西北工业大学
NORTHWESTERN POLYTECHNICAL UNIVERSITY

C程序设计

Programming in C

1011014

主讲：姜学锋，计算机学院

学会用回调函数

1、回调函数

2、函数同步调用和异步调用

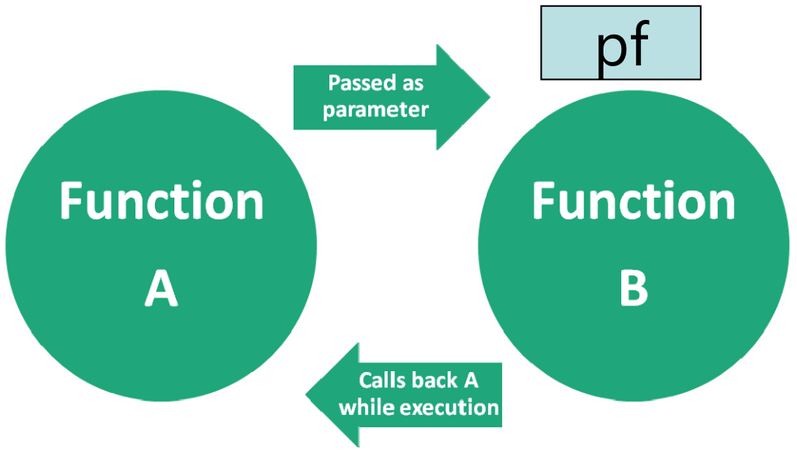
3、钩子函数和HOOK技术

7.5.3 函数指针

- 函数指针变量可以作为函数形参，此时实参要求是函数名（即函数地址），或者是函数指针，而形参和实参有相同的返回类型、参数个数、参数类型（即函数原型相同）。

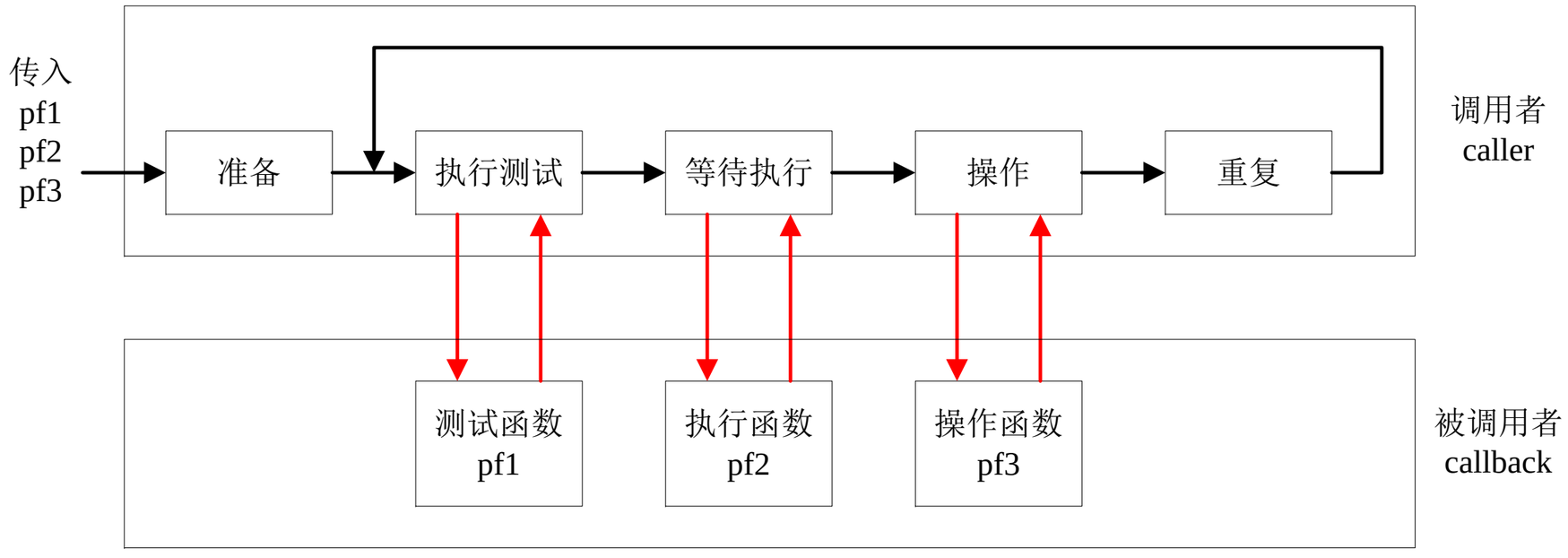
7.5.3 函数指针

- 1．回调函数
- 把一个函数callback的指针（地址）pf作为参数传递给另一个函数caller，并通过函数指针pf调用callback函数，称callback函数为回调函数。简而言之，回调函数就是一个通过函数指针调用的函数。



7.5.3 函数指针

- 使用回调函数可以把调用者caller和被调用者callback分开，调用者中间不是固定地调用哪个函数，而是固定地通过函数指针调用函数，究竟是哪个函数由传到caller的值来决定。



7.5.3 函数指针

【例7.24】

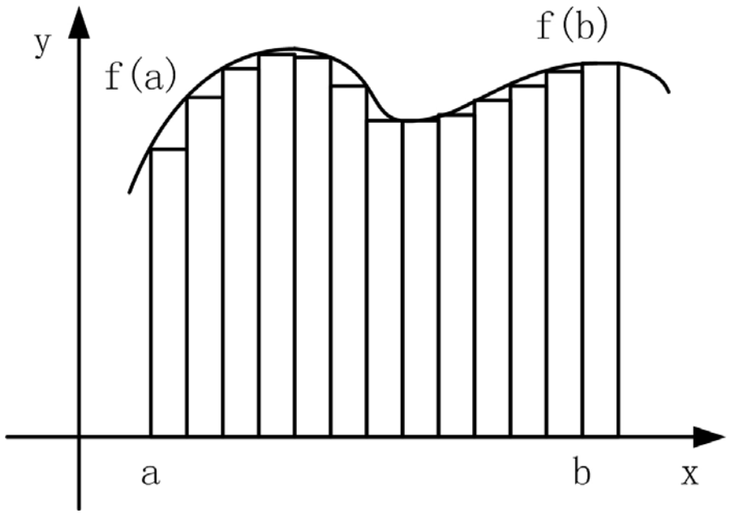
编写程序计算如下公式。

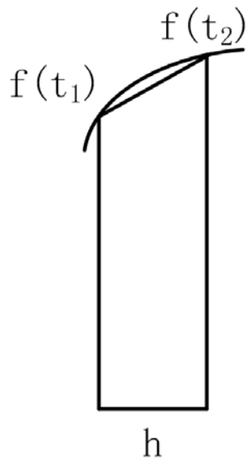
$$\int_a^b (1+x)dx + \int_a^b e^{-\frac{x^2}{2}}dx + \int_a^b x^3 dx$$

7.5.3 函数指针

例题分析

这里用梯形法求定积分 $\int_a^b f(x)dx$ 的近似值。如图所示，求 $f(x)$ 的定积分就是求 $f(x)$ 曲线与x轴包围图形的面积，梯形法是把所要求的面积垂直分成n个小梯形，然后面积求和。





7.5.3 函数指针

例题分析

根据上述思想编写函数integral，由于需要计算多个不同 $f(x)$ 的值，因此向integral传递 $f(x)$的函数指针，由integral调用具体的 $f(x)$求值，其函数原型为：

```
double integral(double a,double b,double (*f)(double x));
```

7.5.3 函数指针

例7.24

```
1 #include <stdio.h>
2 #include <math.h>
3 double integral(double a,double b,double (*f)(double x))
4 { //求定积分
5   int n=1000, i;
6   double h, x, s=0.0;
7   h=(b-a)/n;
8   for(i=1;i<=n;i++) {
9     x=a+(i-1)*h;
10    s=s+(f(x)+f(x+h))*h/2; //调用f函数求f(x)、f(x+h)
11  }
12  return s;
13 }
14 double f1(double x)
15 { return 1+x;
```

7.5.3 函数指针

例7.24

```
16 }
17 double f2(double x)
18 { return exp(-x*x/2);
19 }
20 double f3(double x)
21 { return x*x*x;
22 }
23 int main()
24 {
25   double t,a,b;
26   scanf("%lf%lf",&a,&b);
27   t=integral(a,b,f1)+integral(a,b,f2)+integral(a,b,f3);
28   printf("%lf\n",t);
29   return 0;
30 }
```

7.5.3 函数指针

- 编程：写一个选择排序函数供他人调用，其中必包含比较大小和交换。

- 麻烦来了：此时并不知要比较的是何种类型数据--整数、浮点数、字符串？于是只好为每类数据制作一个不同的排序函数。

7.5.3 函数指针

- 更通行的办法是在排序函数参数中列2个回调函数，并通知调用者：你自己准备一个比较函数，其中包含两个指针类参数，函数比较此二指针所指数据之大小，并由函数返回值说明比较结果，以及自己准备一个交换函数。

- 排序函数借此调用者提供的函数来比较大小，借指针传递参数，可以全然不管所比较的数据类型。被调用者回头调用调用者的函数，即回调。

7.5.3 函数指针

【回调函数示例】

用回调函数来完成多种数据类型的选择排序。

```
void SelectionSort(int A[],int n) //选择排序 n为数组元素个数
```

原先：仅int型

现在：支持int、double、char *str[]字符串数组

7.5.3 函数指针

例题分析

在前面设计的选择排序函数SelectionSort中：

```
1 for(i=0; i<n-1; i++) { //选择排序法
2   k=i;
3   for(j=i+1; j<n; j++) //一趟选择排序
4     if (A[j] < A[k]) k=j; //<升序 >降序
5   if(i!=k)
6     t=A[i], A[i]=A[k], A[k]=t;
7 }
```

7.5.3 函数指针

例题分析

为了对任意类型排序，用回调函数来实现比较和交换，原型如下：

```
1 for(i=0; i<n-1; i++) { //选择排序法
2   k=i;
3   for(j=i+1; j<n; j++) //一趟选择排序
4     if ( compare(A[j],A[k]) ) k=j;
5   if(i!=k)
6     swap(A[i],A[k]);
7 }
```

7.5.3 函数指针

例题分析

为了选择排序函数SelectionSort支持任意类型，函数接口如下：

```
void SelectionSort(const void *A,int n,int size,
int (*compare)(void*,void*),void (*swap)(void*,void*))
```

A：设计成void*类型，支持任意类型
n：数组元素数目
size：由于是任意类型，需要给出元素的内存大小
compare：回调函数指针，根据给定条件比较两个元素，返回真排序
swap：回调函数指针，交换两个元素
compare和swap回调函数两个参数均为void*类型，支持任意类型

7.5.3 函数指针

例题分析

compare回调函数原型如下：

```
int compare(void *a, void *b)
{
   //根据将a,b指针转换为实际类型指针
   //对*a,*b作比较，升序选择<，降序选择>
   //条件成立返回1，否则返回0
}
```

7.5.3 函数指针

例题分析

compare回调函数原型如下：

```
void swap(void *a, void *b)
{
   //根据将a,b指针转换为实际类型指针
   //交换*a,*b
}
```

7.5.3 函数指针

例7.52

```
 1 #include <stdio.h>
 2 #include <string.h>
 3 void SelectionSort(const void *A,int n,int size,int (*comp
are)(void*,void*),void (*swap)(void*,void*))
 4 { //选择排序 n为数组元素个数
 5   int i,j,k;
 6   for(i=0; i<n-1; i++) { //选择排序法
 7     k=i;
 8     for(j=i+1; j<n; j++) //一趟选择排序
 9       if (compare((char*)A+j*size,(char*)A+k*size)) //A[j],
A[k]
10         k=j;
11     if(i!=k)
12       swap((char*)A+i*size,(char*)A+k*size);//A[i],A[k]
13   }
```

7.5.3 函数指针

例7.52

```
14 }
15 int compare_int(void *a, void *b)
16 { //升序，由小到大 a,b为int指针
17   if(*(int*)a < *(int*)b) return 1;
18   return 0;
19 }
20 void swap_int(void *a, void *b)
21 { //a,b为int指针
22   int t;
23   t=*(int*)a,*(int*)a=*(int*)b,*(int*)b=t;
24 }
25 int compare_double(void *a, void *b)
26 { //降序，由大到小 a,b为double指针
27   if(*(double*)a > *(double*)b) return 1;
28   return 0;
```

7.5.3 函数指针

例7.52

```
29 }
30 void swap_double(void *a, void *b)
31 { //a,b为double指针
32   double t;
33   t=*(double*)a,*(double*)a=*(double*)b,*(double*)b=t;
34 }
35 int compare_string(void *a, void *b)
36 { //降序，由大到小 a,b为char*指针
37   if(strcmp(*(char**)a,*(char**)b)>0) return 1;
38   return 0;
39 }
40 void swap_string(void *a, void *b)
41 { //a,b为char*指针
42   char *t;
43   t=*(char**)a,*(char**)b=*(char**)b,*(char**)b=t;
```

7.5.3 函数指针

例7.52

```
44 }
45 int main()
46 {
47   int i;
48   int A[15]={32,2,6,-1,0,8,1,4,21,99,7,5,-7,-10,17};
49   double D[10]={3.2,2.5,6.1,-1.2,0.5,1.8,4.6,2.1,9.9,7.5};
50   char *S[5]={"Pascal","C++","Python","Java","Basic"};
51   //int排序
52   SelectionSort(A,15,sizeof(int),compare_int,swap_int);
53   for(i=0; i<15; i++) printf("%d ", A[i]);
54   printf("\n");
55   //double排序
56   SelectionSort(D,10,sizeof(double),compare_double,swap_do
uble);
57   for(i=0; i<10; i++) printf("%.1lf ", D[i]);
```

7.5.3 函数指针

例7.52

```
 58    printf("\n");
 59    //char *排序
 60    SelectionSort(S,5,sizeof(char*),compare_string,swap_stri
ng);
 61    for(i=0; i<5; i++) printf("%s ", S[i]);
 62    return 0;
 63 }
```

7.5.3 函数指针

例7.52

程序运行屏幕

```
-10 -7 -1 0 1 2 4 5 6 7 8 17 21 32 99
9.9 7.5 6.1 4.6 3.2 2.5 2.1 1.8 0.5 -1.2
Python Pascal Java C++ Basic
```

7.5.3 函数指针

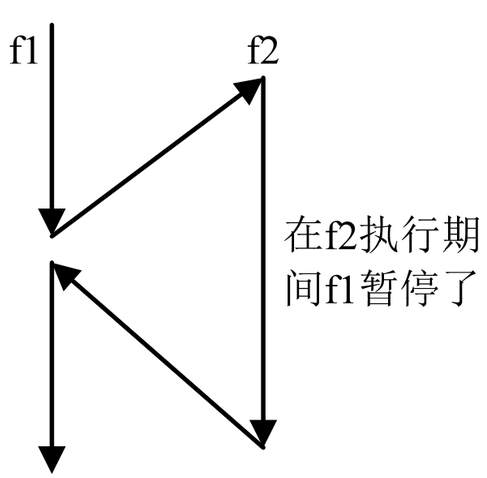
- 2．函数同步调用和异步调用
- 软件模块之间总是存在着一定的接口，从调用方式上，可以把它们分为三类：同步调用、回调和异步调用。

7.5.3 函数指针

- （1）同步调用是一种阻塞式调用，调用方要等待对方执行完毕才返回，它是一种单向调用；

```
PlaySound("WindowsXPStart.wav",NULL,SND_SYNC);
```

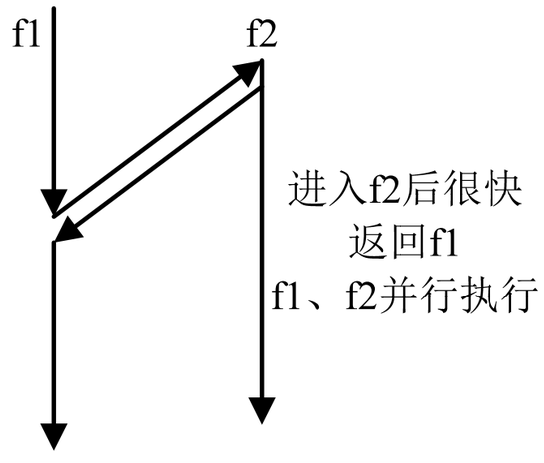
函数同步调用模式



- （2）异步调用是一种类似消息或事件的机制，不过它的调用方向刚好相反，接口的服务在收到某种讯息或发生某种事件时，会主动通知客户方（即调用客户方的接口）。

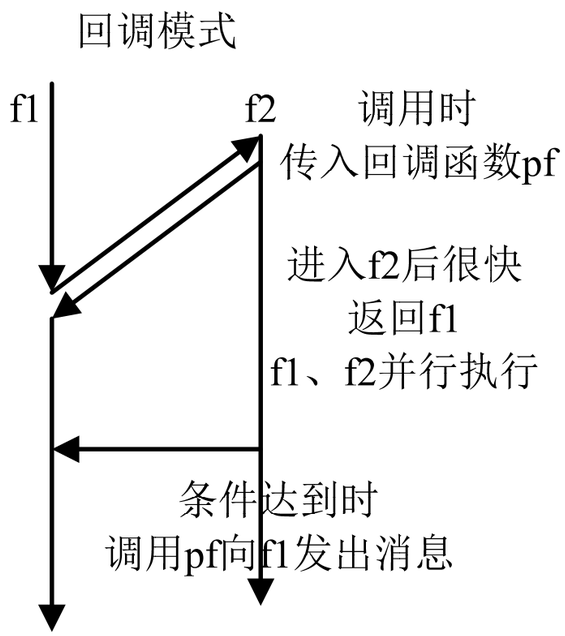
```
PlaySound("WindowsXPStart.wav",NULL,SND_ASYNC);
```

函数异步调用模式



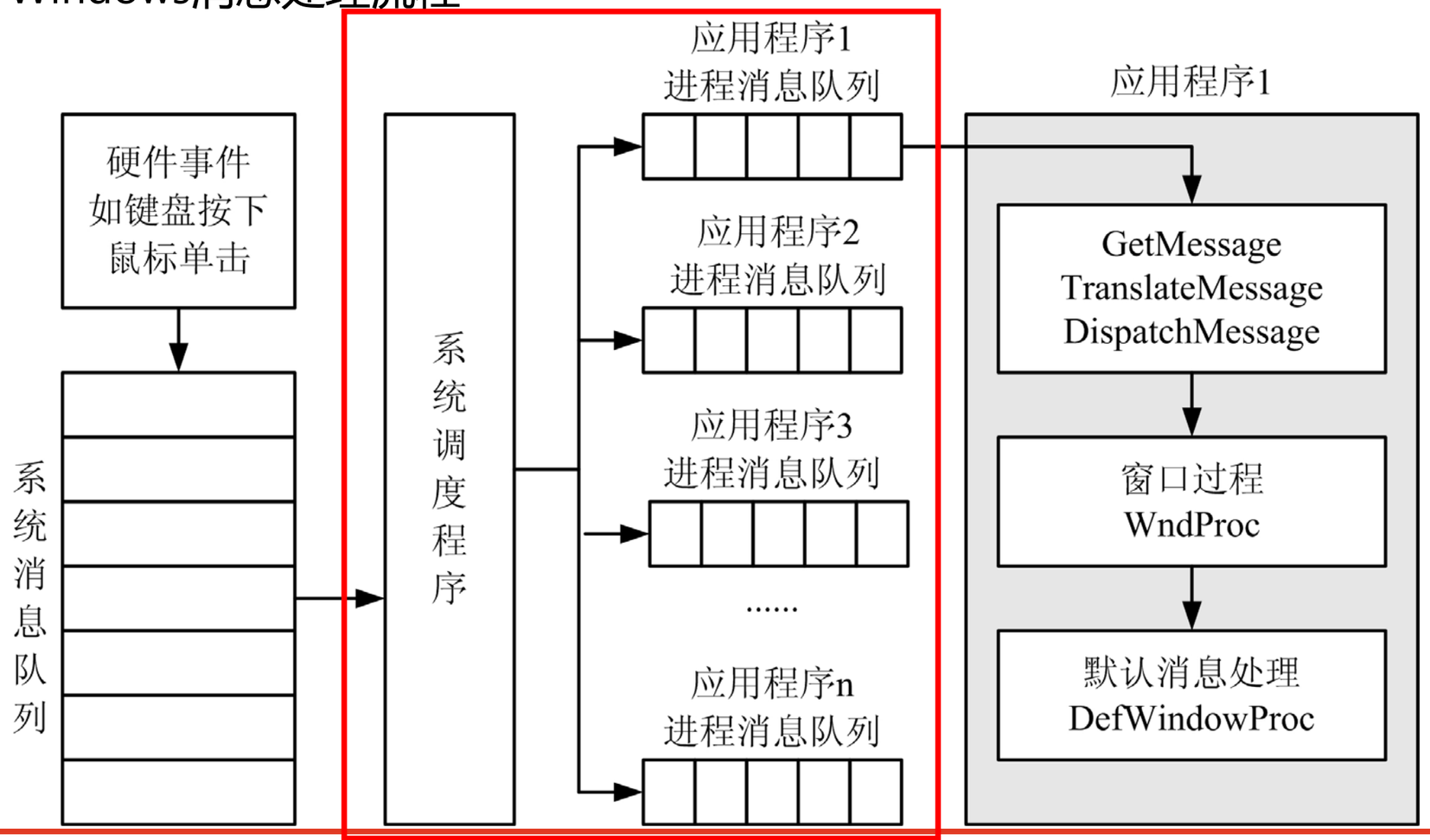
7.5.3 函数指针

- （3）回调是一种双向调用模式，也就是说，被调用方在接口被调用时也会调用对方的接口；



7.5.3 函数指针

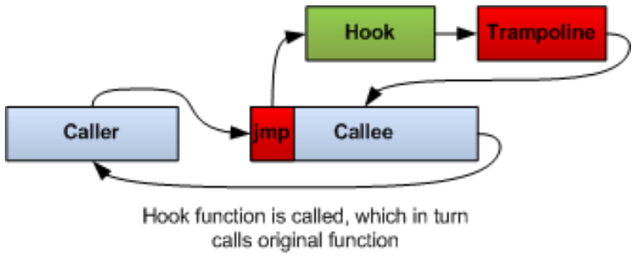
图13.6 Windows消息处理流程

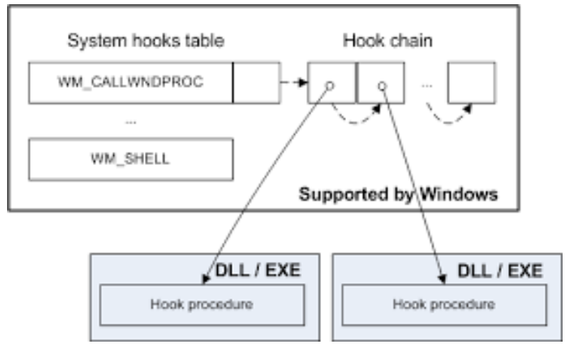


7.5.3 函数指针

- 回调和异步调用的关系非常紧密，通常使用回调来实现异步消息的注册，通过异步调用来实现消息的通知。

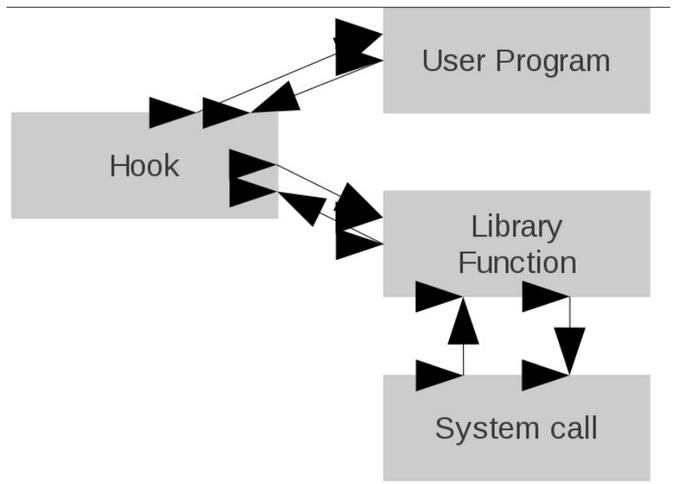
- 3．钩子函数和HOOK技术
- 钩子函数是特殊的回调函数。通过“钩挂”，可以给WINDOWS一个处理或过滤事件的回调函数，该函数就称为“钩子函数”，每当发生事件时，WINDOWS都将调用该函数。





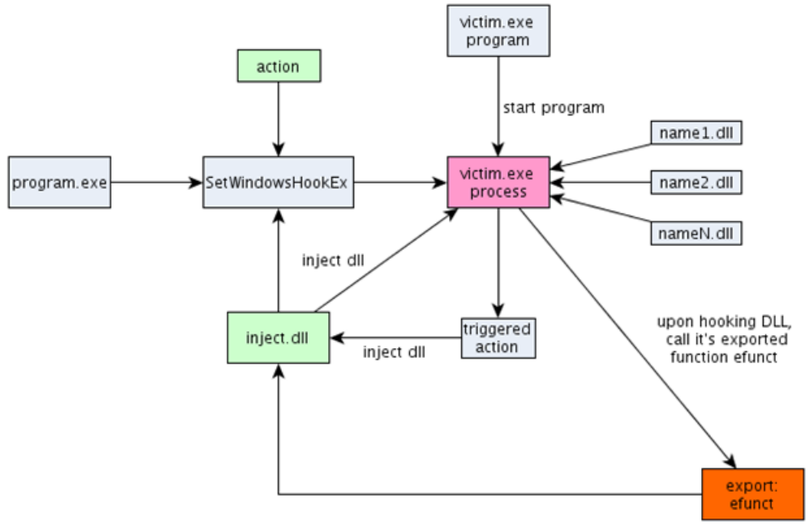
7.5.3 函数指针

- 钩子(Hook)技术，是Windows消息处理机制的一个平台,应用程序可以在上面设置子程序以监视指定窗口的某种消息，而且所监视的窗口可以是其他进程所创建的。当消息到达后，在目标窗口处理函数之前处理它。钩子机制允许应用程序截获处理window消息或特定事件。



7.5.3 函数指针

- 每当特定的消息发出，在没有到达目的窗口前，钩子程序就先捕获该消息，亦即钩子函数先得到控制权。这时钩子函数即可以加工处理（改变）该消息，也可以不作处理而继续传递该消息，还可以强制结束消息的传递。




程序设计