SDL进程及SDL环境通信接口分析*

罗一静,段红光
(重庆重邮信科股份有限公司,重庆 400065)

摘 要:简单介绍了SDL语言产生的背景和特点,对SDL的结构、通信、信号以及SDL状态等基本概念做了较详细的说明。这里着重论述了SDL进程和SDL环境的通信机理及实现方法,给出了一种SDL进程在Nucleus PLUS多任务系统上运行的接口实现方法。对于利用SDL软件进行实时通信协议设计具有很好的参考价值。

关键词:进程;多任务;实时多任务系统;信道;环境;通信接口
中图分类号:TP312 SD **文献标识码**:A **文章编号**:1004-5694(2003)03-00062-06

The Analysis of the SDL Process and the Communication Interface of SDL Environment

LUO Yi-jing, DUAN Hong-guang
(*Chongqing Chongyou Information Technology Co. LTD, Chongqing* 400065, *P. R. China*)

Abstract: This paper briefly introduces the background and characteristics of SDL language, and explains the concepts of structure, communication, data, signal and states of SDL in detail. It mainly discusses the communication mechanism between the SDL process and SDL environment. At last, this paper presents a way of realizing the interface of SDL process running on the Nucleus PLUS, which is beneficial to real time communication protocol development.

Key words: process; multi-task; RTOS; channel; environment; communication interface

1 SDL语言介绍

SDL(specification and description language)语言是CCITT推荐的规范描述语言。经过ITU-T的发展和标准化,最后定义为Z.100建议。1988年推出第一个正式版本,以后每4年进行一次增补更新。作为国际标准化的正式语言,它用来规范描述实时系统。目前SDL-2000是SDL的最新版本。作为SDL相关的技术,Z.120建议——消息序列图(MSC)和Z.105建议——抽象语法符号(ASN.1)同SDL一起形成了一个完整的描述语言。

SDL主要用于通信协议软件的设计,有很多优点。它有文本和图形两种表述方式,有利于协议过程的描述;它以多任务方式实现设计,有利于和实时多任务系统的集成;TTCN测试则有利于软件的测试和分析。正因为具有这些优点,SDL语言在通信协议设计中得到了广泛的应用。

SDL软件对协议的描述过程和RTOS系统对任务的描述过程有很多相似之处。一个完整的SDL系统是由单个的进程组成的,各进程间通过路由进行通信,并以图形方式表示出来。

SDL进程是SDL系统中一个重要的概念,SDL所有的工作都是围绕进程展开的。为了方便SDL进程的设计和管理,SDL采用了结构化设计;为了SDL进程间的信息传递,各通信进程间有信道相连;为了SDL进程处理的方便,SDL提供了完成的

* 收稿日期:2002-07-12
作者简介:罗一静(1972-),男,重庆市人,工程师,主要从事第三代移动通信系统TD-SCDMA的高层信令开发工作。

数据类型;SDL 的行为定义了SDL 进程所处理的任务。

1.1 结构

SDL 设计的系统是一个等级结构的系统,包括了系统、块、进程和过程。在系统级下定义的数据类型、信号等可以在该系统下所有的块和过程中使用,在块下定义的信号可以在下级块或是进程中使用。

在SDL 中,一个系统结构可以有很多个块级结构,每个块级又可以有很多个进程或是服务组成,它们之间通过信号相互传递信息而组成一个有机的整体,如图1 所示。

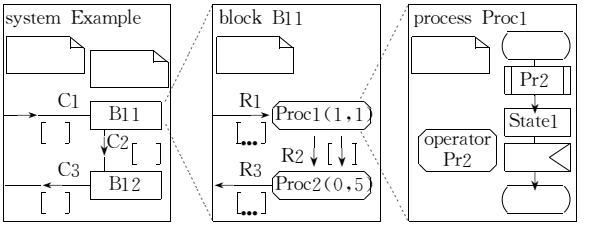

图1 SDL系统结构图

Fig. 1 SDL System Structure

1.2 通信

SDL 系统中没有全局变量,在过程中将需要交换的数据和信息都通过SDL 的信号完成,信号完成了进程间的数据传递。

在SDL 系统中,为了实现多任务系统,发送和接收都是异步进行的,即信号往往不是立即处理的。SDL 系统有自己的消息队列,SDL 信号的发送和接收由SDL 系统的内核所管理。为了方便用户的开发使用,在SDL 中可以定义信号的优先级,优先级高的优先被它所归属的进程处理,如图2 所示。SIGNAL 是SDL 关键字,用来申明SDL 信号,格式为信号名(信号数据类型)。

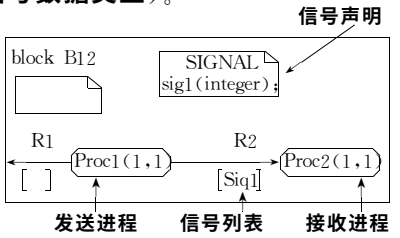

图2 SDL信号收发块图

Fig. 2 SDL Signal Sending and Receiving

1.3 行为

SDL 系统处理事件是在进程中完成的。在进程中,SDL 可以分析接收数据并发送出相应的分析结果,即完成消息的加工和控制。在SDL 中,系统级和块级的SDL 图是相对稳定的,在SDL 的处理过程中不会增加和删除,但是进程可以动态地创建和删除。如图3 中,SDL 系统在初始化期间并没有产生进程1,而是在进程2 的产生过程中,通过使用SDL 的进程产生图标生成进程1。

1.4 数据类型

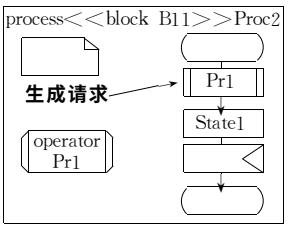

图3 SDL功能描述图

Fig. 3 SDL Function Description

SDL 提供了一套完整的数据类型定义,有比特、比特串、字符、字符串等形式,用户还可以根据自身需要,定义自己专用的数据结构。Telelogic Tau 公司的SDL 软件提供的数据及数据类型定义按照ITU-T 的Z.105 进行,并且支持在通信中最有用的ASN.1 描述。

为了方便设计人员对数据的处理,SDL 系统除了可以定义数据类型以外,还可以定义数据类型间的运算关系。此外,SDL 的数据结构的定义很容易映射到其他高级语言的数据类型上,如表1 所示,SDL 的数据类型在C 语言中基本都有相应的数据类型对应,SDL 的这些特点为SDL 移植到RTOS 系统提供了方便。

表1 SDL 和C 语言主要数据类型对应表

Tab. 1 The Data Type Comparison Table of SDL and C

C 语言数据类型	SDL 数据类型
Char	Character
Int	Integer
long int	LongInt
short int	ShortInt
unsigned long int	UnsignedLongInt
unsigned short int	UnsignedShortInt
float	Float
double	Real
void *	Voidstar
char *	CharStar
typedef x,y	syntype y = x endsyntype
typedef struct { int a; } x;	newtype x struct a Integer; endnewtype;

1.5 状态图

在实时操作系统中,状态和状态转换是最基本的也是最重要的概念,几乎所有的工作都是围绕状态而进行的,每个状态都有自己的任务。

SDL 的每个进程(process)由多个状态组成,每个状态都定义了状态的属性、行为等。在运行中,每个进程有且只有一个状态处于激活状态。

SDL 软件运行以后,每个进程都从进程的初始状态进入到相应的稳定状态。此后的时间里,如果该进程没有收到其它进程或是环境送来的信号,该进程不会自动启动自己的任务运行(除了有定时器的进程以外),该进程将继续等待需要自己处理信号的到来。只有进程收到需要自己处理的信号时,才有可能发生状态的跳变。如果进程收到不属于自己激活状态所处理的信号时,又没有保存该信号的申明,那么该进程会将此信号丢掉,并保持原状态。

图 4 状态转移图表明,在系统开始运行后,SDL 直接进入到 Locked 状态。在 Locked 状态收到 Unlock 信号而进入到 Unlocked 状态,系统要进入 DoorOpen 状态,还需要 Open 信号的驱动。在系统运行期间,这 3 个状态在一定信号的驱动下相互转换。

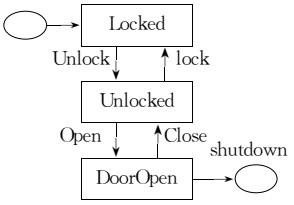

图4　状态转移图

Fig. 4 State Transfer Chart

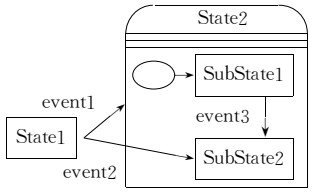

图5　状态和子状态关系图

Fig. 5 The Relation of State and Sub—State

为了提供更灵活的开发设计,在一个状态下又可以分成很多子状态,这样有利于大型或是复杂系统的开发,也有利于提高软件的运行速度。在图 5 中,State2 就有两个子状态 SubState1 和 SubState2,这三个状态 State2、SubState1、SubState2 它们有自己的行为和定义,状态 State1 可以发送信号给 State2 的子状态。

2　SDL 的进程管理

SDL 和 RTOS 有不同的集成方法,决定了 SDL 不同的进程管理。在深度集成方式中,SDL 的进程和信息由 RTOS 系统调度和管理,轻度集成则由 SDL 自己的内核管理。本节将介绍 SDL 轻度集成方式的工作机理。

在 SDL 启动以后,SDL 首先根据 SDL 设计的内容,初始化 SDL 系统。这个过程包括 SDL 内核的初始化(内存分配、生成空的信号队列、环境进程、定时器等)和 SDL 进程的初始创建工作。以后的任务就是 SDL 内核对各个 SDL 进程的管理。SDL 以信号驱动方式启动进程,在没有信号需要处理的时候,SDL 一直处于等待状态。

图 6 就是 SDL 的信号流向图,在整个 SDL 系统中,所有的 SDL 进程共用一个信号的队列,信号首先发送到 SDL 的信号队列中,等待 SDL 内核程序处理,SDL 内核程序根据信号的内容启动相应的 SDL 进程。在 SDL 的进程运行管理中,SDL 信号队列起到关键的作用。实际上 SDL 信号队列是由数据结构组成的双向列表,在每个信号中定义了信号的优先级、信号的收发进程标识、信号的名字以及信号的参数,如图 7 所示。

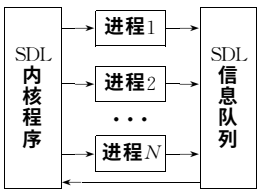

图6　SDL信号流程图

Fig. 6 The Flow of SDL Signal

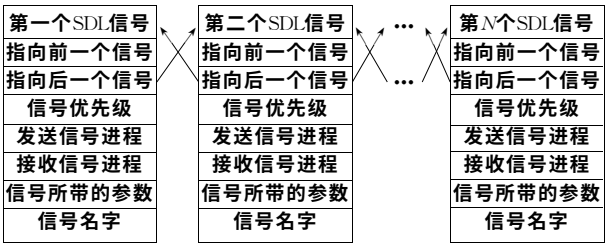

图7　SDL信号列表图

Fig. 7 SDL Signal List

每当 SDL 进程发送信号到其它进程时,SDL 就在信号队列中增加一个信号,当 SDL 内核把信号从队列中取出时,SDL 内核程序将把该信号从信号队列中删除。SDL 不时的查询信号队列,并根据信号来完成对 SDL 进程的调度。SDL 进程和 SDL 信号的优先级处理也是在 SDL 对信号队列的处理中完成的,优先级高的信号,SDL 内核将该信号优先取出处理。

3　SDL 系统和操作系统的集成

在实际应用中,由 SDL 描述生成的 C 源代码最终要在硬件系统上运行,常常需要 RTOS 系统的支持。SDL 的进程任务在 RTOS 系统中的处理情况可以分成两种:深度集成(tightintegration)和轻度集

成(lightintegration)，它们的效果是一样的。区别在于深度集成将SDL 描述中的每一个进程作为RTOS 中的一个进程来处理，而轻度集成是将整个SDL 系统作为一个RTOS 进程来进行处理，如图8 所示。在实际应用中，一般使用轻度集成，这便于 SDL 和RTOS 系统的集成，只需要修改很少的SDL 的源代码即可。无论使用何种方法，在硬件系统上运行时，都需要对环境文件进行修正，以满足不同的硬件需要。

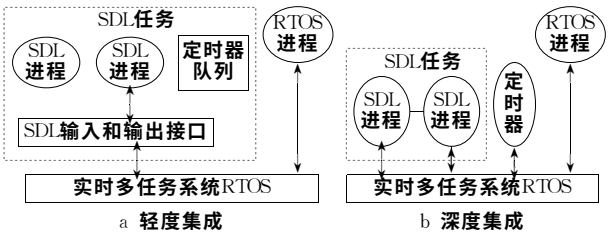

图8 SDL和RTOS系统集成方法图

Fig. 8 SDL and RTOS Integration

SDL 可以和很多种RTOS 系统相互集成，它们的原理是相同的，这里只以SDL 在Nucleus Plus 下的集成情况来举例说明。在SDL 的接口文件setenv.c 中需要增加下列Nucleus PLUS 的函数。

NU_Create_Memory_Pool()

NU_Allocate_Memory()

NU_Create_Task()

前面两个函数都是Nucleus PLUS 用来内存分配，保存SDL 进程控制块，在这里就不再介绍了，有必要说明的是NU_Create_Task()函数的使用，它是SDL 进程和Nucleus PLUS 接口的关键函数。

NU_Create_Task()函数总共有 11 个参数，Create_Task(Task_Ptr，"SDL_fn"，SDL_main，0，NULL，Stack_Ptr，MANAGER_STACK_SIZE，MANAGER_TASK_PRIORITY，MANAGER_TIME_SLICE，NU_PREEMPT，NU_START)；其中 Task_Prt 是指向进程的控制块、"SDL_fn"是进程的名字、Stack_Ptr 是进程的堆栈指针、MANAGER_STACK_SIZE 是堆栈的大小，MANAGER_TIME_SLICE 是进程可以占用的最大的时间(以ticks 来计算)。经过修改以后的接口函数，只要把SDL 生成的源代码和操作系统提供的代码一起编译就可以了。每次系统启动以后，Nucleus PLUS 将SDL 生成的源代码当作一个独立的任务进行处理，SDL 进程由SDL 内核管理，而整个SDL 系统则由Nucleus PLUS 管理。Nucleus PLUS 系统有自己一套完整的消息处理机制，同时SDL 生成的代码也有自己的一套信号处理机制，但是它们相互独立，共同作用组成应用系统。

3.1 SDL 和外部环境的接口

SDL 设计的软件在多任务系统上运行，那么SDL 进程必然和多任务系统或是环境进程有数据交换。SDL 和外部接口是SDL 应用中的一个重要内容。由于实际的操作系统和硬件环境千姿百态，所以SDL 软件不可能给出一个固定的代码来完成和外部接口的功能，需要用户根据自己的要求作特定的设计。在完成 SDL 和外部环境通信时，既要满足SDL 生成代码的格式，又要考虑发送信号在其它RTOS 进程或是硬件驱动的要求。

3.2 SDL 系统和外部环境进行通信的机制

SDL 系统要生成一套完整的代码，有自己完整的通信机制，是一个有机的整体，和 SDL 系统进行通信都必须满足这种机制。

由于SDL 系统和RTOS 系统集成，SDL 系统发出的信号，如果是发送到其它 RTOS 进程，必须将SDL 发送出来的信号以及内容转换成 RTOS 消息的格式后，才能由RTOS 系统的其它进程接收和处理。如果信号是直接发送到硬件驱动的，可以根据硬件的要求直接处理。

对于SDL 接收信号的情况有所不同，无论信号来自硬件还是其它 RTOS 系统，都需要将发送到SDL 系统的数据转化成SDL 可接收的信号格式，并且将生成的信号格式加入到SDL 的信号队列中。因为在RTOS 发送信号到SDL 系统时，SDL 系统并不知道RTOS 在发送信号给它，需要等到SDL 内核进行信号查询时才处理，所以这种方法没有像SDL 发送信号到环境那样处理方便。在SDL 和外界通信不频繁和实时性要求不高的设计中，信号查询引起的时间延迟可以不考虑。

如果SDL 系统和RTOS 频繁交互，而SDL 系统又没有及时处理从 RTOS 发送到SDL 系统的信号，有可能造成信号被覆盖而丢失，从而影响系统的稳定性和可靠性。所以 SDL 处理除提供上述方法以外，还可以将RTOS 产生的信号直接加入到SDL 系

统的信号队列中。

3.3　环境文件的具体修改方法

SDL 产生 C 语言源代码原理很简单，但是在实际应用中，要完全掌握 SDL 系统生成的源代码不是一件容易的工作，SDL 软件为了方便用户的使用，提供一个可修改环境文件的模板，它保存在…\telelogical tau40\sdt\sdtdir\wini386\include\sctenv.c 中。在该文件中有 4 个主要的函数，即 xInitEnv()、xCloseEnv()、xOutEnv()和 xInEnv()。xInitEnv() 用来增加初始化环境和硬件的处理函数；xCloseEnv() 用来关闭 SDL 软件前的处理；xOutEnv() 用来发送 SDL 信号到 RTOS 系统的处理；xinEnv() 用来接收从外界送来的信号，将信号添加到 SDL 的信号队列中，在该文件中还有其它的源代码，它们都是用来协助这几个函数的。

对 SDL 系统环境文件 setenv.c 的修改原理是一样的，但是在实际操作中有所不同，而且 SDL 的编译模式又有多种，例如 Cadvanced，Cbasic，Cmicro 等，它们生成的环境文件都是不相同的，在这里只说明 Cadvanced 一种情况。

环境文件中有 4 个主要函数 xInitEnv()，xCloseEnv()，xOutEnv()，xInEnv()，同信号发送和接收相关的只有两个 xInEnv() 和 xOutEnv()，任何一个信号的发送和接收都是由这两个函数处理的。

(1) xOutEnv() 函数的修改

SDL 发送信号到外部环境，首先设置发送的信号标志，保留发送信号在原来的队列中，等待 SDL 内核来处理。在环境进程处理中，由环境进程组装成 RTOS 信号或是直接处理，最后将已经处理完毕的信号从 SDL 信号队列中删除。为了方便和加快处理速度，SDL 生成 C 源代码的时候将为每个接口信号生成一个输出标志，这个标志就是该发送信号的名字(SDL 转换发送信号名字成 C 源代码时，转换的字符的大小写是有区别的)。下面是 SDL 软件自带的 C 源代码模板程序。

```
if ( ((*S)->NameNode) == ActualSDL_SignalName)
{
    MyVar_1 = ( (yPDP_Actual_SDL_SignalName)(*
S) )->Param1;
    MyVar_2 = ( (yPDP_Actual_SDL_SignalName)(*
S) )->Param2;
    ⋮
    MyVar_i = ( (yPDP_Actual_SDL_SignalName)(*
S) )->Parami;
    xReleaseSignal(S);
    return;
}
```

在 SDL 公司给出默认的模板程序中，如果有输出 Win 信号，那么需要将有 ActualSDL_SignalName 的地方用 Win 来替换即可，修改结果将变成

```
if ( ((*S)->NameNode) == Win) {
    MyVar_1 = ( (yPDP_Win)(*S) )->Param1; /*
MyVar_1 是 RTOS 可以任意处理的变量 */
    xReleaseSignal(S); /* 用于将发送的 SDL 信号从消息
队列中清除 */
    return;
}
```

SDL 系统提供这样的接口有利于对发送出来的 SDL 信号进行处理和重新加工，可以对 MyVar_1 变量任意处理。如果需要发送到 RTOS 的其它进程，只要利用 RTOS 系统提供信号生成函数，即可容易完成。如果信号没有参数，只要在 if 语句后面增加环境收到 Win 信号后应该处理的函数即可。

(2) xInEnv() 函数的修改

在处理从环境发送到 SDL 系统的信号时，要比处理从 SDL 发送到环境的信号困难一些，需要修改发送进程名字、信号名字以及变量名字。

在修改 xInEnv 中，最重要的就是 xGetSignal() 函数的处理，利用这个函数将产生一个 SDL 信号，它有 3 个形式参数 ActualSDL_SignalName，xNotDefPid 和 xEnv 参数，如图 9 所示。第一个参数将直接改写成 SDL 接收的信号名字；第二个参数就是 SDL 接收信号的进程名字(如果只有单个 SDL 进程和环境通信，不需要修改)；第三个参数就是指明信

图 9　接收信号函数

Fig. 9 The Function of Receiving SDL Signal

号发向环境进程。SDL 系统所定义的名字和各种变量，在 SDL 编译成 C 源代码时，如果没有特别的限

制，SDL 编译软件将改成自己独有的名字，所以在使用环境时，需要修改和环境通信的进程名字，使进程名字在生成源代码时不被改变。下面就是一个SDL 软件提供的模板代码：

```
while( AnyInputActive ){
    if ( Input_1_Active ){
        S = xGetSignal( ActualSDL_SignalName, xNotDefPId, xEnv );
        ((yPDP_ActualSDL_SignalName)S)_>Param1 = MyData_1;
        ((yPDP_ActualSDL_SignalName)S)_>Paramn = MyData_n;
        SDL_ Output ( S xSigPrioPar (xDefaultPrioSignal), (xIdNode *)0 );
        Input_1_Active = 0;
    }
}
```

对xInEnv()函数的修改和发送信号xOutEnv()的修改是基本相同的，需将ActualSDL_SignalName修改成SDL 实际使用的信号名字，与发送信号不同的是这里还有一个变量Input_1_Active，在环境发送一个信号到SDL 系统之前，首先应设置这个接收信号的变量标识，通知SDL 有外来信号发送到SDL 进程。

(3) 提高SDL 信号响应速度的方法

SDL 系统在接收来自环境的信号时不能做到实时处理，即不能实时地将产生的信号加入到SDL 信号队列中，只能暂时地保留在变量中等待SDL 来查询是否有信号发送到SDL 系统。为了提高SDL 系统对信号的响应速度和系统可靠性，需要直接将发送到SDL 系统的信号加入到SDL 系统的信息队列中。

在xInEnv()函数中，不需要判断发送信号标志，也没有必要等到SDL 系统来查询是否有发送到SDL 系统的信号，而是在程序中或是RTOS 的发送消息的函数中，直接加入下面的代码来修改SDL 进程的信号队列。

```
S = xGetSignal (ActualSDL_ SignalName, xNotDefPId,xEnv );
((yPDP_ActualSDL_SignalName)S)_>Param1=MyData_1;
((yPDP_ActualSDL_SignalName)S)_>Paramn=MyData_n;
SDL_ Output (S xSigPrioPar (xDefaultPrioSignal), (xIdNode *)0 );
```

这样，在任何时候只要有信号要发送到SDL，信号不需要作暂时地保留就可以直接发送到SDL 的信号队列中。这种方法的缺点是在SDL 的测试中，不便于对信号的跟踪。

4 结束语

重庆重邮信科股份有限公司TD-SCDMA 系统的二三层高层信令就是利用SDL 软件开发设计的。由于SDL 进程和RTOS 进程有着相似的工作机理，很容易将SDL 开发的信令软件移植到RTOS 系统上。在设计过程中，使用了SDL 的图形设计和TTCN 测试功能，避开了繁琐的板级调试，减少了在硬件平台上的跟踪调试，提高了工作效率，从而在较短的时间内完成从SDL 移植到ARM 硬件工作平台。

参 考 文 献

[1] 0001027-001 Rev 101. Nucleus company. Nucleus PLUS internals. Accelerated Technology[R]. Nucleus company，2000.

[2] 0001026-001 Rev. 102. Nucleus company. Nucleus PLUS Reference Manual[R]. Nucleus company，2000.

[3] 0001000-001 Rev. 108. Nucleus company. ARM Developer Suite Nucleus Target Specific Notes[R]. Nucleus company， 2000.

[4] Telelogic tau AB-2001. Telelogic tau™ SDL suite 4.0 Getting Started[S].

[5] Telelogic tau AB-2001. Telelogic tau™ SDL suite 4.0 Methodology Guidelines[S].

[6] 陈贤亮，黄俊伟，段红光. 应用SDL 仿真实现OSI 参考模型的探讨[J]. 重庆邮电学院学报(自然科学版)，2002，14(2)：26-28.

(郭继笃)