AUTOMATION OF INTERACTIONS WITH TERMINAL-BASED APPLICATIONS

E. Agafonov, A. Bargiela
Intelligent Simulation and Modelling Group, School of Computing and Mathematics, The Nottingham Trent University, Burton Street, Nottingham, NG1 4BU

Key words to describe this work: Unix, inter-process communication, automation, pseudo-terminals, data processing, interactive interfaces

Key Results: Developed a methodology that allows automation of the control over terminal-based interactive software interfaces

How the work advances the state-of-the-art: The application of the methodology in the areas of research that involve routine data collection from sources with access to the data via an interactive interface dramatically reduces the costs of data collection process

Motivation (Problems addressed): Reduction or elimination of the need for a human operator for data collection routines

Introduction

In Transportation Research data collection plays an exceptionally important role. Nowadays, with the advances in traffic control engineering, most of the traffic data is collected from Traffic Control Systems. SCOOT [1], one of such systems, is now widely installed across the UK and other countries. Unfortunately, the access to SCOOT given to researchers is limited to an interactive terminal based interface accessed via IP network using *telnet* or *rlogin* protocols. A solution to this problem has long been the use of standard terminal software with an ability to log the I/O into a file and then parse the file in order to extract the relevant data. This approach clearly had several problems. The main problem was the need for a human operator to control the process – process the files and restart the terminal with specification of a new file. Failure to do so resulted in poor data quality and induced costs on additional data pre-processing. The situation is aggravated even further since SCOOT has internal clean-up routines that reset the commands given by the operator after which the output of the requested data is ceased. This required the operator to enter the same commands to the SCOOT on a periodic basis (daily). All the above problems make the data retrieval a laborious process, which eradicates the advantages of the routinely collected traffic measurement data.

Technical details

From the point of data collection operator, SCOOT system is a remote host, which can be accessed via an IP network using telnet or rlogin protocol. Telnet and rlogin protocols have been developed to simulate a terminal device for remote access over network. Thus, a SCOOT interface is a terminal-based application, which is a standard of textual interfaces in the family of Unix systems. In the first instance, SCOOT requests for authentication of the user. After the authentication procedure is passed, SCOOT starts interactive interface software from which the operator enters commands and requests for data output. As a necessary command is given, SCOOT starts printing traffic data in the form of textual messages to the requesting terminal. At this point, the output must be “grabbed” at the operator’s side and stored for further use. The idea behind this work was to simulate the activity of an operator by a software agent, which will be able to perform the routines without the need for a human interaction with the process.

Proposed Solution

The below Figure 1 shows the mechanism used in Unix operating system to transfer control terminal I/O streams between console application.

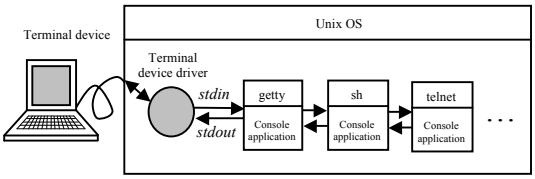

Figure 1. Forwarding stdin/stdout between console applications

The obvious solution to the problem would be to create a software agent, which will create its own I/O streams and provide the streams to telnet or rlogin as if they were from legitimate terminal device (Figure 2).

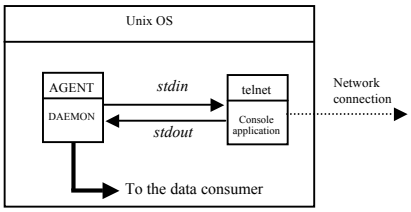

Figure 2. Agent with controlling stdin/stdout

In the scheme above the agent is a *daemon* process, which creates two streams using *pipe* and spawns a telnet (rlogin) process with its standard I/O streams initialised with the agent's streams. In such scheme, the agent receives telnet's output via telnet's *stdout* and sends control commands to telnet via its *stdin* streams.

The above scheme was implemented and tested on several stream I/O based applications, such as *sh*, *cat*, *ls*, *ping*, etc. Unfortunately, if an application uses terminal capabilities, it requires the support of underlying hardware terminal functions (*ioctl* system call). Since regular streams do not provide such terminal functionality, a terminal-based application may consider it a fatal error and cease to work. Most of applications that use terminal capabilities, such as colours or addressable cursor, are such applications (so is the SCOOT interface).

A solution for this problem has long been implemented in all Unix systems and is called "pseudo-terminals". Pseudo-terminals are bi-directional communication channels that can be used to connect two processes or a process group to another process [2]. What pseudo-terminals provide is a lot like two pipes, but with a support of hardware terminal interface. Although the implementation of hardware terminal functions is "dummy" it is sufficient for the terminal based application to function properly. Figure 3 demonstrates the application of pseudo-terminals to the data acquisition procedure.

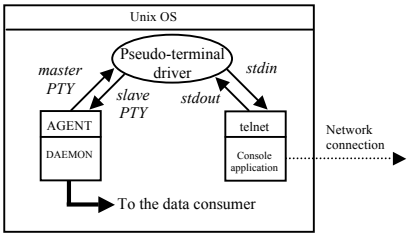

Figure 3. Implementation with pseudo terminals

In the scheme presented in Figure 3, the interface application (telnet) has an output stream with the support of terminal functions (slave PTY). Everything what is written to the slave PTY by telnet can be read by the agent. Similar piping capability has the master PTY (data flow from the agent to telnet).

Conclusion

The above scheme has been implemented and successfully tested with the SCOOT interface. The implemented agent is now used to automatically retrieve data from the SCOOT system and forward it to DIME (Distributed Shared Memory) [3] system for further processing. The methodology allows implementation of similar interface control agents for any terminal-based interfaces.

References

1. Hunt, P.B., Robertson, D.I., Bretherton, R.D., Winton, R.I. SCOOT – A traffic responsive method of co-ordinating signals. TRRL Report LR1014. Transport and Road Research Laboratory, 1981.
2. Stevens, R.W. Advanced Programming in the UNIX® Environment. Addison-Wesley Professional Computing Series, 1992.
3. Argile, A., Peytchev, E., Bargiela, A., Kosonen, I. DIME: A shared memory environment for distributed simulation, monitoring and control of urban areas. Proc ESS'96, Genoa, Vol. 1, pp.152-156, 1996