        printf("a"); fflush(stdout);
        exit(0);
    }
    else{
    printf("b"); fflush(stdout);
    waitpid(-1,NULL,0);
    }
    printf("c");  fflush(stdout);
    exit(0);
}
```

（1）请画出该程序的进程图

（2）该程序运行后，可能的输出数列是什么？

42. C 程序如下，请画出对应的进程图，并回答父进程和子进程分别输出什么？

```
int main()
{
        int x = 1;
        if(Fork()  ！ = 0)
                printf(“p1: x=%d\n", ++x);
        printf(“p2: x=%d\n", --x);
        exit(0);
}
```

第九章　虚拟内存

一、　选择题

1. 当函数调用时，(　　)可以在程序运行时动态地扩展和收缩。

A. 程序代码和数据区　　B. 栈　　C. 共享库　　D. 内核虚拟存储器

2. Intel X86-64 的现代 CPU，采用（　　）级页表

A. 2　B.3　C.4　D.由 BIOS 设置确定

3. 存储器垃圾回收时，内存被视为一张有向图，不能作为根结点的是（　　）

A. 寄存器　B.栈里的局部变量　C.全局变量　D.堆里的变量

4. “Hello World”执行程序很小不到 4k，在其首次执行时产生缺页中断次数（　）

A.0　B.1　C.2　D.多于 2 次

5. 在进程的虚拟地址空间中，用户代码不能直接访问的区域是(　　)

A.程序代码和数据区　B.栈　C. 共享库　D. 内核虚拟内存区

6. 记录内存物理页面与虚拟页面映射关系的是（　）

A. 磁盘控制器　B.编译器　C.虚拟内存　D.页表

7. 某 CPU 使用 32 位虚拟地址和 4KB 大小的页时，需要 PTE 的数量是（　　）

A. 16　B.8　C.1M　D.512K

8. 动态内存分配时的块结构中，关于填充字段的作用不可能的是（　　）

A.减少外部碎片　B.满足对齐　C.标识分配状态　D.可选的

9. 虚拟内存系统中的虚拟地址与物理地址之间的关系是（　）

A.1 对 1　B.多对 1　C.1 对多　D.多对多

10. 虚拟内存发生缺页时，缺页中断是由（　　）触发

A.内存　B.Cache L1　C.Cache L2　D.MMU

11. 当调用 malloc 这样的 C 标准库函数时，(　)可以在运行时动态的扩展和收缩。

A. 堆　B. 栈　C. 共享库　D. 内核虚拟存储器

12. 虚拟内存页面不可能处于（　）状态

A.未分配、未载入物理内存　B. 未分配但已经载入物理内存

C.已分配、未载入物理内存　D. 已分配、载入物理内存

13. 下面叙述错误的是（　）

A.虚拟页面的起始地址%页面大小恒为 0；

B.虚拟页面的起始地址%页面大小不一定是 0；

C.虚拟页面大小必须和物理页面大小相同；

D.虚拟页面和物理页面大小是可设定的系统参数；

14. 虚拟内存发生缺页时，正确的叙述是（ ）触发的

A. 缺页异常处理完成后，重新执行引发缺页的指令

B. 缺页异常处理完成后，不重新执行引发缺页的指令

C. 缺页异常都会导致程序退出

D. 中断由 MMU 触发

15. 程序语句"execve("a.out",NULL,NULL);"在当前进程中加载并运行可执行文件 a.out 时,错误的叙述是（ ）

A.为代码、数据、bss 和栈创建新的、私有的、写时复制的区域结构

B.bss 区域是请求二进制零的，映射到匿名文件，初始长度为 0；

C.堆区域也是请求二进制零的，映射到匿名文件，初始长度为 0；

D.栈区域也是请求二进制零的，映射到匿名文件，初始长度为 0；

16. 某进程在成功执行函数 malloc(24)后，下列说法正确的是（ ）

A. 进程一定获得一个大小 24 字节的块

B. 进程一定获得一个大于 24 字节的块

C. 进程一定获得一个不小于 24 字节的块

D. 进程可能获得一个小于 24 字节的块

17. 虚拟页面的状态不可能是（ ）

A．未分配 B．已分配未缓存 C．已分配已缓存 D．已缓存未分配

二、 填空题

18. C 语言函数中的整数常量都存放在程序虚拟地址空间的_____段。

19. TLB(翻译后备缓冲器）俗称快表，是_____的缓存。

20. 虚拟页面的状态有_____、已缓存、未缓存共 3 种

21. I7 的 CPU，L2 Cache 为 8 路的 2M 容量，B=64，则其 Cache 组的位数 s=_____。

22. 虚拟内存系统借助_____这一数据结构将虚拟页映射到物理页。

23. Linux 虚拟内存区域可以映射到普通文件和_____，这两种类型的对象中的一种。

24. （　　）系统中当前运行进程能够分配的虚拟页面的总数取决于虚拟地址空间的大小。

25. C 语言 printf 中的格式串是都存放在内存的_____段。

26. Intel I7 的 CPU 其 TLB 的每行的存储块 Block 是_____字节。

27. 虚拟页面的状态有_____、已缓存、未缓存共 3 种

28. I7 的 CPU，L2Cache 为 8 路的 2M 容量，则其 Cache 组的位数 s=_____。

29. 程序运行时，指令中的立即操作数存放的内存段是：_____段。

30. Cache 命中率分别是 97%和 99%时，访存速度差别_____(很大/很小)。

31. 当工作集的大小超过高速缓存的大小时，会发生_____不命中。

32. 虚拟内存在内存映射时，映射到匿名文件的页面是_____的页。

33. 存储器层次结构中，高速缓存（Cache）是_____的缓存。

34. TLB 常简称快表，它是_____的缓存。

35. 虚拟内存发生缺页时，MMU 将触发_____。

三、 判断题

36. （　）在动态内存分配中，内部碎片不会降低内存利用率。

37. （　）如果系统中程序的工作集大小超过物理内存大小，虚拟内存系统会产生抖动。

38. （　）虚拟内存系统能有效工作的前提是软件系统具有“局部性”。

39. （　）动态存储器分配时显式空闲链表比隐式空闲链表的实现节省空间。

40. （　）动态内存隐式分配是指应用隐式地分配块并隐式地释放已分配块。

41. （　）显式空闲链表的优点是在对堆块进行搜索时，搜索时间只与堆中的空闲块数量成正比。

四、 简答题

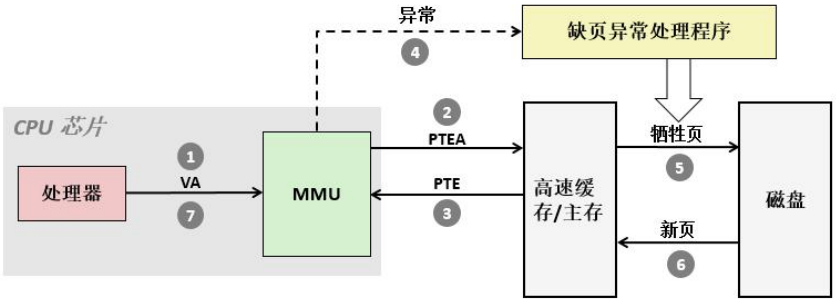
42. 假设：某 CPU 的虚拟地址 14 位；物理地址 12 位；页面大小为 64B；TLB 是四路组相联，共 16 个条目；L1 数据 Cache 是物理寻址、直接映射，行大小为 4 字节，总共 16 个组。分析如下项目：

（1）虚拟地址中的 VPN 占_____位；物理地址的 PPN 占_____位。

（2）TLB 的组索引位数 TLBI 为_____位。

（3）用物理地址访问 L1 数据 Cache 时，Cache 的组索引 CI 占_____位，Cache 标记 CT 占_____位。

43. 结合下图，简述虚拟内存地址翻译的过程。



五、 分析题

44. Intel I7 CPU 的虚拟地址 48 位，物理地址 52 位。其内部结构如下图所示，依据此结构，每一页面 4KB，分析如下项目：

虚拟地址中的 VPN 占_____位；其一级页表为_____项。

L1 数据 TLB 的组索引位数 TLBI 为_____位，L1 数据 Cache 共_____组。

用物理地址访问 L1 数据 Cache 时， Cache 标记 CT 占_____位