

#include<stdio.h>

#include <malloc.h>

void print(char \*,int);

int main()

{

char \*s1 = "abcde";

//"abcde"作为字符串常量存储在常量区 s1、s2、s5拥有相同的地址

char \*s2 = "abcde";

char s3[] = "abcd";

long int \*s4[100];

char \*s5 = "abcde";

int a = 5;

int b =6; //a,b在栈上，&a>&b地址反向增长

printf("variables address in main function: s1=%p s2=%p s3=%p s4=%p s5=%p a=%p b=%p \n",

s1,s2,s3,s4,s5,&a,&b);

printf("variables address in processcall:n");

print("ddddddddd",5);

//参数入栈从右至左进行,p先进栈,str后进 &p>&str

printf("main=%p print=%p \n",main,print);

//打印代码段中主函数和子函数的地址，

编译时先编译的地址低，后编译的地址高main<print

}

void print(char \*str,int p)

{

char \*s1 = "abcde"; //abcde在常量区，s1在栈上

char \*s2 = "abcde"; //abcde在常量区，s2在栈上

s2-s1=6可能等于0，编译器优化了相同的常量，只在内存保存一份

//而&s1>&s2

char s3[] = "abcdeee";//abcdeee在常量区，

s3在栈上，数组保存的内容为abcdeee的一份拷贝

long int \*s4[100];

char \*s5 = "abcde";

int a = 5;

int b =6;

int c;

int d; //a,b,c,d均在栈上，&a>&b>&c>&d地址反向增长

char \*q=str;

int m=p;

char \*r=(char \*)malloc(1);

char \*w=(char \*)malloc(1) ; // r<w 堆正向增长

printf("s1=%p s2=%p s3=%p s4=%p s5=%p a=%p b=%p c=%p d=%p str=%p q=%p p=%p m=%p r=%p w=%p \n",

s1,s2,s3,s4,s5,&a,&b,&c,&d,&str,q,&p,&m,r,w);

/\* 栈和堆是在程序运行时候动态分配的，局部变量均在栈上分配。

栈是反向增长的，地址递减；malloc等分配的内存空间在堆空间。堆是正向增长的，地址递增。

r,w变量在栈上(则&r>&w)，r,w所指内容在堆中(即r<w)。\*/

}

运行结果：

variables address in main function:

s1=0x400800

s2=0x400800

s3=0x7ffc78ad4c40

s4=0x7ffc78ad4920

s5=0x400800

a=0x7ffc78ad491c

b=0x7ffc78ad4918

variables address in processcall:n

s1=0x400800

s2=0x400800

s3=0x7ffc78ad48b0

s4=0x7ffc78ad4590

s5=0x400800

a=0x7ffc78ad458c

b=0x7ffc78ad4588

c=0x7ffc78ad4584

d=0x7ffc78ad4580

str=0x7ffc78ad4568

q=0x40087b

p=0x7ffc78ad4564

m=0x7ffc78ad457c

r=0x11f9010

w=0x11f9030

main=0x40057d

print=0x40063e

截图：

