Crossroads
The ACM Student Magazine

Linux DSP Shell

by ***Michael Stricklen***, ***Bob Cummings***, and ***Brandon Bonner***

Project History

The Center for Telecommunications Education and Research (CTER) at the University of Alabama at Birmingham is presently involved with research in the area of telepathology system architectures and performance. The purpose of the research is to aid physicians in the diagnoses of disease in microscopic tissue specimens. If the image acquisition component of the telepathology system had the potential to render preliminary diagnostic results in certain cases, this would allow the physician to make a complete diagnosis in a shorter period of time. This approach to telepathology is a primary motivation behind the Linux DSP Shell **(Bonner, 1999)**. This type of research requires a robust, stable platform that allows for real-time image acquisition and complex processing. The nature of the telemedicine application also requires the use of subsystems that do not adversely affect overall system performance. For these reasons, as well as for economic considerations, Linux was the environment of choice. However, image processing solutions for the Linux environment have been limited. This was the case until Wes Hosking et al., **(Hosking 1998)**, ported the Dipix Vision Library (XVL) to Linu x .This made it possible to pursue a real-time image processing system for the Linux environment.

Although the concept for this project arose from a sponsored research project related to telemedicine, the development of the prototype DSP Shell system was initiated as part of an author's Senior Design sequence in the UAB Dept. of Electrical and Computer Engineering. As such, the student's design objectives were parallel to that of the project. By combining results from development of an application platform for Linux with reporting on the culmination of the undergraduate experience in Engineering, we hope to expose interested readers to both the process of technical education and the excellent development properties of the Linux environment.

Objectives

The desired outcomes for the Linux DSP Shell, were defined at the outset but as the project progressed these ``flexible'' items tended to change. The current objectives are as follows:

- The initial set of commands available to the user should be commands that are considered highly important to image processing techniques.
- The modularity of the software should allow for flexibility and robustness of the system, including

robust error checking.

- The user should be able to control the system with shell-scripts.

Design Decisions

As mentioned previously, this project is part of a Senior Design sequence. As such, the ``time to completion'' is tightly constrained, and appropriate simplifications must to be incorporated in the problem statement. One such simplification results from noticing that the complexity and nature of this project lends itself directly to a hierarchical solution. To provide the user with a transparent interface, an interactive script is used instead of a completely integrated ``shell''. In the simplified DSP Shell, a daemon (long-lived background process) controls the XPG-1000 subsystem via the Dipix XVL library function calls, and the script interacts with the daemon through named p ipes(FIFOs) as shown in Figure 1.

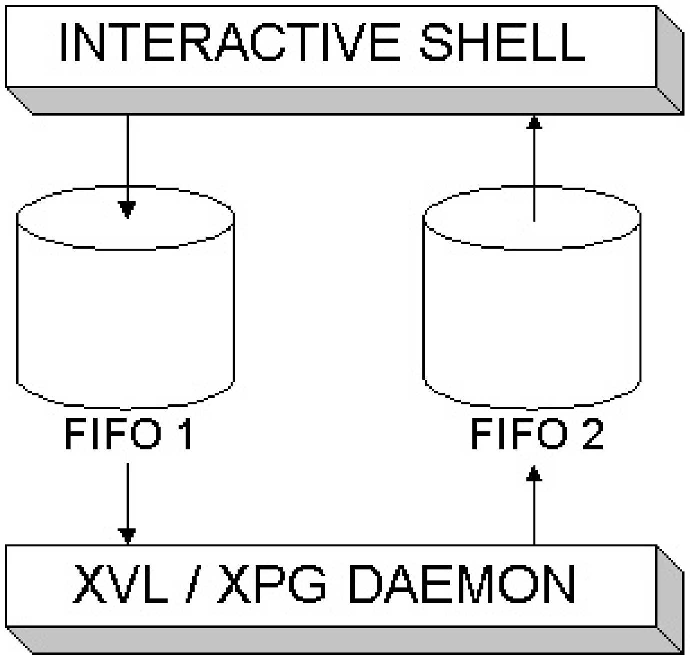


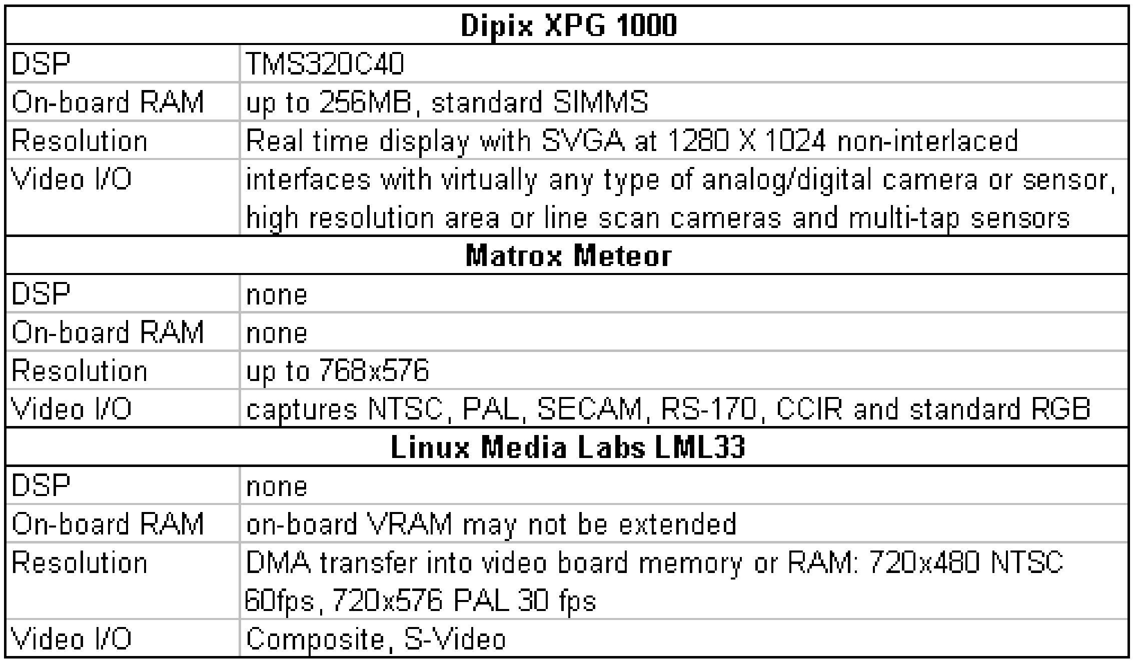
Figure 1. Diagram of the Linux DSP Shell architecture.

In addition to being straightforward to implement, this architecture has a de-coupling effect on the user-to-DSP interface. In effect, the embedded DSP, acquisition solution, and image processing library become a shared system resource which can be accessed by a variety of mechanisms.

PROJECT DEVELOPMENT

Image Acquisition In Linux And The Dipix XPG 1000

There are not many choices for sophisticated image processing solutions in the Linux environment. Table 1 compares the more powerful choices of imaging hardware that is available for Linux. The imaging hardware listed in the table are all excellent choices for image or video acquisition and/or processing. However, the Dipix XPG 1000 has particular features which make it far more powerful and versatile than the others. Key features of the XPG 1000 include the capability for simultaneous image acquisition and real-time processing, frame grabbing, and data transfer at up to 48 Mbytes per second, and straight-forward support for multiple standard or non-standard analog and digital cameras, and video signal formats. The camera acquisition modules (xCM) for the XPG 1000 interface with the XPG via a ``piggyback'' connection. Together, the XPG and camera interface module occupies a single PCI bus slot in the host computer. The variety of flexible, programmable camera acquisition modules allow the XPG to simultaneously interface with multiple analog or digital cameras.



Dipix XPG 1000	
DSP	TMS320C40
On-board RAM	up to 256MB, standard SIMMS
Resolution	Real time display with SVGA at 1280 X 1024 non-interlaced
Video I/O	interfaces with virtually any type of analog/digital camera or sensor, high resolution area or line scan cameras and multi-tap sensors
Matrox Meteor	
DSP	none
On-board RAM	none
Resolution	up to 768x576
Video I/O	captures NTSC, PAL, SECAM, RS-170, CCIR and standard RGB
Linux Media Labs LML33	
DSP	none
On-board RAM	on-board VRAM may not be extended
Resolution	DMA transfer into video board memory or RAM: 720x480 NTSC 60fps, 720x576 PAL 30 fps
Video I/O	Composite, S-Video

Table 1. Framegrabbers, DSPs, and real-time image processors for Linux

The Dipix XVL machine vision function library is also a significant consideration in contrasting the capabilities of various solutions. XVL contains over 200 DSP-based functions for real-time image acquisition, pro cessing,and control. Also included with the XVL package is an interactive Windows-based command-line environment that provides user access to most of the XVL functions. This software environment allows for interactive exploration of different imaging techniques and provides immediate results. Duplication and robust extension of this environment is the purpose of the Linux DSP Shell project.

Developing An Appropriate Interface To The XPG 1000

As constrained by the requirements of the design process, the interface to the XPG 1000 must accept commands that are abstractions of the functionality of the XPG 1000. For example, the user should not be concerned with initializing the hardware or allocating memory for an image buffer. The commands should also apply directly to image processing terminology. For example, if the user wanted to ``snap'' a picture with the attached camera, ``snap'' could be the command issued to accomplish this. Some other examples of appropriate commands are ``acquire'' (save the image to disk) and ``filter'' (filter the image while stored in the hardware).

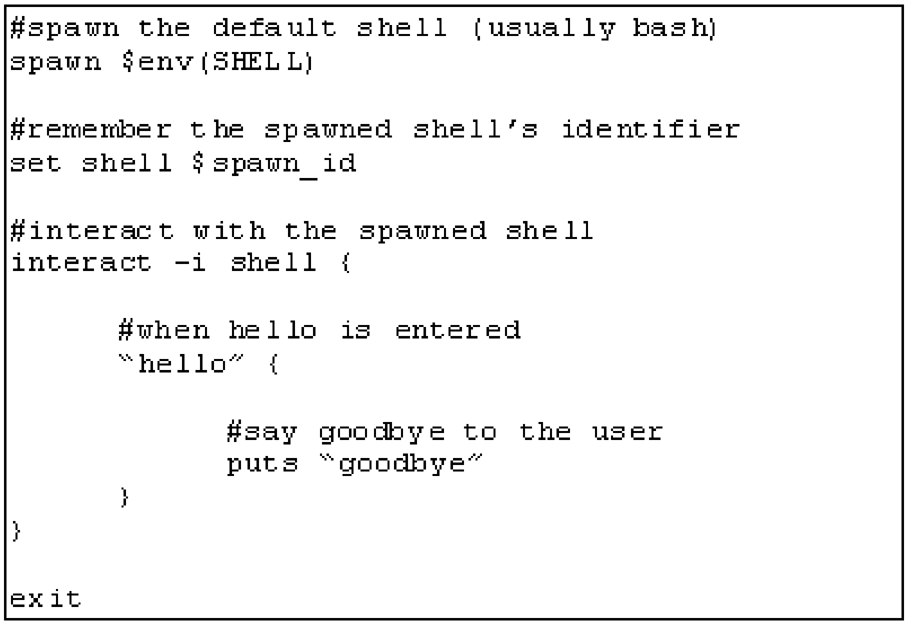
The Dipix XVL function library is accessible through C function calls. This limits direct access of the XPG 1000 to the use of software linked with the XVL libraries. This limitation causes difficulty in retaining the functionality of the user's shell. There are several solutions to this problem. The first of these solutions is the development of an entirely new UNIX shell. This implementation would allow for the full functionality of the XPG 1000 to be inherent in the shell itself. With this approach, a separate software package would not be required to interact with the XPG 1000 since commands could simply be entered at the shell prompt or placed in a file like other shell scripts. This solution would be the best of the alternatives, but to implement this functionality would entail the construction of an interpreter to parse the script structure, etc. So, this approach was not feasible for the Senior Design sequenc edue to time constraints. However, the final goal of the ongoing Linux DSP Shell project is to fully implement this shell with support for a range of DSP solutions in addition to the Dipix XPG.

A second solution that was considered was the use of many small programs. For example, if the user wished to filter an image, a program named ``filter'' could be called from the shell prompt. This solution would allow for modularity and functionality of the shell to be retained. However, the system loses efficiency and flexibility with this approach, since the burden placed on the host computer is too large.

After considering these and several other options, the system solution chosen was the creation of an intermediary interface to the XVL function library. Rather than completely integrating the user interface and XVL/XPG interface into a monolithic program, this interface would not abstract the XPG 1000's capabilities to the level desired for the user. Instead, this approach provides an interface to the XVL function library based upon input and output rather than C-language function calls. This effectively de-couples the C-language XVL library from the user-level application, since the input and output of this interface can be tied to the output and input of any other program by the use of interprocess communication. **Interprocess communication**, which will be discussed in detail later, is defined as communication between two or more processes or programs. This solution generalizes the interface to the XPG 1000 so that another program, or even a script, may have access to the XVL function library. This approach allows for the creation of a ``pseudo-shell'' which the user can invoke to overlay his current operating environment without losing command-line functionality.

Creating A ``Pseudo-Shell'' With Expect

Expect is an extension to the scripting language called TCL, otherwise known as the ``Tool Command Language'' **(Ousterhout, 1994)**. Expect was created by Do nLibes for the purpose of automating interactive programs **(Libes, 1995)**. An Expect script accomplishes automation of interactive programs by spawning, or starting, an interactive program or process. Then, using an ``interact'' command, the Expect ``pseudo-shell'' acts as a filter for input to the spawned process. When a string passes between the user and the spawned program (or vice-versa), it is first examined by the pseudo-shell. If a pattern in the string is recognized by the pseudo-shell, the script executes instructions defined by the author of that script. When the script does not recognize a string pattern, that string is passed directly through to the spawned process (or to the user). An example Expect script that interacts with a user shell is shown in Figure 2. In the example, the original command-line interface is spawned and an interactive ``pseudo-shell'' is placed between the user and the spawned shell. If the user enters ``hello'' at the command prompt of the pseudo-shell, the shell traps the string and echoes back the string ``goodbye''. If the user enters any other string or shell command for which a pseudo-shell action is not defined, the command passes directly through to the original shell for execution. This method can easily create a transparent user interface to the XPG 1000 by spawning the user's original shell as an interactive process. In this fashion, the user would be able to interact with his shell to issue ``usual'' commands, but all input would be examined by the Expect script ``pseudo-shell'' first. The Expect script would then appear to be invisible to the user. Also, functionality of the original shell would be retained because any shell command entered by the user would not be recognized by the script, and therefore would be passed on to the shell. Additionally, XPG 1000 functionality would appear to exist within the user's shell.



```
#spawn the default shell (usually bash)
spawn $env(SHELL)

#remember the spawned shell's identifier
set shell $spawn_id

#interact with the spawned shell
interact -i shell {

      #when hello is entered
      "hello" {

            #say goodbye to the user
            puts "goodbye"
      }
}

exit
```

Figure 2. Example Expect script which interacts with a shell.

Interprocess Communications

So far, we have defined a generalized interface to the XPG 1000 based on an XVL daemon and a user interface using an Expect script acting as a ``pseudo-shell''. The two interfaces will utilize a form of interprocess communication (IPC) for their interfaces. Currently the system uses named pipes as a simple form of IPC, however, there are several methods for achieving IPC in a Unix environment. In the future, the Linux DSP Shell will be implemented using System V IPC.

The System V IPC interface consists of semaphores, message queues, and shared memory segments. **(Goldt et al., 1995; Matthew et al., 1997**; **Volkerding et al., 1997)** **Semaphores** are used to ensure mutual exclusivity for a shared resource. This exclusive access prevents disasters from occurring when several processes are reading or writing to the same storage space. The processes will have to wait in turn before access is granted.

Message queues provide a way to send a block of data from one process to another. With this method, synchronization of reads and writes are not necessary since the message queue exists outside of the processes. However, there is a system-dependent limitation on the size of the data block that can be sent to the message queue as well as the number of blocks that may exist inside the queue at any one time.

Shared memory, as the name implies, allows two processes to share the same segment of memory. Shared memory is the fastest form of IPC. Shared memory allows for each process to immediately see any changes to data made by another process. However, this method is asynchronous. Synchronization of accessing the shared memory is left to the programmer. Such synchronization issues are usually handled through the use of semaphores.

One of the simpler methods of IPC is the use of pipes. Pipes are common in the UNIX environment. Pipes are so common that they may be used for redirecting the input and output of programs executed from the command line. Pipes can also be setup dynamically by a process. named pipes, or FIFOs (first-in, first-out), were chosen as the IPC method for this project. The use of FIFOs in this case is helpful because, relative to the other IPC methods, they are simple to implement and easy to use. Once a FIFO has been created, it exists within the file structure of the operating system. As a result, communicating through FIFOs is as easy as reading and writing regular files. FIFOs are sometimes referred to as ``special files'', because once a process reads the contents of the FIFO, the contents are removed and the FIFO is empty.

System Design

As discussed previously, the transparent user interface to the XPG 1000 is accomplished with the use of Expect. This allows the user to interact with his normal shell and simultaneously with the XPG 1000. The Expect script communicates with the intermediary interface to the XPG 1000 through the use of FIFOs. The intermediary interface is a special program linked to the XVL libraries. The XVL daemon runs as a background process so that it may be invisible to the user and available to all users without need for setup or allocation. The use of the daemon also allows for multiple interfaces to be presented to users without changing the XVL programming interface. Two FIFOs are used to allow two-way communication between the Expect script and the XPG daemon. Figure 3 illustrates this communication process. An example walk-through will further illustrate the command and processing flow of the DSP shell.

When the user enters a command from his shell prompt, the Expect script compares this command with high level image processing functions for the DSP shell. These functions are tokenized with names such as snap, acquire, and filter. If the command corresponds to an image processing function, it is trapped by the pseudo-shell for further tokenizing into simpler commands. The f irstof these simpler commands is then written to a FIFO, where the XPG daemon is waiting for such communications (e.g. blocking). Once the low-level command has been written to the FIFO by the Expect script, the XPG daemon reads the FIFO. This low level command may be broken into even lower level commands by the daemon in order to make the appropriate function calls through the XVL function library. Once the XVL functions return, the daemon writes information concerning success or failure of these ``atomic commands'' to a second FIFO, where the Expect pseudo-shell is waiting for the response. The pseudo-shell reads the information sent by the XPG daemon. This information is then relayed to the user to give notification of success or failure of the process. This procedure iterates until the high level command given by the user is completed, or failure of a part of the process occurs.

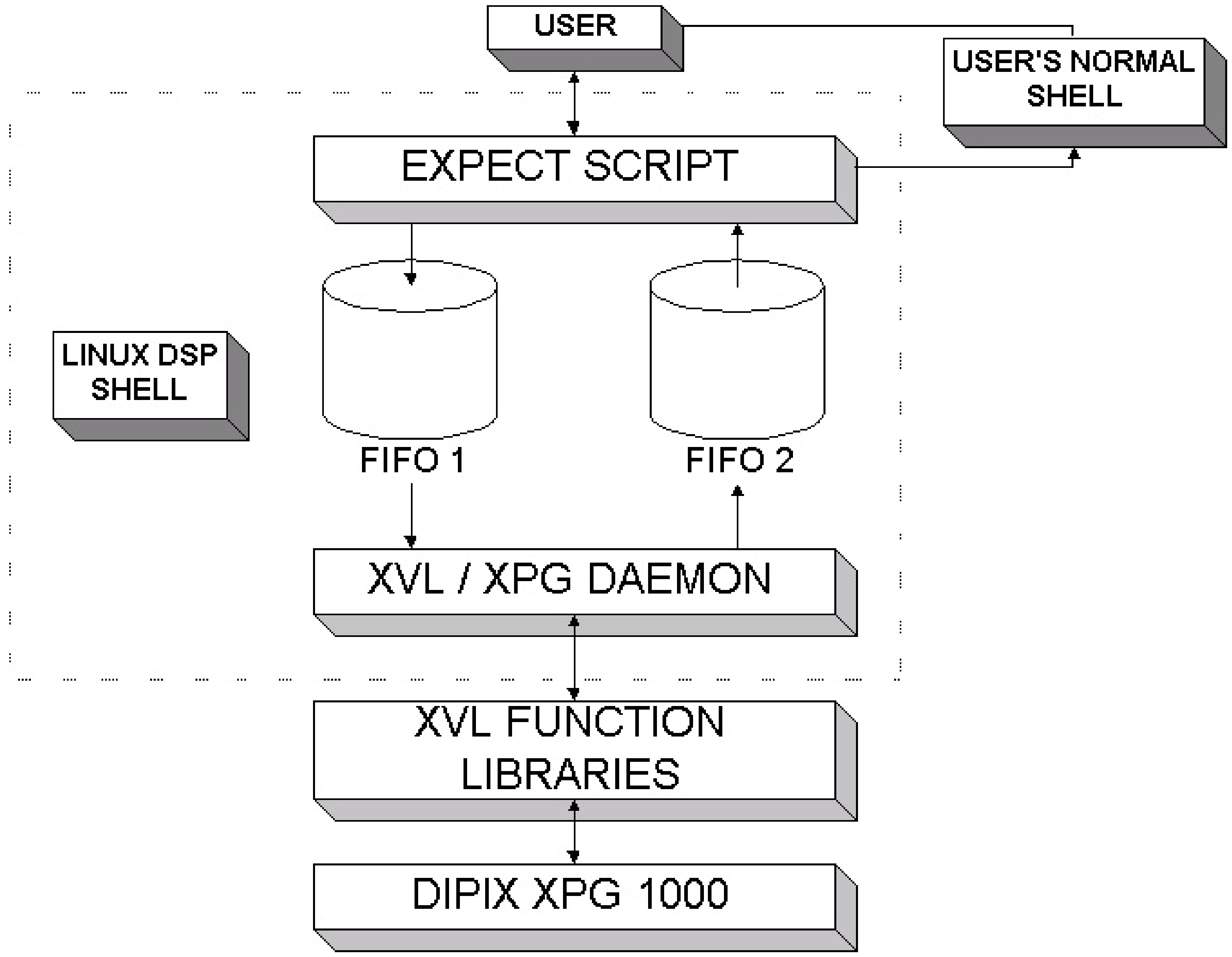


Figure 3. Linux DSP Shell system diagram.

Control Via Shell Scripts

This implementation of the Linux DSP Shell allows for intuitive control of a real-time, embedded image processing subsystem which can be implemented through simple shell scripts. Shell scripts are simply text files which contain sequences of shell commands. In the UNIX environment, one can automate common shell functionality with such scripts. Since the interface to XPG 1000 appears to exist within the shell, one may automate complex image processing and acquisition tasks. This capability would allow for rapid development of real-time image processing applications by creating the possibility for quickly testing an algorithm in an intuitive, flexible fashion.

Conclusions And Recommendations

In conclusion, the greatest value of this project is to build a robust, rapid application development environment for signal and image processing that is currently not available on any platform. Recommendations for future work would include full imple mentationof Dipix XPG 1000 capabilities. Also, the system should be implemented with the use of message queues and shared memory segments instead of FIFOs. Message queues would allow for the separate processes to be more efficient, in that they would not have wait for the other process to send information as is the case with simple FIFOs. Shared memory segments would allow the programs to exchange larger amount of data leading to a multi-user, multi-processing XVL daemon. The current goal of the DSP shell project is the Dipix XPG functionality. The project's overall goal is to extend this shell capability to other DSP solutions available for Linux. One way to do this is to abstract the most widely implemented DSP functions into a common API. From there, each individual DSP solution could include its own proprietary function module. With this technology the Linux DSP shell could be made to interact with other DSP cards as they become available for the Linux environment. Such a tool would prove to be valuable for developers of digital signal processing applications.

Acknowledgements

The authors would like to thank the following people: Dr. Stan McClellan (our fearless leader), Dr. Murat M. Tanik (for his endless vision and infinite wisdom), Linus Torvalds, GNU Project, Wesley Hosking, Coreco (fomerly Dipix), the Linux community, the members of the Linux Dipix mailing list, and our families and friends.

References

1
Bonner, B., Stricklen, M., McClellan, S. ``The Linux DSP Shell.'' To appear in *Conference Proceedings LinuxEXPO 99*.

2
Goldt, S., van der Meer, S., Burckett, S., Welsh, M. *The Linux Programmer's Guide.* March 1995. **http://linuxwww.db.erau.edu/LPG/index.html**

3
Hosking, W. Atlantek. **http://www.atlantek.com.au/USERS/wes/linux/frame.html**

4
Libes, D. *Exploring Expect.* Cambridge: O'Reilly & Associates, Inc., 1995

5
Matthew, N., Stones, R. *Beginning Linux Programming.* Birmingham, UK: WROX Press, 1997

6
Ousterhout, J. *Tcl and the Tk Toolkit.* Reading, Massachusetts : Addison Wesley, 1994

7
Volkerding, P., Foster-Johnson, E., and Reichard, K. *Linux Programming.* New York : MIS:Press, 1997

Biography

Michael Stricklen is a research assistant of Dr. Stan McClellan at the Center for Telecommunications Education and Research on the campus of the University of Alabama at Birmingham. He is currently involved with a research project funded by Bellsouth (which involves Linux, of course). When not staring

at a monitor he enjoys snowboarding, golf, and a number of other outdoor activities. He can be reached via e-mail at goose@uab.edu

Bob Cummings works as a student research assistant at the Center for Telecommunications Education and Research at UAB, where he is currently an undergraduate in Electrical Engineering. His interests include Perl programming, fishing, and percussion. He can be reached via e-mail at bahb@uab.edu.

Brandon Bonner works as a research assistant at the University of Alabama at Birmingham, where he is wo rkingon his degree in Electrical and Computer Engineering. In his abundant supply of spare time, he enjoys getting outdoors and playing musical instruments. He can be reached via e-mail at bwbonner@uab.edu.