目录

1 实验目的

学习和掌握 Promela 语言，并能在 Spin 上对协议进行模拟和分析。

2 实验环境

操作系统 Ubuntu 20.04.2 LTS, Spin Version 6.4.9 以及 iSpin Version 1.1.4 等

3 实验内容

在本次实验中，我们学习 Promela 语言的基本语法，使用 Promela 语言描述 Go-Back-N 协议，并通过 Spin 模拟与验证该协议的内容。

4 实验原理

4.1 Promela 语言

Promela 语法在许多方面类似 C 语言语法，这里我们整理一些其常见的语法规则。

4.1.1 数据类型

内置的基础类型有：bit(u1), bool(u1), byte(u8), short(16), int(32) 等，这里前缀 u 表示无符号数。

声明一个变量可以类 C 语法 `typename name [ = anyexpr ]`, 也可以使用这种语法指定该类型占用的位数 `unsigned name : constant [ = anyexpr ]`, 这样将声明一个占用 conststant 位的无符号数。当初始化的值超出了数据类型能表示的范围后，数值将会被 Spin 截断造成信息的损失。

`mtype` 是一个特殊的内置类型，它允许用户自己指定该类型的取值内容，比较类似于 C 语言的枚举 (enum) 类型，在随后的代码实现中我们将使用该类型区分数据包的类型。

4.1.2 原子操作

atomic { sequence } 使得指令序列不可分割地执行。由于 Promela 描述的是异步的过程，因而原子操作在某些情况是必要的。

4.1.3 进程

4.1.3.1 定义一个进程 使用关键字 proctype 进行定义，proctype name ([decl_lst]) { sequence } 类似 C 函数定义的方式，进程之间异步运行。

4.1.3.2 运行一个进程 使用语法 run name ([arg_lst]) 即可运行进程。

4.1.4 管道与消息传递

4.1.4.1 管道定义 管道是 Promela 中进程通信的重要工具，定义管道的语法为:

chan name = [const] of typename [, typename]*

该语法定义了一个拥有 const 单位容量的管道 name，每单位信息都有若干 typename 类型的域。

当 const 为 0 时则定义了一个同步管道，发送方与接收方通过握手协议才能进行通信。

4.1.4.2 消息发送 对管道发送消息的语法为: name ! send_args 或 name !! send_args

对一个有缓冲区的未满管道发送消息是不会阻塞的，也可以通过 spin 添加参数-m 使得对有缓冲区的管道发送消息总是非阻塞的，当管道满时，多余的消息将会丢失。

对于有缓冲区的管道而言，第一种单叹号方式将会把消息插入到管道尾，保持 FIFO 的顺序；第二种双叹号方式则会按照数值大小插入，保证管道队列有序。

4.1.4.3 消息接收 从管道接收消息的语法为: name ? recv_args 或 name ?? recv_args 或

name ?< recv_args > 或 name ??< recv_args >

第一种和第三种只有当管道第一条消息与对应的参数匹配时才可执行。

第二种和第四种当管道中任意位置存在一条消息与对应的参数匹配时即可执行。

无尖括号的方式在取完消息后清除消息队列中对应的消息，有尖括号的方式则保留不动。

4.1.5 超时

Spin 的超时系统涉及到一个预定义的、全局的可读布尔变量 `Timeout` ，当整个系统中没有任何一个进程活跃时则该变量为 *true*，不然则为 *false*。

4.1.6 控制语句

Promela 控制语句一个十分重要且不同于其他语言的地方在于`::`。只要条件满足，双双引号引导的代码指令序列均可能被执行，Spin 允许用户指定随机数种子，据此来选取每次执行的指令分治。

4.1.6.1 条件选择语句 `if :: sequence [ :: sequence ]* fi`

4.1.6.2 循环语句 通过 break 分支结束运行。`do :: sequence [ :: sequence ]* od`

4.1.6.3 无条件跳转 label 即为标签名，类似 C 语言。`goto label`

4.2 Go-Back-N 协议

为了提高 ARQ 协议的效率，在等待 ACK 确认时需要传输多个帧来使得管道忙碌，Go-Back-N(GBN) 是一种可以达到这个目的的协议。

消息的发送方维护一个长度大小固定的滑动窗口，按序号不断发送窗口中的数据包；当其接收到某个 ACK 包后，采用累积确认的方式，认为小于该 ACK 包对应序号的数据包均被成功接收，接下来只需要发送该序号及其后面的数据即可，同时更新滑动窗口的左端，使得更后的数据能够被发送。

消息的接收方维护一个长度大小为 1 的滑动窗口，只有当收到的包序号恰为当前窗口的序号，接收方才返回一个 ACK 信息，并将窗口向前滑动一格。

在网络通道环境较差时，数据包可能会超时、出错或丢失。当发送方没有在规定时间内收到一个合法的 ACK 包后，他将从窗口的左端开始重新发送一遍窗口内所有的数据包；当接收方没有在规定的时间内收到期望得到的数据包后，他也将再次将带有期望数据包序号的 ACK 请求发送给发送方。

5 具体实现

我们结合 Promela 语言的代码进行说明实现方式。

5.1 数据结构定义

通过 $WIDTH$ 定义发送端窗口的大小。这里要注意，发送方最多能发送的包数量不是 $WIDTH$ 而是 $WIDTH-1$，这是 GBN 协议循环取模所要求的，不然则可能出现数据传输的错误。

通过 $MAXN$ 来定义信道的容量，通常在实际传输中 $MAXN$ 不会是 0，但在这里我们为了便于调试与说明选用同步管道。

定义 *mtype* 为三种类型，分别表示正确的数据包、ACK 确认包、错误包。

定义两个管道 *send2recv* 和 *recv2send*, 每条信息都有两个域，第一个 *mtype* 即为数据包类型，第二个 *byte* 为序列号。

```
//定义发送方窗口大小为 WIDTH
#define WIDTH 4

//定义信道容量为 MAXN
#define MAXN 0

mtype = {msg, ack, err}
//{数据类型, 序号}
chan send2recv = [MAXN] of {mtype, byte}
chan recv2send = [MAXN] of {mtype, byte}
```

5.2 发送方

在发送方的业务逻辑中，每次循环首先读取 in 管道，查看所有的 ACK 数据包信息，如果存在正确的 ACK 数据包则可以将窗口的左端口更新为该序列号；如果出现了传输中损坏的 ACK 数据包，那么什么也不做；重复取数据包直到 in 管道清空为止，此时发送方已经尽最大努力更新了窗口的左端点。

随后，发送方将检查窗口，此时分两种情况：

- 如果窗口中数据包已经发送完了，且窗口左端点仍然未更新，那么可以认为是传输中发生了错误，将回退 N 步，更新指针位置，重新开始循环。

- 如果窗口中数据包仍然未发送完，则尝试进行一次数据发送。由于 Spin 模拟环境下信道永远是可靠的，所以我们通过随机发送错误包或不发送包来模拟实际环境中数据包在传输中的损坏或丢失情况：
  - 发送的数据包正确传输，发送 {msg, seq}，发送指针循环右移。
  - 发送的数据包损坏，发送 {err, seq}，发送指针循环右移。实际传输中，接收方可能通过 crc 循环校验等手段得知该包已损坏，其内容已经没有价值，但是在这里，我们仍然保留了正确的序列号 *seq* 为了调试说明。
  - 发送的数据包超时或丢失，发送指针循环右移，但不发送任何数据。

```
proctype send(chan in, out){
    //ready为窗口中即将发送的序号，left为窗口左边界，seq为接受到的ack号
    byte ready = 0, left = 0, seq = 0;
    do::
        // 首先读取所有可能存在的ACK信息
        do
            //读取到一个正确的ACK包
            :: in ? ack(seq) -> left = seq;

            //读取到错误的ACK包
            :: in ? err(seq) -> skip;

            //清空后退出
            :: empty(seq) -> break;
        od


        if
            :: timeout -> ready = left;
            //窗口的数据还没发送完，能发送的信息只有WIDTH - 1个
            :: ((left + WIDTH - 1) % WIDTH != ready) -> if
                //正确的发送一个信息
                ::
                    out ! msg(ready);
                    ready = (ready + 1) % WIDTH;
                //信息在传输中发生错误
                ::
                    out ! err(ready);
                    ready = (ready + 1) % WIDTH;
                //信息在传输中丢失、超时
                ::
                    skip;
                    ready = (ready + 1) % WIDTH;
            fi
        fi
    od
}
```

5.3 接收方

在接收方的业务逻辑中，每次循环只需要不断地读数据包即可：

- 读到正确的、期望的数据包，循环右移接收窗口，尝试发送载有新的序号的 ACK 包。
- 读到错误的数据包，重新发送载有旧序号的 ACK 包。
- 未读到任何数据包且超时，重新发送载有旧序号的 ACK 包。

这里，发送 ACK 包也分多种情况来模拟真实网络信道的传输情况：

- 数据包正确传输，发送 {ack(wanted)}
- 数据包损坏，发送 {err(wanted)}
- 发送的数据包超时或丢失，不发送任何数据。

```
proctype recv(chan in, out){
    //wanted 为期望得到的序列号, seq为实际获得的序列号
    byte wanted = 0, seq;
    do
        :: timeout -> sendACK: if
            //发送正确的序列号
            :: out ! ack(wanted)
            //序列号在传输中损坏
            :: out ! err(wanted)
            //数据包丢失、超市
            :: skip
        fi
        :: in ? err(seq) -> goto sendACK
        :: in ? msg(seq) -> if
            //序列号正确
            :: (wanted == seq) ->
                wanted = (wanted + 1) % WIDTH;
                goto sendACK;
            :: else -> goto sendACK;
        fi
    od
}
```

6 结果分析

选择随机数种子为 1, 通过参数 $-n1$ 来设定。

结合图1中 iSpin 给出的图示阐述部分双方交流的过程。

Sender(以下由 S 简称) 首先发送 0 号包，Receiver(以下由 R 简称) 收到后尝试返回一个 1 号 ACK 包，结果由于传输错误发送了一个错误包，S 收到错误 ACK 包后不予理睬继续发送后续的 1 号包，R 收到后尝试返回 2 号 ACK 包，结果由于传输问题传丢了该 ACK 包，S 窗口还剩下 2 号包可以发送，R 收到后再次由于传输问题传丢了 3 号 ACK 包。

此时 S 窗口所有数据包都已经传输结束，却没有收到有效的 ACK 包更新窗口，因而 S 回退 N 步，重新从 0 开始传输所有的数据包。在步骤 61 中 S 传输 0 号包，但由于传输问题传输了错误的 0 号包，而 R 在之前的传输中已经成功获得了 0、1、2 号包，正期待 3 号包，因而 R 丢弃该错误包后发送 3 号 ACK 包再次请求，然而该 3 号 ACK 包也发生传输错误，S 收到错误 ACK 包后不予理睬，并不知道 R 正等待 3 号包，继续发送 1 号包，但 1 号数据包也发生错误，...

随后连续发生了若干次丢包和包损坏的问题，当双方按照协议继续运行一段时间后才终于传输了正确的数据包，如图2所示。

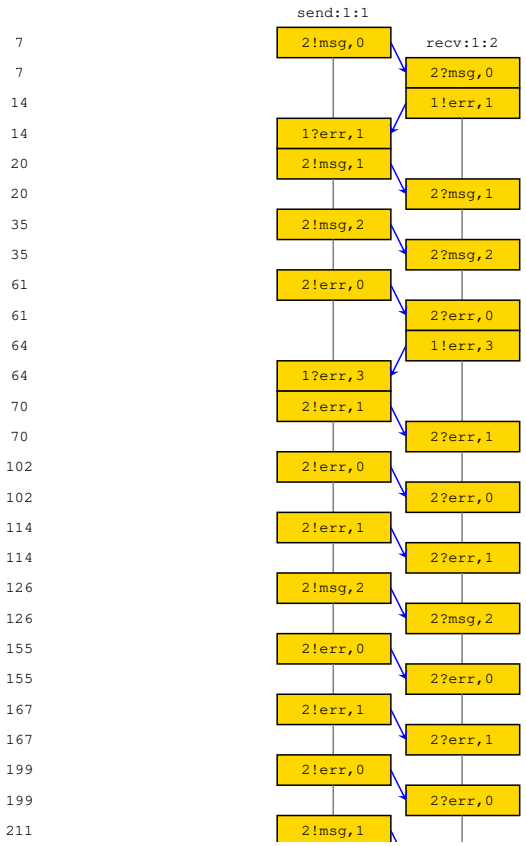

图 1: iSpin 模拟图 a

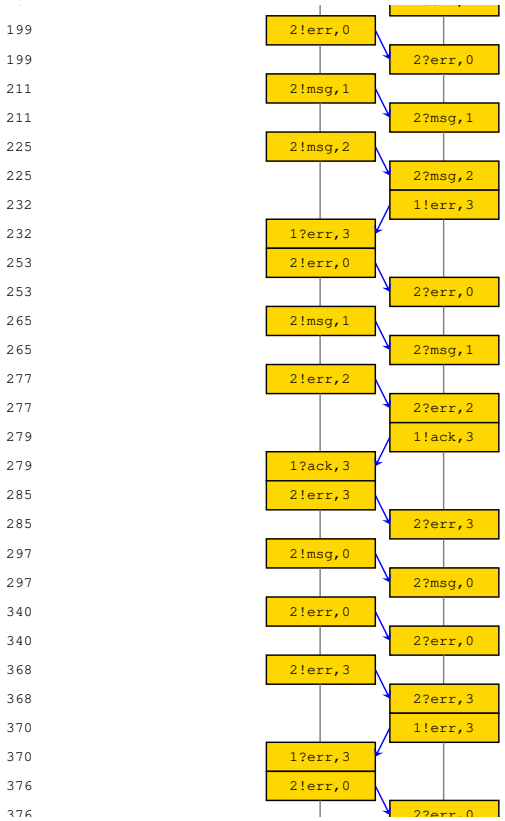

图 2: iSpin 模拟图 b

设置执行步骤上限为 3000，模拟后发现协议能够正确运行，不会发生死锁，信息能够在任何信道环境中可靠传输，如图3所示。至此认为 GBN 协议模拟成功。

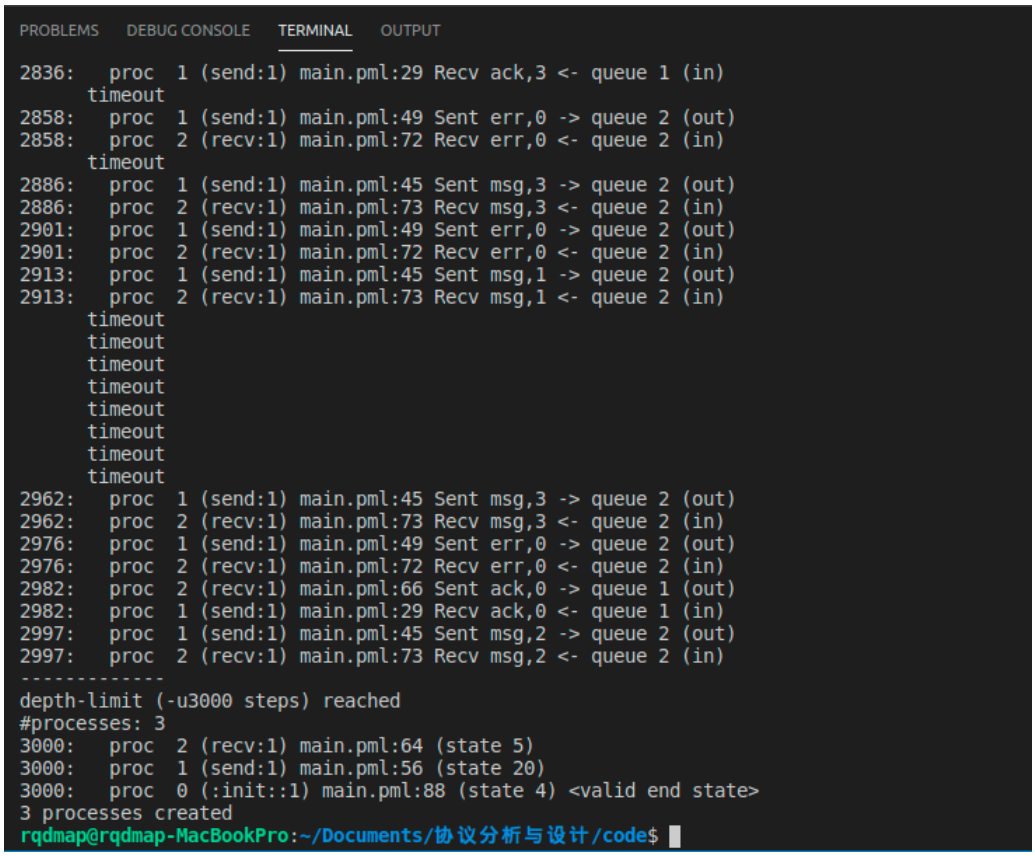

图 3: Spin 输出结果

7 代码实现

main.pml

```
//定义发送方窗口大小为 WIDTH
#define WIDTH 4

//定义信道容量为 MAXN
#define MAXN 0

mtype = {msg, ack, err}
//{数据类型, 序号}
chan send2recv = [MAXN] of {mtype, byte}
chan recv2send = [MAXN] of {mtype, byte}

/*
可能出现的问题:
    send发送数据包丢失、超时，通过send什么都不实际发送模拟
    recv返回ACK丢失、超时，通过recv什么都不返回模拟

    send发送错误数据包、通过发送错误类型表示
    recv类似
    实际数据包中，可以通过crc循环校验判断
*/

```

```
proctype send(chan in, out){
    //ready为窗口中即将发送的序号, left为窗口左边界, seq为接受到的ack号
    byte ready = 0, left = 0, seq = 0;
    do::
        // 首先读取所有可能存在的ACK信息
        do
            //读取到一个正确的ACK包
            :: in ? ack(seq) -> left = seq;

            //读取到错误的ACK包
            :: in ? err(seq) -> skip;

            //清空后退出
            :: empty(seq) -> break;
        od


        if
            :: timeout -> ready = left;
            //窗口的数据还没发送完, 能发送的信息只有WIDTH - 1个
            :: ((left + WIDTH - 1) % WIDTH != ready) -> if
                //正确的发送一个信息
                ::
                    out ! msg(ready);
                    ready = (ready + 1) % WIDTH;
                //信息在传输中发生错误
                ::
                    out ! err(ready);
                    ready = (ready + 1) % WIDTH;
                //信息在传输中丢失、超时
                ::
                    skip;
                    ready = (ready + 1) % WIDTH;
            fi
        fi
    od
}

proctype recv(chan in, out){
    //wanted 为期望得到的序列号, seq为实际获得的序列号
    byte wanted = 0, seq;
    do
        :: timeout -> sendACK: if
            //发送正确的序列号
            :: out ! ack(wanted)
            //序列号在传输中损坏
            :: out ! err(wanted)
            //数据包丢失、超市
            :: skip
        fi
        :: in ? err(seq) -> goto sendACK
        :: in ? msg(seq) -> if
            //序列号正确
            :: (wanted == seq) ->
                wanted = (wanted + 1) % WIDTH;
                goto sendACK;
            :: else -> goto sendACK;
        fi
    od
}
```

```
82
83 init{
84     atomic{
85         run send(recv2send, send2recv);
86         run recv(send2recv, recv2send);
87     }
88 }
```

8 参考资料

- Go-Back-N 协议维基百科 https://en.wikipedia.org/wiki/Go-Back-N_ARQ
- Promela 维基百科 https://en.wikipedia.org/wiki/Promela
- Promela 手册 http://spinroot.com/spin/Man/active.html