1. **教材6.6 下面是C语言两个函数f和g的概略（它们不再有其它的局部变量）：**

```
int f (int x) { int i; …return i +1; … }
int g (int y) {int j; … f (j +1); … }
```

请按照图6.11的形式，画出函数g调用f，f的函数体正在执行时，活动记录栈的内容及相关信息，并按图6.10左侧箭头方式画出控制链。假定函数返回值是通过寄存器传递的。

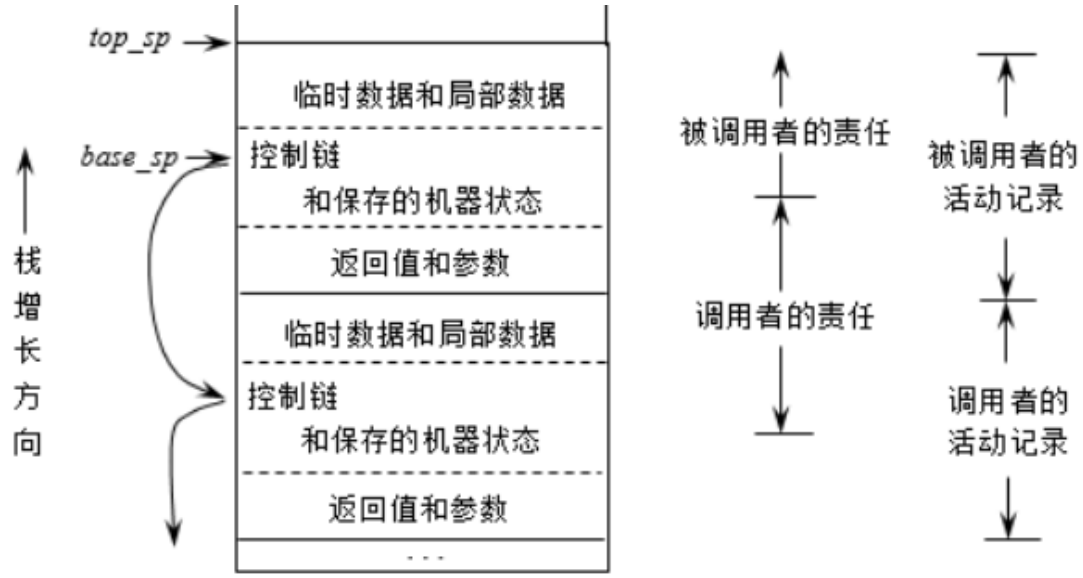


图 6.10 调用者和被调用者之间的任务划分

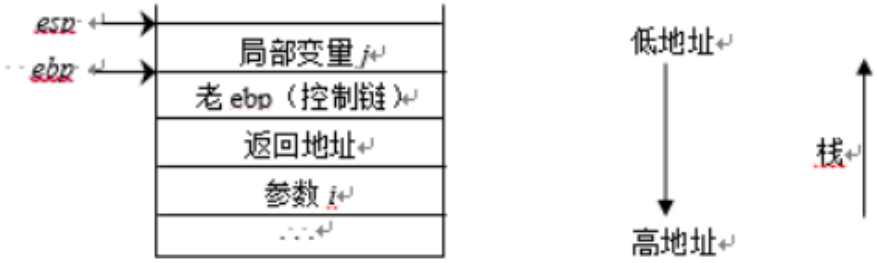
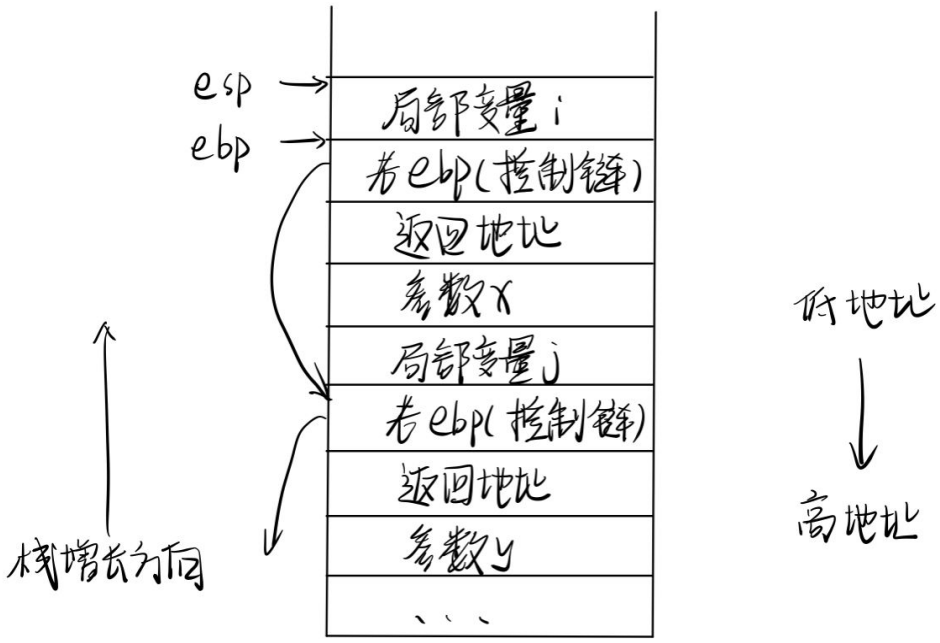


图 6.11 活动记录的内容及相关信息



2. **教材6.18 下面是一个C语言程序：**

```
#include <stdio.h>
int main() {
     long i;
     long a[0][4];
     long j;
     i = 4; j = 8;
     printf("%ld, %d\n", sizeof(a), a[0][0]);
 }
```

虽然出现long a[0][4]这样的声明，但在x86/Linux系统上，用编译器GCC 7.5.0 (Ubuntu 7.5.0-3ubuntu1~16.04)编译时，该程序能够通过编译并生成目标代码。请在你自己的机器上实验，回答下面两个问题（说明你使用的编译器及版本并给出汇编码）：

(a) sizeof(a)的值是多少，请说明理由。

根据数组大小的计算公式，0*4*8=0

(b) a[0][0]的值是多少，请说明理由。

-402910656，编译时出现了warning，因为a[0][0]是一个long int类型，而printf给出的输出格式为%d

```
xw@xw-virtual-machine:~/expr$ gcc hw.c -o hw
hw.c: In function ‘main’:
hw.c:8:18: warning: format ‘%d’ expects argument of type ‘int’, but argument 3 h
as type ‘long int’ [-Wformat=]
    8 |     printf("%ld,%d\n",sizeof(a),a[0][0]);
      |                 ~^               ~~~~~~~
      |                  |                  |
      |                  int                long int
      |                 %ld
xw@xw-virtual-machine:~/expr$ ./hw
0,-402910656
```

编译器：gcc

版本：

```
xw@xw-virtual-machine:~/expr$ cat /proc/version
Linux version 5.11.0-40-generic (buildd@lgw01-amd64-010) (gcc (Ubuntu 9.3.0-17u
buntu1~20.04) 9.3.0, GNU ld (GNU Binutils for Ubuntu) 2.34) #44~20.04.2-Ubuntu
SMP Tue Oct 26 18:07:44 UTC 2021
```

汇编码：

```
    .file   "hw.c"
    .text
    .section    .rodata
.LC0:
    .string "%ld,%d\n"
    .text
    .globl  main
    .type   main, @function
main:
.LFB0:
    .cfi_startproc
    endbr64
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    subq    $32, %rsp
    movq    %fs:40, %rax
    movq    %rax, -8(%rbp)
    xorl    %eax, %eax
    movq    $4, -32(%rbp)
    movq    $8, -24(%rbp)
    movq    -16(%rbp), %rax
    movq    %rax, %rdx
    movl    $0, %esi
    leaq    .LC0(%rip), %rdi
    movl    $0, %eax
    call    printf@PLT
    movl    $0, %eax
    movq    -8(%rbp), %rcx
    xorq    %fs:40, %rcx
    je  .L3
    call    __stack_chk_fail@PLT
.L3:
    leave
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE0:
    .size   main, .-main
    .ident  "GCC: (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0"
    .section    .note.GNU-stack,"",@progbits
    .section    .note.gnu.property,"a"
    .align 8
    .long    1f - 0f
    .long    4f - 1f
    .long    5
0:
    .string  "GNU"
1:
```

```
    .align 8
    .long    0xc0000002
    .long    3f - 2f
2:
    .long    0x3
3:
    .align 8
4:
```

3. **对于如下C程序**

```
int main()
{
     char *cp1, *cp2;

     cp1 = "12345";
     cp2 = "abcdefghij";
     strcpy(cp1,cp2);
     printf("cp1 = %s\ncp2 = %s\n", cp1, cp2);
}
```

1）在某些系统上的运行结果是：

```
 cp1 = abcdefghij
 cp2 = ghij
```

为什么cp2所指的串被修改了？

因为"12345"、"abcdefghij"放在连续存储的区域，执行strcpy后变成了如下状态：

```
(cp1所指向的位置)abcdef(cp2所指向的位置)ghij\0fghij\0
```

故cp1=abcdefghij，cp2=ghij

2）在某些系统上运行会输出段错误，为什么？

因为"12345"、"abcdefghij"放在字符常量区，cp1、cp2分别指向这个区域，而这个区域是一个const属性，是不可以修改的，使用strcpy会造成写一个只读内存空间，故会造成段错误。