给裸机移植printf功能

一.可变参数函数的原型声明

typeVAFunction(type arg1, type arg2, … );

参数可以分为两部分：个数确定的固定参数和个数可变的可选参数。函数至少需要一个固定参数，固定参数的声明和普通函数一样；可选参数由于个数不确定，声明时用"..."表示。固定参数和可选参数公同构成一个函数的参数列表。

二.具体分析

下面分析c库中的printf函数，此类方法完全适用于内核的 printk

三个关键宏：

void va\_start ( va\_list arg\_ptr, prev\_param );

type va\_arg ( va\_list arg\_ptr, type );

void va\_end ( va\_list arg\_ptr );

在这些宏中，va就是variableargument(可变参数)的意思;

arg\_ptr是指向可变参数表的指针；

prev\_param指可变参数表的前一个固定参数；

type为可变参数的类型。

va\_list也是一个宏，其定义为typedef char \* va\_list，实质上是个char型指针。

char型指针的特点：作用的结果是增1和减1(因为sizeof(char)为1)。

<1>va\_start宏

(1)定义：

#defineva\_start ( ap, v ) ( ap = (va\_list)v + \_INTSIZEOF(v) )\_INTSIZEOF宏定义为：#define\_INTSIZEOF(n) ((sizeof ( n ) + sizeof ( int ) - 1 ) ~( sizeof(int ) -1 ) )

(2)作用:

根据v取得可变参数表的首指针并赋值给ap，方法：最后一个固定参数v的地址+第一个变参对v的偏移地址，然后赋值给ap，这样ap就是可变参数表的首地址。

(3)举例：

如果有一 个va函数的声明是voidva\_test(char a, char b,char c, …)，则它的固定参数依次是a,b,c，最后一个固定参数argN为c，因此就是va\_start(arg\_ptr,c)。

<2>va\_arg宏

(1)定义:

#defineva\_arg(list, mode) ((mode \*)(list = (char\*) ((((int)list + (\_\_builtin\_alignof(mode)&lt;=4?3:7)) &amp; (\_\_builtin\_alignof(mode)&lt;=4?-4:-8))+sizeof(mode))))[-1]

(2)作用:

指取出当前 arg\_ptr所指的可变参数并将ap指针指向下一可变参数

<3>va\_end宏

(1)定义为：

#defineva\_end ( list )

(2)作用:

结束可变参数的获取。va\_end( list )实际上被定义为空，没有任何真实对应的代码，用于代码对称，与va\_start对应。

三.实践

<1>怎样得到可变参数个数?归纳起来有三种办法:

(1)函数的第一个参数,指定后续的参数个数,如func(intnum,...)

(2)根据隐含参数,判断参数个数,如printf系列的,通过字符串中%的个数判断

(3)特殊情况下(如参数都是不大于0xFFFF的int),

可以一直向低处访问堆栈,直到

返回地址。

<2>举例说明三种情况：

(1)情况1

#include&lt;stdio.h&gt;  
#include&lt;stdarg.h&gt;  
void VariableFunc(int prev\_param, ...)   
{   
va\_list  
arg\_ptr;                                         
//可变参数表的首指针  
va\_start(arg\_ptr,  
prev\_param); //取得可变参数表的首地址并赋给arg\_ptr  
for(int  
i=0;i&lt;prev\_param;i++)   
{   
    int  
    ParamValue;   
    ParamValue=va\_arg(arg\_ptr,  
int);//取出当前arg\_ptr所指的可变参数并将ap指针指向下一可变参数  
printf(&quot;这是第%d个可变参数,值:%d,类型:int\n&quot;,i+1,ParamValue);  
}   
va\_end(arg\_ptr);//执行清理工作  
}

(2)情况2

#include&lt;stdio.h&gt;   
#include&lt;stdarg.h&gt; //包含些头文件  
#include&lt;string&gt;  
using namespace std;   
//模仿printf函数，写一个printk函数  
void printk(char\* prev\_param, ...)   
{   
int  
j = 0;   
va\_list  
arg\_ptr; //可变参数表的首指针  
va\_start(arg\_ptr,prev\_param);  
//取得可变参数表的首地址  
string  
FormatStr(prev\_param); //保存格式化的字符串  
int  
InsertPos;  
//当在固定参数中找到%符号时:  
while(-1!=(InsertPos=FormatStr.find(&quot;%&quot;)))  
{  
    //根据%后面的字符分别进行处理          
    if(FormatStr[InsertPos+1]=='d')  
    //%号后是'd'就转为字符再插入  
    FormatStr  
    {  
     char  
     buf[15];   
     int  
     IntValud=va\_arg(arg\_ptr,int); //从可变参数列表中获得数据  
     itoa(IntValud,buf,10);                             
     //Int 转string并保存在buf  
     FormatStr.erase(InsertPos,2);//擦除两个字符%d  
     FormatStr.insert(InsertPos,buf);  
//插入Int值到FormatStr  
    }              
    else  
if(FormatStr[InsertPos+1]=='s') ////%号后是's'就直接将字符串插入FormatStr  
    {  
     FormatStr.erase(InsertPos,2);      
FormatStr.insert(InsertPos,va\_arg(arg\_ptr,char\*));  
    }  
}   
printf(&quot;%s\n&quot;,FormatStr.c\_str());  
//打印出处理后的FormatStr  
va\_end(arg\_ptr);//执行清理工作  
}   
void main()   
{   
printk(&quot;show  
you how %s %s work %d&quot;,&quot;printf&quot;,&quot;function&quot;,88);  
}

(3)情况3

#include   &lt;stdio.h&gt;      
#include   &lt;stdarg.h&gt;       
struct   T\_Progs{       
int   x;       
int   y;      
};   
void   func(T\_Progs    \*tProgs,...)       
{    
int   total    =    0;       
va\_list   ap;       
T\_Progs   \*p;       
va\_start(ap,  
   tProgs);       
p  =    tProgs;       
printf(&quot;x[%d]=%d\n&quot;,total,p-&gt;x);  
printf(&quot;y[%d]=%d\n&quot;,total,p-&gt;y);  
total++;  
while  
   (p    =    (va\_arg(ap,T\_Progs\*)))         
{    
    printf(&quot;x[%d]=%d\n&quot;,total,p-&gt;x);  
    printf(&quot;y[%d]=%d\n&quot;,total,p-&gt;y);  
    total ++;      
}    
va\_end(ap);  
printf(&quot;参数个数:%d\n&quot;,total);  
}    
void   main(void)      
{   
T\_Progs  
   test1,test2;       
test1.x   =    1;test2.x    =    3;       
test1.y   =    2;test2.y    =    4;       
func(&amp;test1,&amp;test2,NULL);  
}

四.最简单的移植步骤

<1>我们许多选择：

(1)移植linux的printf，版本越新越难移植，但是功能也越强大

(2)移植uboot的printf，实际uboot也是移植到内核的

(3)完全自己编写，但是功能比较弱

在保证整个裸机其他代码部分没有任何问题，且编译器也没有任何问题的情况下，上述三种方法都是可行的。

下面我们只是直接采用韦东山老师移植好的printf相关的库文件，他的办法是移植2.4内核版本的printf功能

<2>拷贝附件里相关库文件到裸机代码根目录

<3>修改makefile如附件所示，必须严格按照makefile里的相关设置

<4>make 编译并测试

测试代码如下：

void test\_printf(void)  
{     
     char \*p=&quot;this is %s test&quot;;  
     char c='H';    
     int d=-256;     
     int k=0;      
     printf(&quot;testing printf\n&quot;);  
     printf(&quot;test string :::        %s\ntest char ::: %c\ntest digit ::: %d\ntest X ::: %x\ntest unsigned ::: %u\ntest zero ::: %d\n&quot;,p,c,d,d,d,k);  
}