题号	一	二	三	四	五	六	七	八	九	十	十一	十二	十三	十四	十五	十六	十七	十八	十九	二十	总 分
得分																					

2021~2022 学年《Linux 内核分析》试卷（甲）A

注：以下题目基于 x86-32 的 AT&T 汇编格式

一 填空题（15’）

1 汇编指令 mov1 0x123,%ebx 执行后放到 EBX 寄存器里的数据是 _____。

2 如果用一条指令建立函数堆栈，这条指令是_____。如果用一条指令拆除函数堆栈，这条指令是_____。

3 所有用户态进程的祖先进程是_____，负责所有内核线程调度管理的进程是_____。

4 Linux 下的用户态程序运行在处理器的_____级别。此时 CS 寄存器的低两位是_____。

5 中断处理程序中的 SAVE_ALL 指令是把一些通用寄存器的值放入_____中。这个过程被称为_____。

6 应用编程接口函数的英文名称是_____。系统调用是通过_____向内核发出请求。

7 TASK_STRUCT 中的 state 是_____，files_struct 指向的数据结构是_____。

8 在 ELF 文件中，描述文件节区信息的数据结构是_____，和创建进程相关的数据结构是_____。

二 简答题（20’）

1 程序从源代码到可执行程序的步骤有哪些，每个步骤对应的命令是什么？（以 hello.c 为例）

2 进程堆栈的作用是什么，一个进程有哪几个堆栈，分别处于进程的什么位置？

三 应用题（65’）

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 add(int x,int y)
   {
    return ⑤;
   }

int main ( )
   {
    int t1 = 120;
    int t2 = 60;
    int sum=add(t1,t2);
    return sum;
    }
```

```
    pushl  %ebp
    movl   %esp , %ebp
    subl   ①, %esp
    movl   $120, ②(%ebp)
    movl   $60, ③(%ebp)
    movl   ③(%ebp), %eax
    movl   %eax, 4(%esp)
    movl   ②(%ebp), %eax
    movl   %eax, (%esp)
    call   add
    movl   %eax, ④(%ebp)
    movl   ④(%ebp), %eax
    movl   %ebp,%esp
    popl   %ebp
    ret
```

(1)请对照汇编代码和 C 代码，填出其中的 4 个空①-⑤。（每空 1 分，共 5 分）

(2)设程序开始前，EBP 和 ESP 的内容为 0xffffd488，call 语句执行之前，EBP 和 ESP 的内容分别为多少，栈顶内容是多少。（要求给出当时的栈帧布局）（10 分）

2、下面是一段进程上下文切换的内嵌汇编程序，（prev 为将要被切换的进程，next 为下一个被选中的进程）

```
asm volatile(
        "pushl %%ebp\n\t"
        "movl %%esp,%0\n\t"
        "movl %2,%%esp\n\t"
        "movl $1f,%1\n\t"
        "pushl %3\n\t"
        "ret\n\t"
        "1:\t"
        "popl %%ebp\n\t"
        : "=m" (prev->thread.sp),"=m" (prev->thread.ip)
        : "m" (next->thread.sp),"m" (next->thread.ip)
```

```
        );
```

（1） prev->thread 的作用是什么，为什么它只需要两个成员 thread.sp 和 thread.ip，而不需要 thread.bp 保存 ebp 的值。（5 分）
（2） 程序中是通过哪两条语句完成向新进程的程序转移，它能被跳转指令替换吗，如果不能，为什么？（5 分）
（3） prev 进程下一次被调度将会从哪条语句执行，进程是如何记住这个位置？（5 分）

3、阅读以下程序，回答问题。

```
int main(int argc, char * argv[])
{
int pid;    /* fork another process */
pid = fork();
if (pid<0)
{
  exit(-1);
 }
else if (pid==0)
{
  execlp("/bin/ps","ps",NULL);
  printf ("Child start!")
 }
else
  {
    wait(NULL);
    printf("Child Complete!");
    exit(0);
   }
}
```

（1）在 fork()执行过程中，eax 寄存器承担什么角色，有什么变化。（5 分）
（2）已知 fork（）系统调用号为 2，对 pid = fork()用内嵌汇编的方式实现该系统调用。（5 分）
（3）子进程第一次被调度将会从哪儿开始执行，又会在哪条语句后有了独立的不同于父进程的虚拟程序空间。（5 分）

4、现有两个 C 语言程序（分别是 m.c 和 f.c）如下：

```
/* m.c */
    int e1[2]={1,2};
    int b = 1;
    extern int *diff();
    void main()
     {
     int s[2];
     s = diff(e1, b);
     }
```

```
/* f.c */
int *diff(int a[2], int j)
  {
     a[0]=a[0]-j
     a[1]=a[1]-j
     return a;
  }
```

(1)用 objdump -dx m.o 命令显示目标文件 m.o 后的部分内容如下所示：左图为部分反汇编代码：右图为其它信息。

```
00000000 <main>:
   0:   55                      push   %ebp
   1:   89 e5                   mov    %esp,%ebp
   3:   83 e4 f0                and    $0xfffffff0,%esp
   6:   83 ec 20                sub    $0x20,%esp
   9:   a1 00 00 00 00          mov    0x0,%eax
            a: R_386_32 base
   e:   89 44 24 04             mov    %eax,0x4(%esp)
  12:   c7 04 24 00 00 00 00    movl   $0x0,(%esp)
            15: R_386_32    elm
  19:   e8 fc ff ff ff          call   1a <main+0x1a>
            1a: R_386_PC32  diff
  1e:   89 44 24 1c             mov    %eax,0x1c(%esp)
  22:   c9                      leave
  23:   c3                      ret
```

```
SYMBOL TABLE:
00000000 l    df *ABS*  00000000 m.c
00000000 l    d  .text  00000000 .text
00000000 l    d  .data  00000000 .data
00000000 l    d  .bss   00000000 .bss
00000000 l    d  .note.GNU-stack    00000000 .note.GNU-stack
00000000 l    d  .eh_frame  00000000 .eh_frame
00000000 l    d  .comment   00000000 .comment
00000000 g     O .data  00000008 elm
00000008 g     O .data  00000004 base
00000000 g     F .text  00000024 main
00000000         *UND*  00000000 diff

RELOCATION RECORDS FOR [.text]:
OFFSET   TYPE              VALUE
0000000a R_386_32          base
00000015 R_386_32          elm
0000001a R_386_PC32        diff
```

请回答反汇编程序标号 19 处的 e8 是什么意思，后面的 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）
(3)如果对可执行文件 test 采用 objdump -dj.text 进行反汇编显示，在 main 函数中观察到 call 指令处开始地址为 08048406，而 diff 地址为 08048411。请给出此时 test 文件中调用 diff 的 call 指令的完全机器码和相应的汇编语句，并给出解释。（5 分）