1. **单项选择题（每小题1分，共20分）**
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（ C ）**

**A．cpp B.ccl C.as D. ld**

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

**movl $0xaa11, %rax**

**A．0xaal1 B.0x112233445566aa11**

**C. 0x112233440000aa11 D.0x11223344ffffaa11**

1. **下列Y86-64硬件结构中，程序员不可见的是（ B ）**

**A．程序寄存器 B.算逻运算单元（ALU） C.程序计数器 D. 内存**

1. **利用GCC生成代码过程中，不属于编译器优化的结果是（ D ）**

**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**

1. **记录内存物理页面与虚拟页面映射关系的是（ D ）**

**A. 磁盘控制器 B.编译器 C.虚拟内存 D.页表**

1. **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（ B ）**

**A．pushq B.call C.cmovxx D. ret**

1. **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（ C ）**

**A. MMU B.操作系统内核 C.编译器 D.CPU**

1. **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（ D ）**

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

1. **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（ C ）**

**A. 键盘中断 B.陷阱 C. 故障 D. 终止**

1. **导致进程终止的原因不包括（ B ）**
2. **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**
3. **某进程在成功执行函数 malloc(24)后，下列说法正确的是（ C ）**

**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**

1. **下列不属于进程上下文的是（ C ）**

**A.页全局目录pgd B.通用寄存器 C.内核代码 D.用户栈**

1. **下列函数中属于系统调用且在调用成功后，不返回的是( B )**

**A.fork B.execve C.setjmp D.longjmp**

1. **动态内存分配时产生内部碎片的原因不包括（ D ）**

**A. 维护数据结构的开销**

**B. 满足对齐约束**

**C. 分配策略要求**

**D. 超出空闲块大小的分配请求**

1. **链接过程中，赋初值的静态全局变量属于（ D ）**

**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**

1. **虚拟页面的状态不可能是（ D ）**

**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**

1. **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 ）**

**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**

1. **三个进程其开始和结束时间如下表所示，则说法正确的是( D )**

|  |  |  |
| --- | --- | --- |
| **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**

1. **x86-64系统中，函数int sum (int x, int y)经编译后其返回值保存在（ C ）**

**A．%rdi A. %rdi B.%rsi C.%rax D.%rdx**

1. **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（ D ）**

**A．14 B.80 C.140 D. 160**

**二、填空题 ( 每空1 分，共 10 分 )**

1. **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\_0x01\_\_\_。**
2. **按照“向偶数舍入”的规则，二进制小数101.1102舍入到最接近的1/2（小数点右边1位）后的二进制为\_\_\_110.02\_\_\_\_\_\_\_\_。**
3. **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\_ FD FF FF FF\_\_。**
4. **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\_\_\_\_72K\_\_\_\_\_\_\_\_\_\_\_。**
5. **可重定位目标文件中代码地址从\_\_0\_\_\_\_\_开始。**
6. **当工作集的大小超过高速缓存的大小时，会发生\_\_容量\_\_\_\_\_\_\_不命中。**
7. **虚拟内存在内存映射时，映射到匿名文件的页面是\_\_\_\_请求二进制零/bss/stack/heap\_\_\_\_的页。**
8. **存储器层次结构中，高速缓存（Cache）是 主存/DRAM/RAM/内存/下级缓存/二级缓存 的缓存。**
9. **TLB常简称快表，它是 页表/PTE 的缓存。**
10. **虚拟内存发生缺页时，MMU将触发 缺页中断/异常/故障 。**

**三、判断对错（每小题1分，共10分，在题前打√ X 符号）**

1. **（ X ）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。**
2. **（ X ）C语言中，关系表达式：127 > (unsigned char)128U 是成立的。**
3. **（ X ）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。**
4. **（ X ）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。**
5. **（ √ ）CPU在同一次访问Cache L1、L2、L3时使用的地址是一样的。**
6. **（ X ）进程是并发执行的，所以能够并发执行的都是进程。**
7. **（ X ）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。**
8. **（ √ ）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。**
9. **（ X ）X86-64 CPU中的寄存器一定都是64位的。**
10. **（ X ）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。**

**四、简答题（每小题5分，共20分）**

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

**-1 =-1.0x20  阶为0**

**所以，符号S=1（1分）**

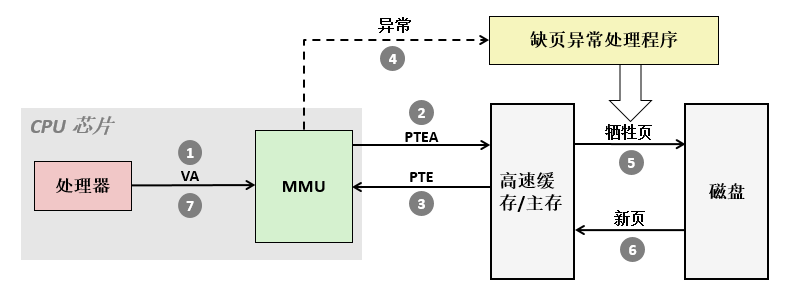
**阶码E=127+0=127=011111112（1分）**

**尾数=000 0000 0000 0000 0000 00002（1分）**

**f的IEEE754编码为1 01111111 000 0000 0000 0000 0000 0000**

**其十六进制为0xBF800000（2分） 可写小端存储形式00 00 80 BF**

1. **结合下图，简述虚拟内存地址翻译的过程。**



1. **处理器将虚拟地址发送给MMU（0.5分）**
2. **、（3）MMU利用虚拟地址对应的虚拟页号生成页表项（PTE）地址，并从页表中找到对应的PTE（0.5分）**
3. **PTE中的有效位为0，MMU触发缺页异常（1分）**
4. **缺页处理程序选择物理内存中的牺牲页（若页面被修改，则换出到磁盘）（1分）**
5. **缺页处理程序调入新的页面到内存，并更新PTE（1分）**
6. **缺页处理程序返回到原来进程，再次执行导致缺页的指令（1分）**
7. **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 );**

**}**

**攻击方法：采用基于缓冲区溢出攻击，让输入字符串s的字符个数大于32，**

**可导致getbuf函数返回到无关的代码处，或返回到指定的攻击**

**代码处。（3分）**

**（2分）修复:限制字符串操作的长度可编码缓冲区溢出的攻击，如：**

**int getbuf\_s(char \*s) {**

**char buf[32];**

**if(strlen(s)<sizeof(buf))**

**strcpy( buf, s );**

**}**

**（2分） 系统级的防范措施：栈空间地址的随机偏移、将栈stack标记为不可执行、**

**或在栈中某个位置放入特定的金丝雀值。**

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

**每步骤1分**

1. **在shell命令行中输入命令：$./hello**
2. **shell命令行解释器构造argv和envp；**
3. **调用fork()函数创建子进程，其地址空间与shell父进程完全相同，包括只读代码段、读写数据段、堆及用户栈等**
4. **调用execve()函数在当前进程（新创建的子进程）的上下文中加载并运行hello程序。将hello中的.text节、.data节、.bss节等内容加载到当前进程的虚拟地址空间**
5. **调用hello程序的main()函数，hello程序开始在一个进程的上下文中运行。**

**五、系统分析题（每小题5分，共20分）**

**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**

1. **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）**

**全局符号：x、y、z、main、addvec、addcnt（2分）**

**强符号：x、y、main、addvec、addcnt（1分）**

**若符号：z（1分）**

**x、y在.data节，z在.bss节，main.text节。（1分）**

**addvec在未定义节（UND），addcnt被优化掉（addvec、addcnt未说明不扣分）**

1. **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. **\_3c \_10 \_60 \_00 ②\_28 \_10 \_60 00\_**

**③\_30\_ 10 \_60 00 ④ b3 fe ff ff**

**⑤ 2d 09 20 00 ⑥ 23 09 20 00**

**⑦ a3 fe ff ff**

1. **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）**

**容量32kb：则64\*8\*8 = 4kB**

**所以：B=64（0.5分）, S=8（1分）, E=8（0.5分）; CO = 0x10（1分）,**

**CI=0x0（1分）,CT=0x3fb31a9008（1分）**

**若以32kB算：B=64，S=64，E=8；C0 = 0x10,CI=0x0,CT=0x7f6635201**

1. **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);**

**}**

**父进程：p1：x=2（1分）**

**p2：x=1**

**p2：x=0**

**子进程：p2：x=0（1分）**

**exit**

**printf**

**p2：x=1**

**X==1**

p1：x=2

**fork**

**exit**

**printf**

**printf**

**main**

进程图3分

**六、综合设计题（每小题10分，共20分）**

1. **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;**

**}**

**可采用一般有用的优化、面向编译器的优化、面向CPU的优化、面向Cache的优化。任意一种或结合都可，写出程序6分，说明理由4分。**

**注：如果程序或理由有缺陷，通过2种及以上方法补足，最多满分。**

1. **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)**

**指令mraddq D(rB)，rA的编码规则如下**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字节0** | | **字节1** | | **字节2...9** |
| **C** | **0** | **rA** | **rB** | **D** |

|  |  |  |  |
| --- | --- | --- | --- |
| **指令** | **mrmovq D(rB),rA** | **addq rA, rB** | **mraddq D(rB), rA** |
| **取指** | **icode:ifun←M1[PC]** | **icode:ifun←M1[PC]** | **设计思想：（3分）**  **方案1：不改变现有指令阶段（机器周期）个数。如可在执行阶段完成地址运算、访存和加法，会增加执行电路的复杂度。**  **方案2：在访存后增加一指令执行阶段**  **注：也可以写出微指令。** |
| **rA:rB←M1[PC+1]** | **rA:rB←M1[PC+1]** |
| **valC←M8[PC+2]** |  |
| **valP←PC+10** | **valP←PC+2（2分）** |
| **译码** |  | **valA←R[rA]** |
| **valB←R[rB]** | **valB←R[rB] （2分）** |
| **执行** | **valE←valB+valC** | **valE←valB + valA**  **Set CC（1分）** |
| **访存** | **ValM←M8[ValE]** |  |
| **写回** | **R[rA]←valM** | **R[rB]←valE（1分）** |
| **更新PC** | **PC←valP** | **PC←valP（1分）** |