2020~2021 学年第 2 学期期末考试《Linux 操作系统内核分析》试卷 甲（A）

题号	一	二	三	四	五	总 分
得分						

注：以下题目里的汇编指基于 IA32 的 AT&T 汇编格式

一、填空题 （每题 1 分，共 20 分）

1 IA-32 的 Linux 系统中，函数调用使用_stdcall 的方式，调用 f(x,y,z)时参数压栈，首先压入的是_____。

2 函数调用通常采用_____保存返回值。

3 进程的用户态执行时，EIP 寄存器中存放的是_____的地址，ESP 寄存器中存放的是_____的地址。

4 C 语言代码 eax = *(int*)0x231 对应的汇编指令是_____。它的寻址方式是_____。

5 Linux 源码 start_kernel 函数相当于 C 语言中的_____函数，在此函数调用前，内核代码用_____语言编写。。

6 Linux 将_____和进程的线程描述符 thread_info 组成一个联合体。

7 程序从源代码到可执行文件的步骤是预处理、_____、_____和链接。

8 静态链接程序被调度一开始就会执行_____函数，动态链接程序在执行该函数之前会执行_____。

9 系统调用指令 int $0x80 的 0x80 的含义是_____，系统进行响应，cpu 切换到内核态会从_____处开始执行指令。

10 Linux 系统里相当于 PCB 的是_____，其中 thread_struct 数据结构主要保存进程上下文中与_____相关的信息。

11 ELF 文件格式提供了两种视图，分别是_____和_____；在 ELF 文件中，描述文件节区信息的数据表是_____。

二 判断题（每题 1 分，共 10 分）

1 EIP 寄存器的内容不能被修改。

2 0 号进程、1 号进程、2 号进程都是系统启动时创建的系统进程，它们都运行在内核态下。

3 Linux 系统里的进程没有专门的运行态，TASK_RUNNING 状态既可以是就绪也可以正在执行。

4 CPU 响应中断时，会自动把中断描述符表里的一个地址装填到 CS：EIP 寄存器中，从而转入内核程序执行。

5 系统调用处理过程最后一条指令是 IRET。

6 进程执行系统调用进入内核态运行，需要通过用户态程序把用户态堆栈切换到内核态堆栈。

7 一个进程只能创建自己的子进程，不能创建自己的兄弟进程。

8 Linux 的内核是没有进程和线程的区分，线程也是用进程来模拟的。

9 当一个进程调用 fork()创建子进程成功后不一定马上返回父进程，而是大概率调度子进程先运行。

10 在 Linux 中，内核线程可进行主动调度，此时需要中断上下文的切换。

三、分析填空题（第 1 题 20 分，第 2 题 10 分，第 3 题 6 分，共 36 分）

1. 一个 C 语言文件 main.c 的源程序清单如下图左，其编译后的主要汇编代码程序如下图右。

```
int g(int x)
{
  return x ④
}

int f(int x)
{
  return ③+10;
}

int main(void)
{
  return f(①)②;
}
```

```
1   g:
2       pushl   %ebp
3       movl    %esp, %ebp
4       movl    8(%ebp), %eax
5       subl    $5, %eax
6       popl    %ebp
7       ret
8   f:
9       pushl   %ebp
10      movl    %esp, %ebp
11      subl    $4, %esp
12      movl    8(%ebp), %eax
13      movl    %eax, (%esp)
14      call    g
15      addl    ⑤, %eax
16      leave
17      ret
18  main:
19      pushl   %ebp
20      movl    %esp, %ebp
21      subl    $4, %esp
22      movl    $6, (%esp)
23      call    f
24      addl    $2, %eax
25      leave
        ret
```

(1).请对照汇编代码和 C 代码，填出其中的 5 个空①-⑤。（每空 1 分，共 5 分）

(2).继续对该汇编程序进行汇编和链接生成可执行程序，然后用 gdb 进行调试，在主函数 main 处设置断点，然后运行程序到断点，显示寄存器 esp 和 ebp 的值。请写出相应命令：1.汇编和链接成可执行程序的命令（2 分）；2.设置断点及运行的命令（4 分）；3.显示 esp 和 ebp 值的命令（4 分）。（共 10 分）

(3). 如果此时显示出 ebp=0xffffd488,esp=0xffffd488, 请画出程序执行到函数 g 返回之前（即执行到 g 里的 ret 处）时，函数调用堆栈空间图（假设汇编代码每条指令的地址即为其行号；需要标明 esp 和 ebp 的位置）。（共 5 分）

2. 以下不完整程序简单模拟 Linux 内核里进程的创建过程(10 分)：

```
…
1: int pid = 0;
2: int i;
3: task[pid].pid =pid;
4: task[pid].state =0;
5: task[pid].task_entry = task[pid].thread.ip = (unsigned long)my_process;
6: task[pid].thread.sp = (unsigned long)&task[pid].stack[KERNEL_STACK_SIZE-1];
7: task[pid]. next = &task[pid];
…
```

请补足程序创建剩余的三个进程，并把四个进程组织成双向循环链表。

3.在进程切换的时候，不仅要切换进程的虚拟地址空间，更重要的是要切换内核堆栈和 CPU 硬件上下文，这在实际的 linux 内核里是通过 switch_to 宏定义函数来实现的。一个简化的主要模拟代码如下所示，假设当前进程为 prev，要切换进来准备执行的进程为 next。请完成此

代码中的空白（2 分）；在这段代码中哪条关键语句完成了内核堆栈的切换？（1 分）；到哪条语句完成了 CPU 里 EIP 的切换？（1 分）；next 进程如果是曾经执行过进程，现在要重新被调度起来执行，从哪个位置开始？（1 分）；函数 switch_to 的第三个参数 last 起什么作用？（1 分）

```
#define switch_to(prev,next,last)
 do{
 asm volateile(
   “pushfl \n\t”
   “pushl  %%ebp \n\t”
   “movl  %%esp, %[prev_sp]\n\t”
   “movl   %[next_sp], %%esp\n\t”
   “movl   $1f, %[prev_ip]\n\t”
   “pushl  %[next_ip]\n\t”
   “jmp    __switch_to\n\t”
   “1:\t”
   “①”
   “②”
```

```
/* output parameters */
: [prev_sp]
"=m"(prev->thread.sp),
 [prev_ip] "=m"(prev->thread.ip),
"=a" (last),
/* input parameters: */
: [next_sp]  "m"
(next->thread.sp),
 [next_ip]   "m"
(next->thread.ip),
 [prev]        "a" (prev),
 [next]        "d" (next)
…
}while(0)
```

四、理解编程题（每题 17 分，共 34 分）

1. 现有两个 C 语言程序（分别是 m.c 和 f.c）如下：

```
/* m.c */
    int a = 1;
    int b = 2;
    extern int sub();
    void main()
     {
     int s;
     s = sub(a, b);
     }
```

```
/* f.c */
int sub(int i, int j)
     {
     return i - j;
     }
```

(1).假定 m.c 和 f.c 分别被编译成目标文件 m.o 和 f.o 后，再通过链接的方式生成可执行文件 test.out。请写出相关实现的 gcc 命令。（2 分）

(2).若 m.c 编译时得到的汇编程序如下：

```
00000000 <main>:
 0:   55                          push   %ebp
 1:   89 e5                       mov    %esp,%ebp
 3:   83 ec 04                    sub    $0x4,%esp
 6:   a1 00 00 00 00              mov    0x0,%eax
 7: R_386_32     j
 b:   50                          push   %eax
 c:   a1 00 00 00 00              mov    0x0,%eax
 d: R_386_32     i
 11:  50                          push   %eax
 12:  e8 fc ff ff ff              call   13 <main+0x13>
 13: R_386_PC32  sum
 17:  83 c4 08                    add    $0x8,%esp
 1a:  89 c0                       mov    %eax,%eax
 1c:  89 45 fc                    mov    %eax,0xfffffffc(%ebp)
 1f:  c9                          leave
 20:  c3                          ret
```

用 objdump -x m.o 命令显示目标文件 m.o 文件的内容部分如下

```
……
SYMBOL TABLE:
……
00000000 g    O .data   00000004 a
00000004 g    O .data   00000004 b
00000000 g    F .text   00000021 main
00000000        *UND*   00000000 sub
RELOCATION RECORDS FOR [.text]:
OFFSET      TYPE            VALUE
00000006    R_386_32         b
0000000e    R_386_32         a
00000012    R_386_PC32       sub
```

请回答 SYMBOL TABLE 和 RELOCATION RECORDS 的作用是什么。从 SYMBOL TABLE 中看出 main 和 sum 有什么不同，RELOCATION RECORDS 中 OFFSET 项记录的数据的作用是什么。（5 分）

(3).汇编程序标号 12 处的 e8 是什么，fc ff ff ff 是什么？这条指令执行时 EIP 寄存器存放的是什么？（5 分）

(4).程序经过链接后 main 在可执行文件中的位置为 080482c2，sum 在可执行文件中的位置为 080482ea，请给出 m.o 中标号 12 和 17 两条指令的变化（包含标号），并给出解释。（5 分）

2. 阅读以下程序，回答问题。(17 分)

```
int main(int argc, char * argv[])
{ int pid;
pid = fork();        /* fork another process */
if (pid<0)
{
   fprintf(stderr, “Fork Failed!”);    /*error
occurred*/
   exit(-1);
  }
else if (pid==0)
{
   execlp("/bin/ls","ls",NULL);
   printf（"Child running!"）
  }
else
   {
      wait(NULL);
      printf("Child Complete!");
      exit(0);
     }
}
```

(1). 上述 C 程序里系统调用 fork()调用成功后，子进程第一次被调度将会从哪一条语句执行，这个时候 EAX 寄存器的内容是什么。（解释原因）（5 分）

(2). 上述程序实际运行时，execlp 这个系统调用会起什么作用？其会传递什么参数？怎么传递的？请分别给出父进程和子进程的输出结果。（7 分）

(3).已知 fork（）系统调用号为 2，用内嵌汇编的方式实现该系统调用。（4 分）（注：只需要写出内嵌汇编代码，参数传递采用 16 进制）（5 分）

2020~2021 学年第 2 学期期末考试《Linux 操作系统内核分析》试卷 甲（A）

标准答案和评分标准

一、填空题 （每空 1 分，共 20 分）

1、Z
2、EAX 寄存器
3、下一条指令、用户态堆栈栈顶
4、movl 0x231,%eax,直接寻址
5、main，汇编
6、内核堆栈
7、编译，汇编
8、main，动态链接器
9、128 号中断，中断处理程序
10、TASK_STRUCT,处理器
11、节，段，节头表

二 判断题（每题 1 分，共 10 分）

1、×2、× 3、√ 4、√ 5、√ 6、× 7、×8、√9、√10、×

三、分析填空题

1、（1）6，+2，g(x)，-5，$10 **评分标准：每空 1 分，共 5 分**

（2）1、gcc main.c
2、b main，r
3、i r esp，i r ebp **评分标准：第 1 问正确 2 分，第 2 问写一个命令 2 分；共 4 分，第 3 问写一个命令 2 分；共 4 分。**

（3）1、popl %%ebp\n\t
popfl\n\t **评分标准：每空 1 分，共 2 分**
2、movl %[next_sp], %%esp\n\t **评分标准：每问 1 分，共 1 分**
3、jmp __switch_to\n\t **评分标准：每问 1 分，共 1 分**
4、1:\t **评分标准：每问 1 分，共 1 分**

四、理解编程题

1、（1）gcc -c m.c -o m.o ， gcc -c f.c -o f.o （2 分）

（2）SYMBOL TABLE 为符号表，用于符号解析。RELOCATION RECORDS 为重定位表，用于链接过程的地址重定位（2 分）

main 为在文件中有定义 sum 为文件中未定义，引用其他文件中的符号

OFFSET 项记录的数据是确定需要修改的地址在文件中的位置（3 分）

(3) call 指令的操作码（2 分），编译器对不能确定的地址设定的虚值（2 分），17（1 分）

（4）080482d4: e8 11 00 00 00 (2 分)

080482d9: 83 c4 08 (2 分)

080482d4=080482c2+12 ， 080482d9=080482c2+17

080482ea-080482d9=00000011 执行 call 指令时，EIP 的内容为 080482d9，因为 call 采用相对寻址，sum 的地址减去 080482d9 就是链接器计算出的偏移地址 （1 分）

2、（1）从 if(pid<0)处执行，（1 分） EAX 寄存器的内容为 0（1 分）pid 被编译器设定为寄存器变量，用 EAX 存放，系统调用执行时用它传递系统调用号，返回时用它存放返回值。子进程的 pid 返回值为 0 （3 分）

（2）execlp 这个系统调用加载一个可执行程序替换父进程的程序，（2 分）它传递可执行程序路径，命令指针，命令执行参数，（2 分）通过寄存器传递内存地址，（1 分）父进程的输出结果是"Child Complete!"，（1 分）子进程的输出结果是当前目录的文件详细列表信息（1 分）

（3）int pid;

```
asm volatile (
    “movl $0x2,%%eax\n\t”
    “int $0x80\n\t”
    “movl %%eax,%0\n\t”
   :”=m”(pid)
  );
```

评分标准：完全正确得 5 分，少 1 步骤扣 1 分。