

第26章 Telnet和Rlogin：远程登录

26.1 引言

远程登录（Remote Login）是Internet上最广泛的应用之一。我们可以先登录（即注册）到一台主机然后再通过网络远程登录到任何其他一台网络主机上去，而不需要为每一台主机连接一个硬件终端（当然必须有登录帐号）。

在TCP/IP网络上，有两种应用提供远程登录功能。

1) Telnet是标准的提供远程登录功能的应用，几乎每个TCP/IP的实现都提供这个功能。它能够运行在不同操作系统的主机之间。Telnet通过客户进程和服务器进程之间的选项协商机制，从而确定通信双方可以提供的功能特性。

2) Rlogin起源于伯克利Unix，开始它只能工作在Unix系统之间，现在已经可以在其他操作系统上运行。

在本章中，我们将介绍Telnet和Rlogin。首先介绍Rlogin，因为Rlogin比较简单。

Telnet是一种最老的Internet应用，起源于1969年的ARPANET。它的名字是“电信网络协议（telecommunication network protocol）”的缩写词。

远程登录采用客户-服务器模式。图26-1显示的是一个Telnet客户和服务器的典型连接图（对于Rlogin的客户和服务器连接图，我们可以画得更加简单）。

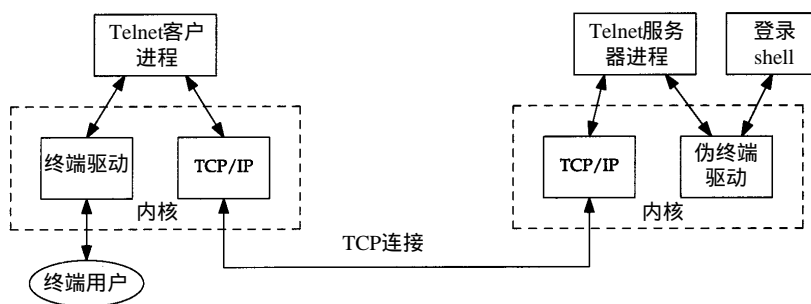


图26-1 客户-服务器模式的Telnet简图

在这张图中，有以下要点需要注意：

1) Telnet客户进程同时和终端用户和TCP/IP协议模块进行交互。通常我们所键入的任何信息的传输是通过TCP连接，连接的任何返回信息都输出到终端上。

2) Telnet服务器进程经常要和一种叫做“伪终端设备”（pseudo-terminal device）打交道，至少在Unix系统下是这样的。这就使得对于登录外壳（shell）进程来讲，它是被Telnet服务器进程直接调用的，而且任何运行在登录外壳进程处的程序都感觉是直接和一个终端进行交互。对于像满屏编辑器这样的应用来讲，就像直接在和终端打交道一样。实际上，如何对服务器进程的登录外壳进程进行处理，使得它好像在直接和终端交互，往往是编写远程登录服务器

进程程序中最困难的方面之一。

3) 仅仅使用了一条TCP连接。由于客户进程必须多次和服务端进程进行通信(反之亦然),这就必然需要某些方法,来描绘在连接上传输的命令和用户数据。我们在后面的内容中会介绍Telnet和Rlogin是如何处理这个问题的。

4) 注意在图26-1中,我们用虚线框把终端驱动进程和伪终端驱动进程框了起来。在TCP/IP实现中,虚线框的内容一般是操作系统内核的一部分。Telnet客户进程和服务端进程一般只是属于用户应用程序。

5) 把服务端进程的登录外壳进程画出来的目的是为了说明:当我们想登录到系统的时候,必须要有一个帐号,Telnet和Rlogin都是如此。

对于Telnet和Rlogin,如果比较一下它们客户进程和服务端进程源代码的数量,就可以知道这两者的复杂程度。图26-2显示了伯克利不同版本的Telnet和Rlogin客户进程和服务端进程源代码的数量。

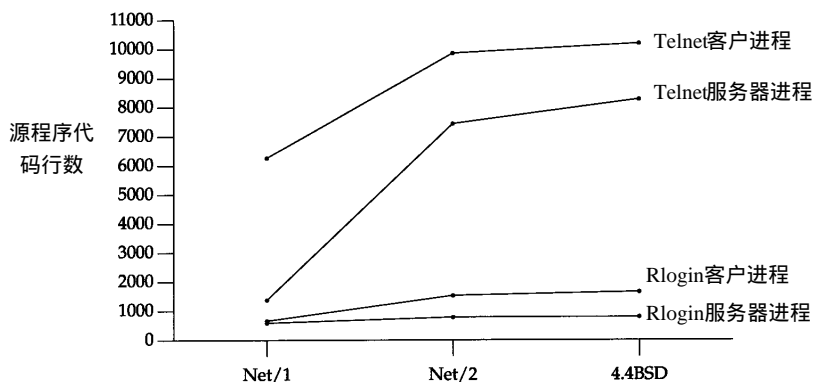


图26-2 Telnet/Rlogin/客户进程/服务端进程的源代码数量比较

现在,不断有新的Telnet选项被添加到Telnet中去,这就使得Telnet实现的源代码数量大大增加,而Rlogin依然变化不大,还是比较简单。

远程登录不是那种有大量数据报传输的应用。正如我们前面讲到的一样,客户进程和服务端进程交互的分组大多比较小。[Paxson 1993]发现客户进程发出的字节数(用户在终端上键入的信息)和服务端进程端发出的字节数的数量之比是1:20。这是因为我们在终端上键入的一条短命令往往令服务端进程端产生很多输出。

26.2 Rlogin协议

Rlogin的第一次发布是在4.2BSD中,当时它仅能实现Unix主机之间的远程登录。这就使得Rlogin比Telnet简单。由于客户进程和服务端进程的操作系统预先都知道对方的操作系统类型,所以就不需要选项协商机制。在过去的几年中,Rlogin协议也派生出几种非Unix环境的版本。

RFC 1282 [Kantor 1991]详细说明了Rlogin协议。类似于选路信息协议(RIP)的RFC,它是Rlogin用了许多年后才发布的。[Stevens 1990]的第15章介绍了远程登录的客户进程及服务端进程端的编程,并且给出了Rlogin的客户进程及服务端进程的完整源代码。[Comer和Stevens 1993]的第25章和第26章给出了Telnet的客户进程的实现细节和源代码。

26.2.1 应用进程的启动

Rlogin的客户进程和服务器进程使用一个TCP连接。当普通的TCP连接建立完毕之后，客户进程和服务器进程之间将发生下面所述的动作。

- 1) 客户进程给服务器进程发送4个字符串：(a) 一个字节的0；(b) 用户登录进客户进程主机的登录名，以一个字节的0结束；(c) 登录服务器进程端主机的登录名，以一个字节的0结束；(d) 用户终端类型名，紧跟一个正斜杠“/”，然后是终端速率，以一个字节的0结束。在这里需要两个登录名字，这是因为用户登录客户和服务器的名称有可能不一样。

由于大多满屏应用程序需要知道终端类型，所以终端类型也必须发送到服务器进程。发送终端速率的原因是因为有些应用随着速率的改变，它的操作也有所变化。例如vi编辑器，当速率比较小的时候，它的工作窗口也变小。所以它不能永远保持同样大小的窗口。

- 2) 服务器进程返回一个字节的0。
- 3) 服务器进程可以选择是否要求用户输入口令。这个步骤的数据交互没有什么特别的协议，而被当作是普通的数据进行传输。服务器进程给客户进程发送一个字符串（显示在客户进程的屏幕上），通常是password:。如果在一定的限定时间内（通常是60秒）客户进程没有输入口令，服务器进程将关闭该连接。

通常可以在服务器进程的主目录(home directory)下生成一个文件（通常叫.rhosts），该文件的某些行记录了一个主机名和用户名。如果从该文件中已经记录的主机上用已经记录的用户名进行登录，服务器进程将不提示我们输入口令。但是很多关于安全性的文献，如[Curry 1992]，强烈建议不要采用这种方法，因为这存在安全漏洞。

如果提示输入口令，那么我们输入的口令将以明文的形式发送到服务器进程。我们所键入的每个字符都是以明文的格式传输的。所以某人只要能够截取网络上的原始传输的分组，他就可以截获用户口令。针对这个问题，新版本的Rlogin客户程序，例如4.4BSD版本的客户程序，第一次采用了Kerberos安全模型。Kerberos安全模型可以避免用户口令以明文的形式在网络上传输。当然，这要求服务器进程也支持Kerberos ([Curry 1992]详细描述了Kerberos安全模型)。

- 4) 服务器进程通常要给客户进程发送请求，询问终端的窗口大小（将在后面解释）。

客户进程每次给服务器进程发送一个字节的內容，并且接收服务器进程的所有返回信息。这在19.2节中已经介绍过了。同样我们也采用了Nagle算法（在19.4节中曾经介绍），该算法可以保证在速率较低的网络上，若干输入字节以单个TCP报文段传输。操作其实很简单：用户键入的所有东西被发送到服务器，服务器发送给客户的任何信息返回到用户的屏幕上。

另外，服务器和客户之间还可以互相发送命令。在介绍这些命令之前，先介绍需要用到这些命令的场合。

26.2.2 流量控制

默认情况下，流量控制是由Rlogin的客户进程完成的。客户进程能够识别用户键入的STOP和START的ASCII字符（Control S和Control Q），并且终止或启动终端的输出。

如果不是这样，每次我们为终止终端输出而键入的Control_S字符将沿网络传输到服务器进程，这时服务器进程将停止往网络上写数据。但是在写操作终止之前，服务器进程可能已经往网络上写了一窗口的输出数据。也就是说，在输出停止之前，成千上万的数据字节还将

在屏幕上显示。图 26-3 显示了这个情况。

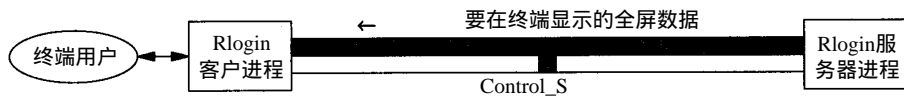


图26-3 服务器进程执行STOP/START的情况

对于一个交互式用户来讲，Control_S字符的响应延时是较大的。

有时候，服务器的应用程序需要解释输入的每个字节，但又不想让客户对它的输入内容进行处理，例如对控制字符如 Control_S 和 Control_Q 进行特殊处理（emacs 编辑器就是这样的一个例子，它把 Control_S 和 Control_C 作为自己的命令）。解决这个问题的办法就是由服务器告诉客户是否要进行流量控制。

26.2.3 客户的中断键

当我们为中断服务器正在运行的进程而键入一个中断字符时（通常是 DELETE 或 Control_C），会发生和流量控制相同的问题。这个情况和图 26-3 所示的类似，在一条 TCP 连接的管道上，从服务器进程向客户进程正在发送大量的数据，而客户进程同时在向服务器进程传输中断字符。而我们的本意是要中断字符尽快终止某个进程，使屏幕上不再有任何响应输出。

在流量控制和中断键这两种情况中，流量控制机制很少终止客户进程到服务器进程的数据流。这个方向仅仅包含我们键入的字符。所以对于从客户输出到服务器的特殊输入字符（Control_S 和中断字符）不需要采用 TCP 的紧急方式（urgent mode）。

26.2.4 窗口大小的改变

如果是窗口风格的显示方式，当应用程序在运行的时候，我们还可以动态地改变窗口的大小。一些应用程序（典型的如那些操作整个窗口的应用程序，如全屏编辑器）需要知道窗口大小的变化。目前大多数 Unix 系统提供这种功能，可以告诉应用程序关于窗口大小的变化。

对于远程登录这种情况，窗口大小的变化发生在客户端，而运行在服务器端的应用程序需要知道窗口大小变化。所以 Rlogin 的客户需要采用某些方法来通知服务器窗口大小变化的情况以及新窗口的大小。

26.2.5 服务器到客户的命令

现在我们介绍通过 TCP 连接，Rlogin 服务器进程可以发送给客户进程的 4 条命令。问题是只有一条 TCP 连接可供使用，所以服务器进程必须给这些命令字节做标记，使得客户进程可以从数据流中识别出这些是命令，而不是显示在终端上。所以我们将使用 TCP 的紧急方式（在 20.8 节中曾经介绍）。

当服务器要给客户发送命令时，服务器就进入紧急方式，并且把命令放在紧急数据的最后一个字节中。当客户进程收到这个紧急方式通知时，它从连接上读取数据并且保存起来，直到读到命令字节（即紧急数据的最后一个字节）。这时候客户进程根据读到的命令，再决定对于所读到并保存起来的数据是显示在终端上还是丢弃它。图 26-4 介绍了这 4 个命令。

采用 TCP 紧急方式发送这些命令的一个原因是第一个命令（“清空输出（flush output）”）需要立即发送给客户，即使服务器到客户的数据流被窗口流量控制所终止。这种情况下，即服

务器到客户的输出被流量控制所终止的情况是经常发生的，这是因为运行在服务器的进程的输出速率通常大于客户终端的显示速率。另一方面，客户到服务器的数据流很少被流量控制所终止，因为这个方向的数据流仅仅包含用户所键入的字符。

字节	描 述
0x02	清仓输出。客户丢弃所有从服务器收到的数据，直到命令字节（紧急数据的最后一个字节）。客户还丢弃任何有可能被缓存的挂起输出（pending output）。当服务器收到客户发出的中断命令时，就发送此命令
0x10	客户停止执行流量控制
0x20	客户继续进行流量控制处理
0x80	客户立即响应，将当前窗口大小发送给服务器，并在今后当窗口大小变化时通知服务器。通常，当连接建立后，服务器就立即发送这个命令

图26-4 服务器到客户的Rlogin命令

回忆一下图20-14中的例子，在那里我们介绍了即使窗口大小是0时，紧急通知通过网络进行传输的情况（在下节中，我们还将介绍一个类似的例子）。其他的3个命令实时性并不特别强，但为了简单起见，也采用了和第一个命令相同的技术。

26.2.6 客户到服务器的命令

对于客户到服务器的命令，只定义了一条命令，那就是：将当前窗口大小发送给服务器。当客户的窗口大小发生变化时，客户并不立即向服务器报告，除非收到了服务器发来的0x80命令（图26-4中有介绍）。

同样，由于只存在一条TCP连接，客户必须对在连接上传输的该命令字节进行标注，使得服务器可以从数据流中识别出命令，而不是把它发送到上层的应用程序中去。处理的方法就是在两个字节的0xff后面紧跟着发送两个特殊的标志字节。

对于窗口大小命令，两个标志字节是ASCII码的字符‘s’。之后是4个16 bit长的数据（按网络字节顺序），分别是：行数（例如，25），每列的字符数（例如，80），X方向的像素数量，Y方向的像素数量。通常情况下，后两个16bit是0，因为在Rlogin服务器进程调用的应用程序中，通常是以字符为单位来度量屏幕的，而不是像素点。

上面我们介绍的从客户进程到服务器进程的命令采用带内信令（in-band signaling），这是因为命令字节和其他的普通数据一起传输。选择0xff字节来表示这个带内信令的原因是：一般用户的操作不会产生0xff这个字节。所以说Rlogin是不完备的，如果我们采用某种方法，使得通过键盘就可以产生两个连续的0xff字节，而且正好在这之前是两个ASCII的‘s’字符，那么下面的8个字节就会被误认为是窗口大小了。

图26-4中介绍的是从服务器到客户的Rlogin命令，由于大多数的API采用的技术叫做“带外数据（out-of-band data）”，所以我们就称它为带外信令（out-of-band signaling）。但是回忆一下在20.8节中对TCP紧急方式的讨论，在那里我们说紧急方式数据不是带外数据，命令字节是按照普通数据流进行传输的，特殊之处是采用了紧急指针。

既然带内信令被用来传输从客户到服务器的命令，那么服务器进程必须检查从客户进程收到的每个字节，看看是否有两个连续的0xff字节。但是对于采用带外信令的、从服务器传输到到客户的命令，客户进程不需要检查收到的每个字节，除非服务器进程进入了紧急方

式。即使在紧急方式下, 客户进程也仅仅需要留意紧急指针所指向的字节。而且由于从客户进程到服务器的数据流量和相反方向的数据流量之比是 1:20, 这就暗示带内信令适合于数据量比较小的情况 (从客户到服务器), 而带外信令适合于数据量比较大的情况 (从服务器到客户)。

26.2.7 客户的转义符

通常情况下, 我们向 Rlogin 客户进程键入的信息将传输到服务器进程。但是有些时候, 我们并不需要把键入的信息传输到服务器, 而是要和 Rlogin 客户进程直接通信。方法是在一行的开头键入代字符 (tilde) “~”, 紧接着是下列4个字符之一:

- 1) 以一个句号结束客户进程。
- 2) 以文件结束符 (通常是 Control_D) 结束客户进程。
- 3) 以任务控制挂起符 (通常是 Control_Z) 挂起客户进程。
- 4) 以任务控制延迟挂起符 (通常是 Control_Y) 来挂起仅仅是客户进程的输入。这时, 不管客户运行什么程序, 键入的任何信息将由该程序进行解释, 但是从服务器发送到客户的信息还是输出到终端上。这非常适合当我们需要在服务器上运行一个长时间程序的场合, 我们既想知道该程序的输出结果, 同时还想在客户上运行其他程序。

只有当客户进程的 Unix 系统支持任务控制时, 后两个命令才有效。

26.3 Rlogin 的例子

在这里举两个例子: 第一个是当 Rlogin 会话建立的时候, 客户和服务器的协议交互; 从第二个例子可以看到, 当用户键入中断键以取消正在服务器运行的程序时, 服务器将产生很多输出。在图 19-2 中, 我们给出了通常情况下, Rlogin 会话上的数据流交互情况。

26.3.1 初始的客户-服务器协议

图 26-5 显示的是从主机 bsd1 到服务器 svr4 的 Rlogin 建立一个连接时的时间系列 (在图中, 去掉了通常的 TCP 连接的建立过程, 窗口通告以及服务类型信息)。

上节介绍的协议对应图中的报文段 1~9。客户发送一个字节的 0 (报文段 1) 之后发送 3 个字符串 (报文段 3)。在本例中, 这 3 个字符串分别是: rstevens (客户的登录名)、rstevens (服务器的登录名) 和 ibmpc3/9600 (终端类型和速率)。当服务器确认了这些信息后回送一个字节的 0 (报文段 5)。

然后服务器发送窗口请求命令 (报文段 7)。这是采用 TCP 紧急方式发送的, 我们又一次看到一个实现 (SVR4) 采用较老的但更普通的解释, 即紧急指针指明的序号是紧急数据的最后一个字节加 1。客户回送 12 字节的数据: 2 字节的 0xff, 2 字节的 's', 4 个 16 bit 长度的窗口数据。

下面的 4 个报文段 (10, 12, 14 和 16) 是由服务器发送的, 是从服务器操作系统的问候 (greeting)。之后报文段 18 是一个 7 字节长度的外壳进程提示符 “svr4%”。

客户输入的信息如图 19-2 所示, 每次发送一个字节。客户和服务器都可以主动中断该连接。如果我们输入一个命令, 让服务器的外壳程序终止运行, 那么服务器将中断该连接。如果我们给 Rlogin 客户键入一个转移符 (通常是一个 “~”), 紧跟着一个句点或者是一个文件结束符号, 那么客户将主动关闭该连接。

图26-5中，客户进程的端口号是 1023，这是由IANA分配的（在 1.9节中介绍）。Rlogin协议要求客户进程用小于1024的端口号，术语叫做保留端口。在Unix系统中，客户进程一般不能使用保留端口号，除非客户进程具有超级用户权限。这是客户进程和服务器进程相互鉴别的一部分，这种鉴别可以使得用户不需要口令而可以登录。[Stevens 1990]详细讨论了客户进程和服务器进程相互鉴别的过程和有关保留端口号的问题。

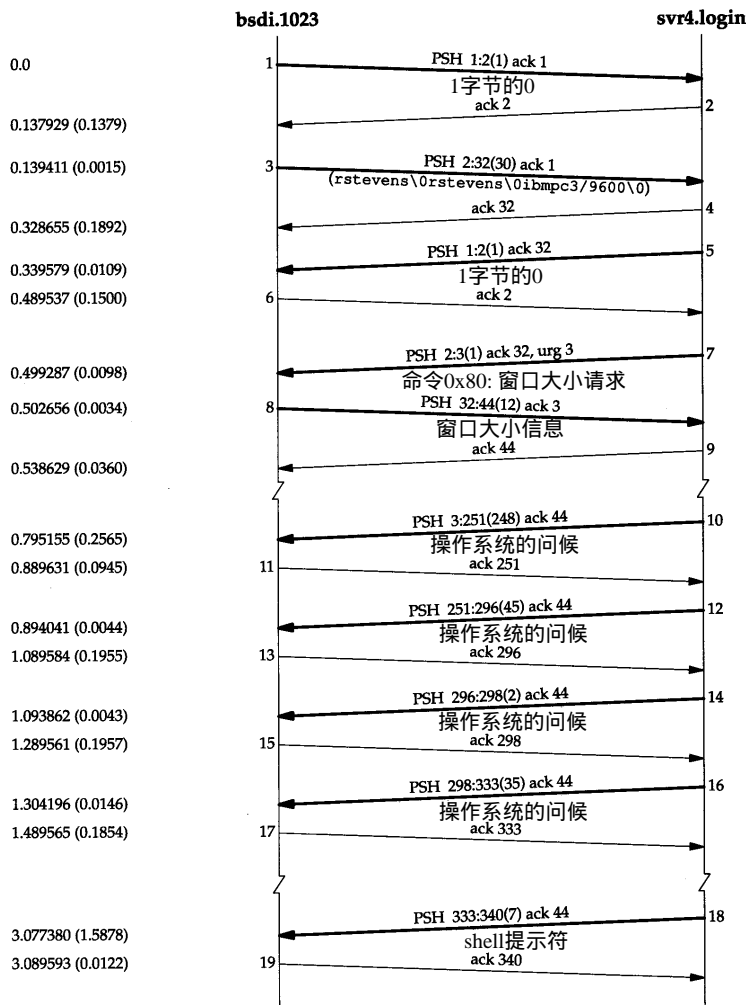


图26-5 Rlogin连接时间次序

26.3.2 客户中断键

让我们看一下另外一个例子，这个例子涉及到 TCP的紧急方式。当数据流已经终止时，我们键入中断键。这个例子要用到前面讲到的很多 TCP算法如：紧急方式、糊涂窗口避免技术、窗口流量控制和坚持计时器。在主机 sun上运行客户进程。我们登录到主机 bsdi，向终端输出一个大文本文件，然后键入 Control_S中断输出。当输出停止时，我们键入中断键 (DELETE) 以异常方式中止该进程。

```
sun % rlogin bsdi
```

所有操作系统的问候

```
bsdi % cat /usr/share/misc/termcap
```

向终端输出大文件

大量的终端输出

键入Control_S以中断输出，

然后等待直到输出停止

```
^?
```

键入中断键，而这被回显了，

```
bsdi %
```

然后输出了提示符

下面这些要点是关于客户、服务器和连接的状态的概述：

- 1) 键入Control_S以停止终端的输出。
- 2) 用户终端的输出缓存很快被填满，所以 Rlogin的客户向终端的写操作被阻塞。
- 3) 此时客户也不能从网络连接上读取数据，所以客户的 TCP接收缓存也将被填满。
- 4) 当接收缓存已满时，客户进程的 TCP会向服务器进程的TCP通告现在的接收窗口是0。
- 5) 当服务器收到客户的窗口为0时，将停止向客户发送数据，这样，服务器的发送缓存也将被填满。

6) 由于发送缓存已满，所以 Rlogin服务器进程将停止。这样，Rlogin服务器将不能从服务器运行的应用程序（cat）处读取数据。

7) 当cat程序的输出缓存也被填满时，cat也将停止。

8) 然后我们用中断键来终止服务器上的 cat程序。这个命令从客户的 TCP传输到服务器的TCP，这是因为该方向的数据传输没有被流量控制所终止。

9) cat应用程序收到中断命令并且终止。这使得它的输出缓存（也就是 Rlogin服务器进程读取数据的地方）被清空，这将唤醒Rlogin服务器进程。然后Rlogin服务器进程进入紧急方式，向客户进程发送“清仓输出”命令（0x02）。

图26-6概括了从服务器到客户的数据流（图中的序号就是下面将介绍的图中的时间系列）。

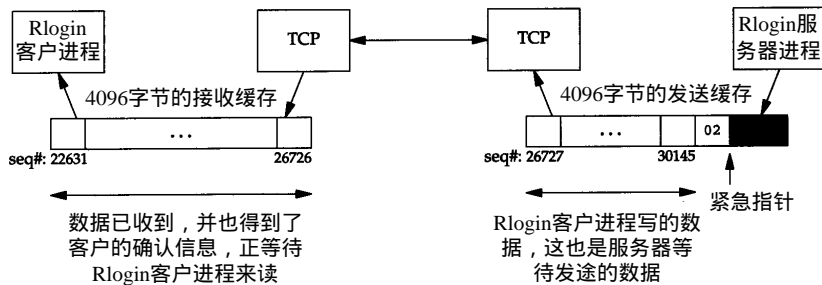


图26-6 Rlogin例子中，服务器进程到客户进程的数据流概述

发送缓存的阴影部分是4096字节的缓存中没有被使用的部分。图26-7是该例子的时间系列。

在报文段1~3，服务器进程向客户进程发送满长度（即1024字节）的TCP报文段。由于此时客户进程不能向终端写信息，客户进程也不能从网络上读数据，所以在报文段4中，客户进程向服务器进程发送ACK确认时，告诉服务器进程此时接收窗口是1024个字节。在报文段5中，服务器进程发送的数据长度就不再是满长度的了。同样，报文段6中客户进程的确认信号所带的接收窗口大小是此时接收缓存的空余字节长度。那么在报文段5中，客户进程ACK信号中为什么接收窗口大小是349而不是0呢？这是因为如果发送的是0（糊涂窗口避免技术），那么窗口指针将右边界移动到了左边界，而这是绝对不能发生的（见20.3节）。当服务器进程收到报

文段6的ACK信号后，它就不能再发送全长的数据报了，这时候它就采用糊涂窗口症避免技术，不发送任何东西，同时置一个5秒的坚持计时器。当计时器超时，服务器进程就发送一个349字节大小的数据（如报文段7）。由于此时客户进程依然不能输出接收缓存的信息，所以接收缓存将被填满，客户进程将发送ACK信号，此时接收窗口大小为0（如报文段8）。

这时候我们键入中断键并且以报文段9显示的那样传输。此时的接收窗口大小依然为0。当服务器进程接收到该中断键后，服务器进程把它发送给应用程序（cat），应用程序就终止。由于应用程序被终端中断键所终止，应用程序就清空它的输出缓存。服务器进程发现该变化后就通过TCP紧急方式向客户进程发送“清空输出”命令，这如报文段10所示。注意命令字节0x02放在第30146字节中（紧急指针减1）。报文段10告诉客户进程在命令字节前还有3419个字节（从26727到30145）在服务器进程的发送缓存中等待发送。

报文段10采用紧急通知方式发送，包含了服务器进程向客户进程发送的下一个字节（序号是26727）。它不包含“清空输出”命令字节。记得在22.2节中曾经介绍过，发送进程可以发送一个字节的数来试探对方的接收窗口是否关闭。报文段10就是采用了这个原理。然后客户进程TCP就立即发送如报文段11所示的数据。虽然此时接收窗口还是0，但是在客户进程内部，由于客户进程的TCP收到了对方的紧急通知，它把该通知告诉客户进程，客户进程就知道服务器进程已经进入了紧急方式了。

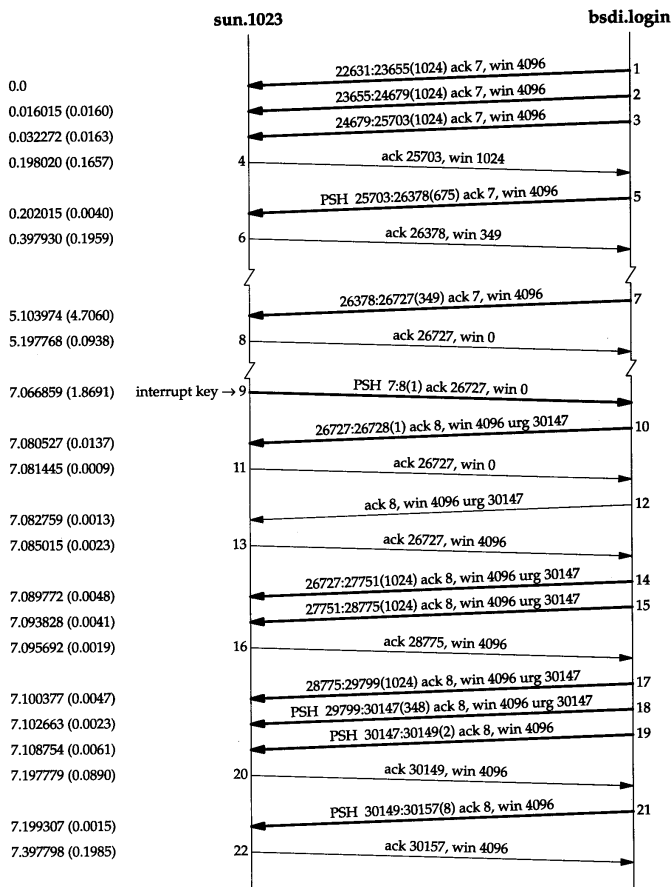


图26-7 Rlogin举例：当客户程停止输出然后终止服务器的应用程序的情况

当Rlogin客户进程从它的TCP收到了紧急通知, 并且客户进程开始读取已经在输入缓存中等待被读取的数据时, 接收窗口就会重新打开(报文段 13)。然后服务器进程就开始正常发送数据(报文段 14, 15, 17和18)。注意报文段18的数据报中包含紧急数据的最后一个字节的数据(序号30146), 该字节包含服务器进程发送给客户进程的命令字节。当客户进程收到该命令后, 它就丢弃报文段 14、15、17和18所收到的数据, 并且清空终端的输出缓存。在报文段 19中的下两个字节是中断键的回显“^?”。最后一个报文段(21)包含了客户进程的外壳提示符。

这个例子描述了当用户键入中断键后, 连接的双方数据如何被存储的情况。如果这些动作仅仅丢弃在服务器的 3419个字节数据, 而不丢弃已经在客户的 4096个字节的数据, 那么这些已经在客户的终端输出缓存中的 4096字节数据将输出到终端上。

26.4 Telnet协议

Telnet协议可以工作在任何主机(例如, 任何操作系统)或任何终端之间。RFC 854 [Postel 和Reynolds 1983a]定义了该协议的规范, 其中还定义了一种通用字符终端叫做网络虚拟终端NVT (Network Virtual Terminal)。NVT是虚拟设备, 连接的双方, 即客户机和服务器, 都必须把它们的物理终端和 NVT进行相互转换。也就是说, 不管客户进程终端是什么类型, 操作系统必须把它转换为 NVT格式。同时, 不管服务器进程的终端是什么类型, 操作系统必须能够把NVT格式转换为终端所能够支持的格式。

NVT是带有键盘和打印机的字符设备。用户击键产生的数据被发送到服务器进程, 服务器进程回送的响应则输出到打印机上。默认情况下, 用户击键产生的数据是发送到打印机上的, 但是我们可以看到这个选项是可以改变的。

26.4.1 NVT ASCII

术语NVT ASCII代表7比特的ASCII字符集, 网间网协议族都使用NVT ASCII。每个7比特的字符都以8比特格式发送, 最高位比特为0。

行结束符以两个字符 CR (回车) 和紧接着的 LF (换行) 这样的序列表示。以 \r\n来表示。单独的一个 CR也是以两个字符序列来表示, 它们是 CR和紧接着的 NUL (字节0), 以 \r\0表示。

在下面的章节中可以看到, FTP, SMTP, Finger和Whois协议都以NVT ASCII来描述客户命令和服务器的响应。

26.4.2 Telnet命令

Telnet通信的两个方向都采用带内信令方式。字节 0xff (十进制的 255) 叫做 IAC (interpret as command, 意思是“作为命令来解释”)。该字节后面的一个字节才是命令字节。如果要发送数据 255, 就必须发送两个连续的字节 255 (在前面一节中我们讲到数据流是 NVT ASCII, 它们都是 7bit的格式, 这就暗示着 255这个数据字节不能在 Telnet上传输。其实在 Telnet中有一个二进制选项, 在 RFC856[Postel和Reynolds 1983b]中有定义, 关于这点我们没有讨论, 该选项允许数据以 8bit进行传输)。图26-8列出了所有的Telnet命令。

由于这些命令中很多命令很少用到, 所以对于一些重要的命令, 如果在下面章节的例子或叙述中遇到, 我们再做解释。

名称	代码(十进制)	描 述
EOF	236	文件结束符
SUSP	237	挂起当前进程（作业控制）
ABORT	238	异常中止进程
EOR	239	记录结束符
SE	240	子选项结束
NOP	241	无操作
DM	242	数据标记
BRK	243	中断
IP	244	中断进程
AO	245	异常中止输出
AYT	246	对方是否还在运行？
EC	247	转义字符
EL	248	删除行
GA	249	继续进行
SB	250	子选项开始
WILL	251	选项协商（图 26-9）
WONT	252	选项协商
DO	253	选项协商
DONT	254	选项协商
IAC	255	数据字节 255

图26-8 当前一个字节是IAC（255）时的Telnet命令集

26.4.3 选项协商

虽然我们可以认为 Telnet 连接的双方都是 NVT，但实际上 Telnet 连接双方首先进行交互的信息是选项协商数据。选项协商是对称的，也就是说任何一方都可以主动发送选项协商请求给对方。

对于任何给定的选项，连接的任何一方都可以发送下面 4 种请求的任意一个请求。

1) WILL：发送方本身将激活 (enable) 选项。

2) DO：发送方想叫接收端激活选项。

3) WONT：发送方本身想禁止选项。

4) DON'T：发送方想让接收端去禁止选项。

	发送方	接收方	描 述
1.	WILL	DO	发送方想激活选项 接收方说同意
2.	WILL	DONT	发送方想激活选项 接收方说不同意
3.	DO	WILL	发送方想让接收方激活选项 接收方说同意
4.	DO WONT		发送方想让接收方激活选项 接收方说不同意
5.	WONT	DONT	发送方想禁止选项 接收方必须说同意
6.	DONT	WONT	发送方想让接收方禁止选项 接收方必须说同意

图26-9 Telnet选项协商的6种情况

由于 Telnet 规则规定，对于激活选项请求（如 1 和 2），有权同意或者不同意。而对于使选项失效请求（如 3 和 4），必须同意。这样，4 种请求就会组合出 6 种情况，如图 26-9 所示。

选项协商需要 3 个字节：一个 IAC 字节，接着一个字节是 WILL, DO, WONT 和 DONT 这四

者之一,最后一个ID字节指明激活或禁止选项。现在,有40多个选项是可以协商的。Assigned Number RFC文档中指明选项字节的值,并且一些相关的RFC文档描述了这些选项。图 26-10显示了在本章中会出现的选项代码。

Telnet的选项协商机制和Telnet协议的大部分内容一样,是对称的。连接的双方都可以发起选项协商请求。但我们知道,远程登录不是对称的应用。客户进程完成某些任务,而服务器进程则完成其他一些任务。下面我们将看到,某些Telnet选项仅仅适合于客户进程(例如要求激活行模式方式),某些选项则仅仅适合于服务器进程。

选项标识(十进制)	名 称	RFC
1	回显	857
3	抑制继续进行	858
5	状态	859
6	定时标记	860
24	终端类型	1091
31	窗口大小	1073
32	终端速率	1079
33	远程流量控制	1372
34	行方式	1184
36	环境变量	1408

图26-10 本章中将讨论的Telnet选项代码

26.4.4 子选项协商

有些选项不是仅仅用“激活”或“禁止”就能够表达的。指定终端类型就是一个例子,客户进程必须发送用一个ASCII字符串来表示终端类型。为了处理这种选项,我们必须定义子选项协商机制。

在RFC1091[VanBokkelen 1989]中定义了如何表示终端类型这样的子选项协商机制。首先连接的某一方(通常是客户进程)发送3个字节的字符序列来请求激活该选项。

<IAC, WILL, 24>

这里的24(十进制)是终端类型选项的ID号。如果收端(通常是服务器进程)同意,那么响应数据是:

<IAC, DO, 24>

然后服务器进程再发送如下的字符串:

<IAC, SB, 24, 1, IAC, SE>

该字符串询问客户进程的终端类型。其中SB是子选项协商的起始命令标志。下一个字节的“24”代表这是终端类型选项的子选项(通常SB后面的选项值就是子选项所要提交的内容)。下一个字节的“1”表示“发送你的终端类型”。子选项协商的结束命令标志也是IAC,就像SB是起始命令标志一样。如果终端类型是ibmpc,客户进程的响应命令将是:

<IAC, SB, 24, 0'I', 'B', 'M', 'P', 'C', IAC, SE>

第4个字节“0”代表“我的终端类型是”(在Assigned Numbers RFC文档中有正式的关于终端类型的数值定义,但是最起码在Unix系统之间,终端类型可以用任何对方可理解的数据进行表示。只要这些数据在termcap或terminfo数据库中有定义)。在Telnet子选项协商过程中,终端类型用大写表示,当服务器收到该字符串后会自动转换为小写字符。

26.4.5 半双工、一次一字符、一次一行或行方式

对于大多数Telnet的服务器进程和客户进程,共有4种操作方式。

1. 半双工

这是Telnet的默认方式，但现在却很少使用。NVT默认是一个半双工设备，在接收用户输入之前，它必须从服务器进程获得GO AHEAD (GA) 命令。用户的输入在本地回显，方向是从NVT键盘到NVT打印机，所以客户进程到服务器进程只能发送整行的数据。

虽然该方式适用于所有类型的终端设备，但是它不能充分发挥目前大量使用的支持全双工通信的终端功能。RFC 857 [Postel 和Reynolds 1983c]定义了ECHO选项，RFC 858 [Postel 和Reynolds 1983d]定义了SUPPRESS GO AHEAD (抑制继续进行) 选项。如果联合使用这两个选项，就可以支持下面将讨论的方式：带远程回显的一次一个字符的方式。

2. 一次一个字符方式

这和前面的Rlogin工作方式类似。我们所键入的每个字符都单独发送到服务器进程。服务器进程回显大多数的字符，除非服务器进程端的应用程序去掉了回显功能。

该方式的缺点也是显而易见的。当网络速度很慢，而且网络流量比较大的时候，那么回显的速度也会很慢。虽然如此，但目前大多数 Telnet实现都把这种方式作为默认方式。

我们将看到，如果要进入这种方式，只要激活服务器进程的 SUPPRESS GO AHEAD选项即可。这可以通过由客户进程发送DO SUPPRESS GO AHEAD (请求激活服务器进程的选项) 请求完成，也可以通过服务器进程给客户进程发送 WILL SUPPRESS GO AHEAD (服务器进程激活选项) 请求来完成。服务器进程通常还会跟着发送 WILL ECHO，以使回显功能有效。

3. 一次一行方式

该方式通常叫做准行方式 (kludge line mode)，该方式的实现是遵照RFC 858的。该RFC规定：如果要实现带远程回显的一次一个字符方式，ECHO选项和SUPPRESS GO AHEAD选项必须同时有效。准行方式采用这种方式来表示当两个选项的其中之一无效时，Telnet就是工作在一次一行方式。在下节中我们将介绍一个例子，可以看到如何协商进入该方式，并且当程序需要接收每个击键时如何使该方式失效。

4. 行方式

我们用这个术语代表实行方式选项，这是在RFC 1184[Borman 1990]中定义的。这个选项也是通过客户进程和服务器进程进行协商而确定的，它纠正了准行方式的所有缺陷。目前比较新的Telnet实现支持这种方式。

图26-11是不同的Telnet客户进程和服务器进程之间默认的操作方式。“char”表示一次一个字符方式，“kludge”表示准行方式，“linemode”表示如RFC 1184定义的实行方式。

客户端	服务器端					
	SunOS 4.1.3	Solaris 2.2	SVR4	AIX 3.2.2	BSD/386	4.4BSD
SunOS 4.1.3	char	char	char	char	kludge	kludge
Solaris 2.2	char	char	char	char	kludge	kludge
SVR4	char	char	char	char	kludge	kludge
AIX 3.2.2	char	char	char	char	kludge	kludge
BSD/386	char	char	char	char	linemode	linemode
4.4BSD	char	char	char	char	linemode	linemode

图26-11 不同的Telnet客户进程和服务器进程之间默认的操作方式

从图中可以看出，只有当客户进程和服务器进程都是BSD/386或4.4BSD的时候才支持实行方式。当服务器进程的操作系统是这两者之一时，如果客户进程不支持实行方式，才会协商进入准行方式。从图中还可以看出，其实任何类型的客户进程和服务器进程都支持准行方式，但是一般都不把它作为默认方式，除非服务器进程指定。

26.4.6 同步信号

Telnet以Data Mark命令(即图26-8中的DM)作为同步信号,该同步信号是以TCP紧急数据形式发送的。DM命令是随数据流传输的同步标志,它告诉收端回到正常的处理过程上来。Telnet的双方都可以发送该命令。

当一端收到对方已经进入了紧急方式的通知后,它将开始读数据流,一边读一边丢弃所读的数据,直到读到Telnet命令为止。紧急数据的最后一个字节就是DM字节。采用TCP紧急方式的原因就是:即使TCP数据流已经被TCP流量控制所终止,Telnet命令也可以在连接上传输。

在下节中我们将看到Telnet同步信号的使用情况。

26.4.7 客户的转义符

和Rlogin的客户进程一样,Telnet客户进程也可以使用户直接和客户进程进行交互,而不是被发送到服务器进程。通常客户的转义字符是Control_](control键和右中括号键,通常以“^]”表示)。这使得客户进程显示它的提示符,通常是“telnet>”。这时候有很多命令可供用户选择,以改变连接的特性或打印某些信息。大多数的Unix客户进程提供“help”命令,该命令将显示所有可用的命令。

在下节中我们将看到客户进程转义的例子,以及此时可以输入的命令。

26.5 Telnet举例

在这里我们将介绍在三种不同的操作方式下Telnet选项协商的情况。这些方式包括:单字符方式、实行方式和准行方式。同样我们还将讨论当用户在服务器端按了中断键退出了一个正在运行的进程后,系统的运行情况。

26.5.1 单字符方式

首先介绍基本的单字符方式,该方式类似于Rlogin。用户在终端输入的每个字符都将由终端发送到服务器进程,服务器进程的响应也将以字符方式回显到终端上。在这里运行的是一个新的客户进程BSD/386,它试图激活很多新的选项,服务器进程还是运行老的SVR4,我们将看到很多选项被服务器拒绝。

为了看到服务器和客户机之间选项协商的内容,我们将激活客户进程的一个选项来显示所有的选项协商。同样我们运行tcpdump来获得数据报交换的时间次序。图26-12显示了这个交互会话。

在图中,我们已经对由SENT或RCVD开头的选项协商的每一步都进行了标注。关于每一步的解释如下:

- 1) 客户发起SUPPRESS GO AHEAD选项协商。由于GO AHEAD命令通常是由服务器发送给客户的,而且客户希望服务器激活该选项,因此该选项的请求方式是DO(由于激活这一选项将会禁止GA命令的发送,上述过程很容易让人混淆)。在第10行可以看到服务器进程同意该选项。

- 2) 客户进程要按照在RFC 1091[VanBokkelen 1989]中的定义发送终端类型。这对Unix类型的客户进程来讲是很普通的。因为客户进程要激活本地的选项,所以该选项的请求方式是WILL。

bsdi % telnet	调用客户进程，不带任何命令行选项
telnet> toggle options	告诉客户进程显示所有的选项协商过程
Will show option processing.	
telnet> open svr4	现在和服务器建立连接
Trying 140.252.13.34...	
Connected to svr4.	
Escape character is '^['.	
SENT DO SUPPRESS GO AHEAD	1. (后面将讨论的行序号)
SENT WILL TERMINAL TYPE	2.
SENT WILL NAWS	3.
SENT WILL TSPEED	4.
SENT WILL LFLOW	5.
SENT WILL LINEMODE	6.
SENT WILL ENVIRON	7.
SENT DO STATUS	8.
RCVD DO TERMINAL TYPE	9.
RCVD WILL SUPPRESS GO AHEAD	10.
RCVD DONT NAWS	11.
RCVD DONT TSPEED	12.
RCVD DONT LFLOW	13.
RCVD DONT LINEMODE	14.
RCVD DONT ENVIRON	15.
RCVD WONT STATUS	16.
RCVD IAC SB TERMINAL-TYPE SEND	17.
SENT IAC SB TERMINAL-TYPE IS "IBMPC3"	18.
RCVD WILL ECHO	19.
SENT DO ECHO	20.
RCVD DO ECHO	21.
SENT WONT ECHO	22.
UNIX(r) System V Release 4.0 (svr4)	
RCVD DONT ECHO	23.
login: rstevens	我们键入用户和口令，服务进程不回显这些数据，
Password:	然后操作系统问候输出，然后是外壳提示符

图26-12 Telnet双方选项协商的初始化过程

3) NAWS的意思是“协商窗口大小”，它在RFC 1073 [Waitzman]中有定义。如果服务器进程同意该选项（实际上不同意，见11行），客户进程就要发送终端窗口的行、列大小的子选项。而且只要窗口大小发生变化，客户进程随时都将向服务器进程发送这一子选项（这和图26-4中Rlogin的0x80命令类似）。

4) TSPEED选项允许发送方（通常是客户进程）发送它的终端速率，这在RFC 1079 [Hedrick 1988b]中有定义。如果服务器进程同意（实际上不同意，见12行），客户进程将发送其发送速率和接收速率的子选项。

5) LFLOW代表“本地流量控制”，这在RFC1371 [Hedrick 和Borman 1992]中定义。客户进程给服务器进程发送该选项，表示客户进程希望用命令方式激活或禁止流量控制。如果服务器进程同意（实际上不同意，见13行），只要Control_S和Control_Q进程需要在客户进程和服务器进程进行切换，客户进程都要向服务器进程发送子选项（这类似于图26-4中Rlogin的0x10和0x20命令）。正如在关于Rlogin的讨论中我们所提到的那样，由客户进程进行流量控制的效果比由服务器进程来完成要好。

6) LINEMODE代表在26.4中所说的实行方式。所有终端字符的处理由Telnet客户进程完成（例如回格，删除行等），然后整行发送给服务器进程。在本节后面，我们将介绍一个例子。该选项同样被服务器进程拒绝，如14行所示。

7) ENVIRON选项允许客户进程把环境变量发送给服务器进程，这在RFC 1408 [Borman

1993a]中有定义。这样就可以把客户进程的用户环境变量自动传播到服务器进程。在 15行, 服务器进程拒绝该选项 (Unix中的环境变量通常是大写字母, 紧跟一个等号, 然后是一个字符串值, 当然这只是一个惯例而已)。默认情况下, BSD/386 Telnet客户进程发送两个环境变量: DISPLAY和PRINTER, 前提是这两个变量已经定义并且有效。Telnet用户可以定义其他一些要发送的环境变量。

8) STATUS选项 (RFC 859 [Postel 和Reynolds 1983e]中定义) 允许连接的一方询问对方对Telnet选项目前状态的理解。在这个例子中, 客户进程要求对方激活选项 (DO)。如果服务器进程同意 (实际上不同意, 见 16行), 客户进程就可以要求服务器进程以子选项的形式发送它的状态值。

9) 这是服务器进程的第一个响应。服务器进程同意激活终端类型选项 (几乎所有的 Unix类型的服务器进程都支持该选项)。但现在客户进程还不能立即发送它的终端类型。它必须要等到服务器进程用子选项的形式询问终端类型的时候才能够发送 (17行)。

10) 服务器进程同意抑制发送 GO AHEAD命令。

11) 服务器进程不同意客户进程发送它的窗口大小。

12) 服务器进程不同意客户进程发送它的终端速率。

13) 服务器进程不同意客户进程实施流量控制。

14) 服务器进程不同意客户进程激活行方式选项。

15) 服务器进程不同意客户进程发送环境变量。

16) 服务器进程不发送状态信息。

17) 这是服务器进程要求客户进程发送终端类型的子选项。

18) 客户进程把终端类型 " IBMPC3 " 以6字节的字符串形式发送给服务器进程。

19) 服务器进程要求客户进程发起请求, 要求服务器进程激活回显选项。这是本例中服务器进程第一次主动发起选项协商。

20) 客户进程同意由服务器进程实现回显功能。

21) 服务器进程要求客户进程实现回显功能。这个命令是多余的, 它只是将前两行进行了交换。这是目前大多数 Unix的Telnet服务器进程判断客户进程是否运行 4.2BSD或更新的BSD版本时的一个方法。如果客户进程回送 WILL ECHO, 就表明客户进程运行的是老版本的 4.2BSD, 不支持TCP的紧急方式 (在这种情况下就不能采用 TCP紧急方式)。

22) 客户进程回送 WONT ECHO, 表示它不是一台 4.2BSD主机。

23) 对于客户进程回送的 WONT ECHO, 服务器进程以 DONT ECHO作为响应。

图26-13显示的是本例中服务器进程和客户进程交互的时间系列 (去掉了连接建立部分)。

报文段1包含了图26-12中的1~8行。该报文段中包含 24个字节数据, 每个选项占3个字节。这是客户进程发起的选项协商。该报文段显示多个 Telnet选项可以打在一个TCP段中发送。

报文段3是图26-12中的第9行, 即 DO TERMINAL TYPE命令。报文段5包含下面的8个选项协商中服务器进程的响应, 即图 26-12中的10~17行。该报文段的长度是 27个字节, 因为 10~16行是常规选项, 每个占3个字节, 而17行的子选项部分占6个字节。报文段6包含12个字节, 和18行对应, 这是客户发送它的终端类型的子选项。

报文段8 (53个字节) 包含两个 Telnet命令和47字节的输出数据。前面的两个服务器进程发送Telnet命令占6字节, : WILL ECHO和DO ECHO (19和21行)。后47个字节的数据是:

```
\r\n\r\nUNIX(r) System V Release 4.0 (svr4) \r\n\r\0\r\n\r\0
```

前面4个字节数据在字符串输出之前产生两个空行。两字节的字符序列“\r\n”在Telnet中被认为是换行命令，而两字节的字符序列“\r\0”则被认为是回车命令。这表明数据和命令可以在一个数据段中传输。Telnet服务器进程和客户进程必须扫描接收到的每个字符，寻找 IAC 字节并执行它后续的命令。

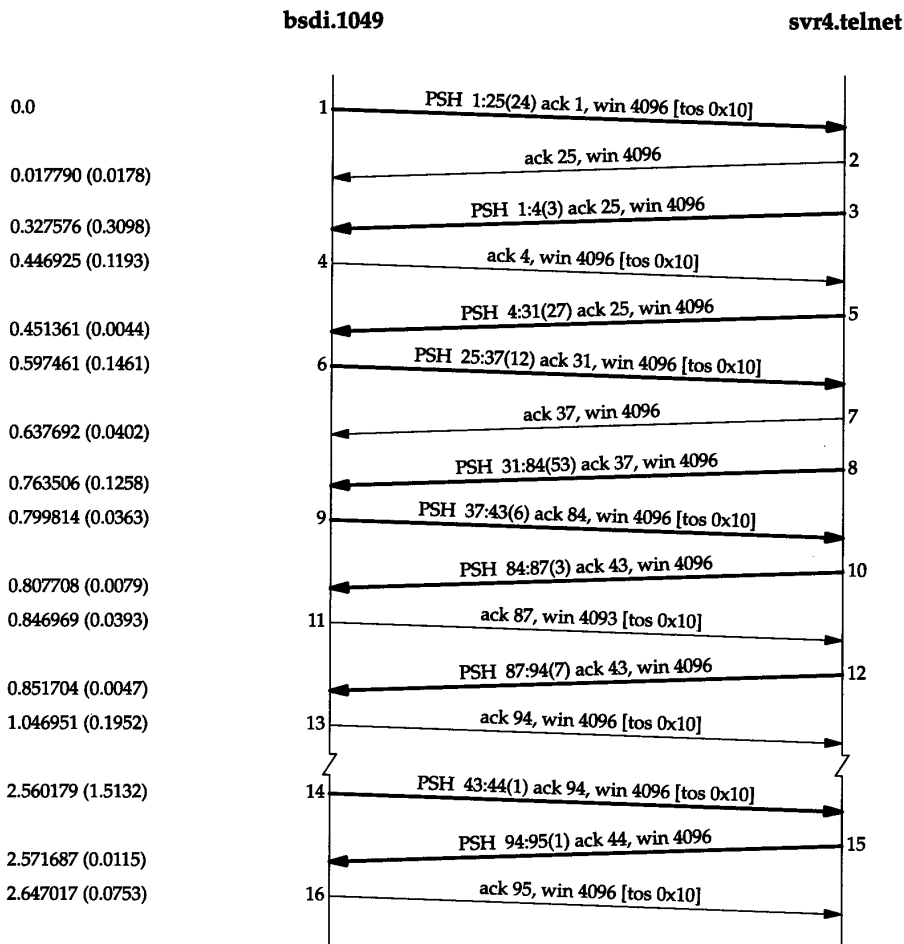


图26-13 Telnet服务器进程和客户进程选项协商初始化过程

报文段9包含客户进程发送的最后两个选项：20和22行。23行是报文段10的响应，也是服务器进程发送的最后一个选项数据。

从现在开始，双方就可以交互数据了。当然在交互数据的过程中还可以进行选项协商，我们在该例子中就不多介绍了。报文段12是服务器进程发送的提示符“login:”。报文段14是用户输入的登录用户名的第一个字符，它的回显在报文段15中。这和我们在19.2节中介绍的Rlogin交互类似：客户进程每次发送一个字符，服务器进程完成回显工作。

图26-12中的选项协商由客户进程初始化的，但是在本书中我们已经介绍了用Telnet客户进程连接某些标准服务器进程如：日间（daytime）服务器、回显(echo)服务器等情况。当然我们介绍这些目的是为了描述TCP的各种特性。但考察这些例子中

的分组交换, 如图 18-1, 我们并没有看到客户进程发起的选项协商。为什么? 这是因为在 Unix 系统中, 除非使用标准的 Telnet 端口号 23, 否则客户进程不进行选项协商。这个特性使得 Telnet 客户进程可以使用标准的 NVT 同其他一些非 Telnet 服务器进程交换数据。我们已经在日间服务器、回显服务器和丢弃 (discard) 服务器中使用了这个特性, 在后面章节介绍 FTP 和 SMTP 服务器的时候我们还将使用该特性。

26.5.2 行方式

为了描述 Telnet 的行方式选项协商过程, 我们在主机 `bsdi` 运行客户进程, 服务器是位于 `vangogh.vs.berkeley.edu` 节点运行 4.4BSD 操作系统的一台主机。BSD/386 和 4.4BSD 都支持这个选项。

我们不详细讨论所有的报文、选项和子选项协商过程, 因为这个过程和前面的例子类似, 而且对于行方式选项我们已经论述得比较清楚。下面我们仅仅讨论在选项协商中的一些区别。

- 1) 对于 BSD/386 希望协商的选项例如: 窗口大小、本地流量控制、状态、环境变量和终端速率等, 4.4BSD 服务器进程都支持。
- 2) 4.4BSD 服务器进程将协商一个 BSD/386 客户进程不支持的新选项: 鉴别 (为避免以明文形式在网络上传输用户口令)。
- 3) 和上个例子一样, 客户进程发送 `WILL LINEMODE` 选项, 由于服务器进程支持该选项, 所以服务器进程发送 `DO LINEMODE`。此时客户进程以子选项形式给服务器进程发送 16 个特定字符。这些字符是能影响客户进程的特定终端字符值: 如中断字符, 文件结束符等。服务器进程给客户进程发送一个子选项, 让客户进程处理所有的输入, 执行所有的编辑功能 (删除字符, 删除行等)。客户进程把除控制字符以外的字符以行的形式发送给服务器进程。服务器进程还要求客户进程把所有中断键和信号键转换为相应的 Telnet 字符。例如中断键是 `Control_C`, 我们可以按 `Control_C` 来中断服务器端的某个进程。客户进程必须把 `Control_C` 转换为 Telnet 的 IP 命令 (`<IAC, IP>`) 传输给服务器进程。
- 4) 当用户输入口令时情况也有所不同。在 `Rlogin` 和一次一字符方式的 Telnet 中, 都是由服务器进程负责回显, 所以当服务器进程读到口令时, 它并不回显这些字符。但在行方式中由客户进程负责回显。下面这些交互过程将处理这种情况:
 - (a) 服务器进程发送 `WILL ECHO`, 以告诉客户进程: 服务器进程将处理回显。
 - (b) 客户进程回送 `DO ECHO`。
 - (c) 服务器进程向客户进程发送字符串 `Password:`, 客户进程把它发送到终端上。
 - (d) 然后用户输入口令, 当用户按下 `RETURN` 键的时候, 客户进程把口令发送给服务器进程。此时口令不回显, 因为客户进程认为服务器进程将回显它。
 - (e) 由于口令的结束符 `RETURN` 没有回显, 服务器进程发送两字节字符序列 `CR` 和 `LF` 以移动光标。
 - (f) 服务器进程发送 `WONT ECHO`。
 - (g) 客户进程回送 `DONT ECHO`。然后继续由客户进程负责回显。

一旦登录完成, 客户进程将把数据以整行的方式发送给服务器进程。这就是行方式选项的目的。行方式大大地减少了客户进程和服务器进程之间的数据交互数量, 而且对于用户的

击键（也就是回显和编辑）提供更快的响应。图 26-14显示的是当我们输入命令时，在行方式连接下分组交换的情况。

Vangogh %date

（去掉了业务种类信息和窗口通告信息）。

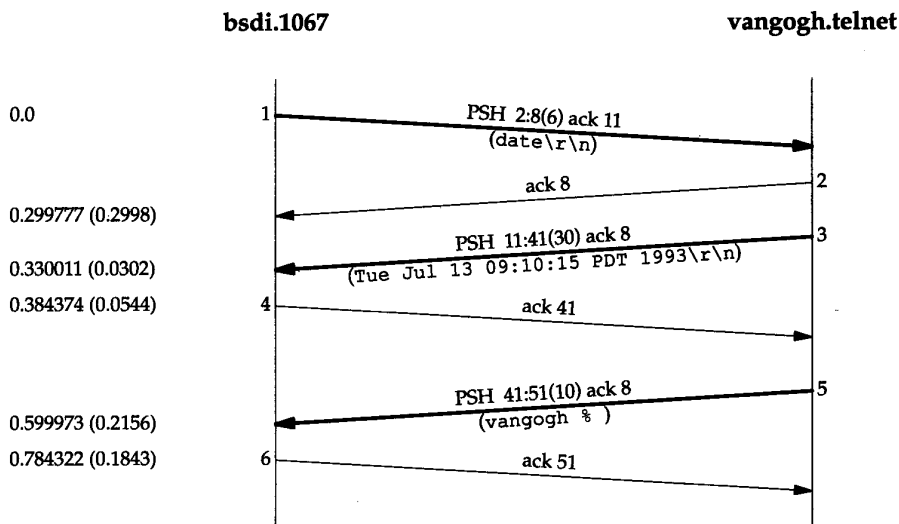


图26-14 Telnet行方式下客户进程向服务器进程发送命令的情况

把它和在Rlogin中输入同样命令（图 19-2）时的情况进行一下比较。我们看到在 Telnet行方式下只需要2个报文段（一个包含数据，另一个用于 ACK，连同IP和TCP首部共86字节），而在Rlogin中要发送15个报文段（5个有键入的数据，5个有回显的数据，5个是ACK，共611字节）。可见节省的数据量是非常可观的。

如果在服务器端运行一个需要进入单个字符方式的应用程序（例如 vi编辑器）会怎么样呢？实际上将发生如下的一些交互：

1) 当服务器的应用程序启动了，并改变其伪终端方式时，Telnet服务器进程被通告需要进入单个字符方式。然后服务器发送 WILL ECHO命令和行方式子选项，以告知客户不要再以行方式工作，转而进入单个字符方式。

2) 客户响应以 DO ECHO，并确认行方式子选项。

3) 应用程序在服务器上运行。我们键入的每个字符将发送到服务器（当然要强制使用 Nagle算法），此时服务器将处理必要的回显工作。

4) 当应用程序终止时，就恢复其伪终端方式，并通告 Telnet服务器。服务器将向客户发送 WONT ECHO命令，同时发送行方式子选项，告诉客户恢复进入行方式。

5) 客户响应 DONT ECHO，确认进入行方式。

上述情况同我们键入口令之间的区别表明：回显功能和单个字符方式与一次一行方式没有依赖关系。当我们键入口令时，回显功能必须失效，但一次一行方式有效。对于一个全屏应用来讲，例如编辑器，回显必须失效而单个字符方式必须有效。

图26-15概括了Rlogin和Telnet不同方式之间的差异。

应用程序	客户进程发送		客户进程回显?	例 子
	一次一字符	一次一行		
Rlogin	•		否	
Telnet	•		否	
Telnet, 行方式		•	是	正常命令
Telnet, 行方式		•	否	键入我们的口令
Telnet, 行方式	•		否	vi编辑器

图26-15 Rlogin和不同方式的Telnet之间的比较

26.5.3 一次一行方式（准行方式）

从图26-11可以看出，如果客户不支持行方式，那么较新的服务器支持行方式选项，它也将转入准行方式(Kludge line mode)。我们同时指出所有的客户进程和服务器进程都支持准行方式，但它不是默认方式，必须由客户进程或用户特地激活它。让我们来看看如何用 Telnet选项激活准行方式。

首先介绍当客户进程不支持行方式时，BSD/386服务器进程如何协商进入该方式。

1) 当客户进程不同意服务器进程激活行方式的请求时，服务器进程发送 DO TIMING MARK选项。RFC 860 [Postel 和 Reynolds 1983f]定义了这个Telnet选项。它的作用是让收发双方同步，关于这个问题将在本节的后面讲到用户键入中断键时讨论。该选项只是用来判断客户进程是否支持准行方式。

2) 客户响应WILL TIMING MARK，表明支持准行方式。

3) 服务器发送WONT SUPPRESS GO AHEAD和WONT ECHO选项，告诉客户它希望禁止这两个选项。我们在前面已经强调：单个字符方式下是假定 SUPPRESS GO AHEAD和ECHO选项同时有效的，所以禁止两个选项就进入了准行方式。

4) 客户响应DONT SUPPRESS GO AHEAD和DONT ECHO命令。

5) 服务器发送login:提示符，然后用户键入用户名。用户名是以整行的方式发送给服务器，回显由客户进程在本地处理。

6) 服务器发送Password:提示符和WILL ECHO命令。这将使客户进程的回显失效，因为此时客户进程认为服务器进程将处理回显工作，所以用户键入的口令就不回显到屏幕上。客户响应DO ECHO命令。

7) 我们键入口令。客户以整行方式发送到服务器。

8) 服务器发送WONT ECHO命令，使得客户重新激活回显功能，客户响应 DONT ECHO。

从此以后的普通命令处理过程就和行方式相似了。客户进程负责所有的编辑和回显，并以整行的方式发送给服务器进程。

在图26-11中，我们已经强调：所有标注为“char”的记录都支持准行方式，只不过默认是单个字符方式罢了。如果要客户进入行方式，我们就能很容易看到选项协商的过程：

```
svr4 %
telnet> status
Connected to svr4.tuc.noao.edu
Operating in character-at-a-time mode.
Escape character is '^]'.
```

客户是sun，服务器是svr 4
键入Control]以和Telnet客户进程通信(无回显)
检验现在是否在一次一字符方式下

```
telnet> toggle options
Will show option processing.
```

注意选项协商过程

```
telnet> mode line
SENT dont SUPPRESS GO AHEAD
SENT dont ECHO
RCVD wont SUPPRESS GO AHEAD
RCVD wont ECHO
```

切换到准行方式
客户发送这两个选项

服务器把WONT做为上述两个选项协商的响应

这将使Telnet会话进入准行方式，此时SUPPRESS GO AHEAD和ECHO选项都是失效的。

如果在服务器端运行如vi编辑器这样的应用程序，同样会有行方式下遇到的问题。当要运行这样的应用程序时，服务器进程必须告诉客户进程从准行方式切换到单字符方式。当应用程序结束时，必须告诉客户进程返回到准行方式。下面是这个过程需要用到技术要点。

1) 当应用程序改变其伪终端方式并通知服务器进程时，服务器进程将进入单字符方式。服务器进程向客户进程发送WILL SUPPRESS GO AHEAD和WILL ECHO，这将使客户进程进入单字符方式。

2) 客户进程回送DO SUPPRESS GO AHEAD和WILL ECHO。

3) 应用程序开始在服务器端运行。

4) 当应用程序结束并改变其伪终端方式时，服务器进程发送 WONT SUPPRESS GO AHEAD和WONT ECHO命令，使得客户进程返回准行方式。

5) 客户进程回送DONT SUPPRESS GO AHEAD和DONT ECHO命令，告诉服务器进程它已经回到了准行方式。

图26-16概括了单个字符方式及准行方式中不同的SUPPRESS GO AHEAD和ECHO选项设置。

方 式	SUPPRESS GO AHEAD	ECHO	举 例
一次一字符			准行方式下的 vi编辑器
准行方式	×	×	正常命令
准行方式	×		键入我们的口令

图26-16 准行方式下Telnet选项的设置

26.5.4 行方式：客户中断键

看一下当用户键入中断键时 Telnet将发生什么情况。假定在客户主机 bsd i和服务器 cangogh.cs.berkeley.edu之间建立了一个Telnet会话。图26-17显示了当用户键入中断键后的时间系列（去掉了窗口通告和服务类型）。

报文段1中显示的是中断键（通常是Control_C或DELETE）已经转换为Telnet的IP（中断进程）命令：<IAC, IP>。下面的3个字节：<IAC, DO, TM>，组成了Telnet的DO TIMING MARK选项。这个标志由客户进程发送，必须使用WILL或WONT响应。所有在响应前收到的数据都要丢弃（除非是Telnet命令）。这是服务器进程和客户机端的同步过程。报文段1没有采用TCP紧急方式。

Host Requirements RFC叙述了IP命令不能使用Telnet的同步信号来发送。如果可以的话，那么<IAC, IP>的后面将跟随<IAC, DM>，同时紧急指针指向DM字节。大多数的Unix Telnet 客户有一个选项来使用同步信号发送IP命令，但是这个选项默认是不用的（正如我们这里看到的）。

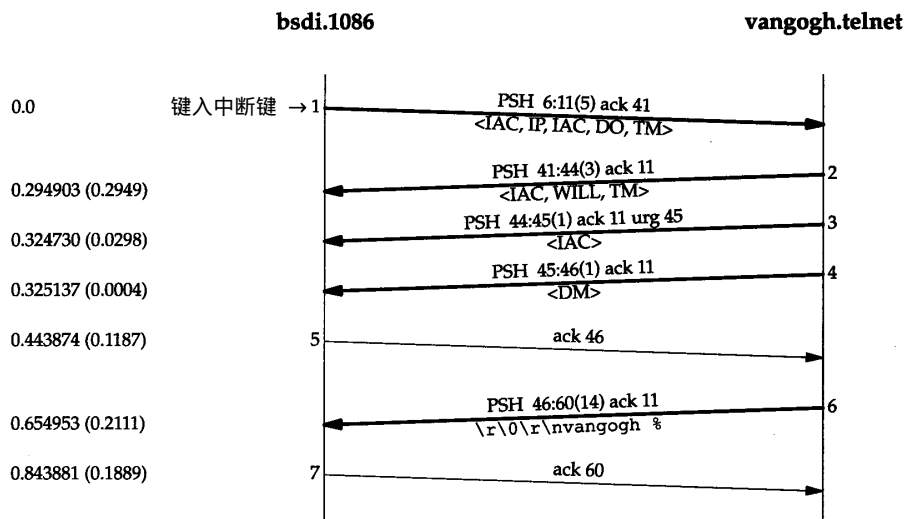


图26-17 行方式下键入中断键后的情况

报文段2是服务器进程对 DO TIMING MARK 选项的响应。紧随其后的是报文段 3和4中 Telnet 的同步信号：<IAC, DM>。报文段3中的紧急指针指向将在报文段4中发送的DM字节。

如果服务器进程到客户进程的窗口已满，那么客户进程发送了如报文段 1中的IP命令后就丢弃收到的所有数据。即使服务器进程被 TCP流量控制所终止而不能发送如报文段 2、3和4中的数据，紧急指针仍然可以发送。这和图 26-7中的Rlogin类似。

为什么同步信号要分为两个数据段发送（3和4）？原因就是我们在 20.8节中详细讨论TCP紧急指针时提到的情况。有关主机需求的 RFC中提到紧急指针应指向紧急数据的最后一个字节，而很多衍生于伯克利的系统中，紧急指针指向紧急数据的倒数第 2个字节（回忆一下在图 26-6中，紧急指针指向命令字节的前一个字节）。Telnet服务器进程故意把同步信号的第 1个字节作为紧急数据，它知道紧急指针将指向下 1个字节（即DM字节），而IAC字节和紧急指针必须立即发送，在下一步才发送DM字节。

最后一个报文段6发送的是数据，它是服务器进程发生的提示符。

26.6 小结

本章我们介绍了Rlogin和Telnet操作。两者都提供了从客户进程远程登录到服务器进程，是我们能够在服务器端运行程序的方法。

这两个应用是不同的。Rlogin假定连接的双方都是 Unix系统，所以只提供一个选项，它是1个简单的协议。Telnet则不同，它用于在不同类型的主机之间建立连接。

为了支持这种多机环境，Telnet提供客户进程和服务器进程的选项协商机制。如果连接的双方都支持这些选项，则可以增强一些功能。对于比较简单的客户进程和服务器进程，它可以提供Telnet的基本功能，而当双方都支持某些选项时，它又可以充分利用双方的新特性。

我们介绍了Telnet的选项协商机制，也介绍了 3种数据传输的方式：单字符方式、准行方式和实行方式。现在的趋势是只要有可能，就尽量工作在准行方式下。这样可以减少网络上的数据量，同时为交互用户提供更好的行编辑和回显的响应。

图26-18概括并比较了Rlogin和Telnet的不同特性。

特 征	Rlogin	Telnet
运输协议 分组方式	一个TCP连接。使用紧急方式 总是一次一字符，远程回显	一个TCP连接。使用紧急方式。 通常的默认是一次一字符，远程回显。带客户回显的准行方式也支持带回显的实行模式。当服务器上的应用进程请求时，总是一次一字符的方式
流量控制	通常由客户完成，可以被服务器禁止	通常由服务器完成，选项允许客户来完成
终端类型	总是提供	选项，通常被支持
终端速率	总是提供	选项
窗口大小	大多数服务器支持此选项	选项
环境变量	不支持	选项
自动登录	默认。提示用户键入口令，口令以明文发送。较新的版本支持 Kerberos 方式的登录	默认是键入登录名和口令。口令以明文发送。较新的版本支持鉴别选项

图26-18 Rlogin和Telnet的不同特性

Rlogin服务器和Telnet服务器通常都将设置TCP的保活选项以检测客户主机是否崩溃（如果服务器的TCP实现支持，见第23章）。这两种应用都采用了TCP紧急方式，以便即使从服务器到客户的数据传输被流量控制所终止，服务器仍然可以向客户发送命令。

习题

- 26.1 在图26-5中，标出所有延迟的ACK。
- 26.2 在图26-7中，为什么要发送报文段12？
- 26.3 我们说过Rlogin客户进程必须使用保留端口号（见1.9节）（通常Rlogin客户使用512~1023之间的保留端口）。这会给主机带来什么限制？有没有解决的办法？
- 26.4 阅读RFC 1097，它描述了Telnet的阈下报文(subliminal-message)选项。