A Merged UNIX/DOS Environment on the Intel 80386

David A. Butterfield

Locus Computing Corporation
3330 Ocean Park Blvd., Santa Monica, CA 90405

ABSTRACT

Merge 386 is a multiuser multitasking environment that allows a user to make simultaneous transparent use of the facilities of the Unix operating system and the MS-DOS operating system on an Intel 80386 based computer system. Users can have a one-system view, making use of application programs of both systems via the user interface of either system. Application programs written for either system can execute in the Merge 386 environment without modification or recompilation. DOS programs may communicate with Unix programs using Unix interprocess communication facilities.

1 The Merge 386 Operating Environment

Merge 386 is a set of independently implemented operating systems and modules brought together into a single coherent user environment. From the user's point of view, Merge 386 appears as a single operating system, from which he may execute the application programs that suit his needs. Some of these application programs may have been written to run under the DOS operating system and others for the Unix operating system. To the user there need be no distinction between the two. Thus, the Merge 386 system is said to be *transparent* to the user.

This *user-interface transparency* is valuable in a modern computer system because it allows the user to benefit from the services and application programs of the two operating systems without having to suffer the complexity of knowledge of an additional operating system. The cost in training and frustration is significantly reduced.

From the point of view of an application program, it is running under its host operating system, whether that be Unix or MS-DOS. This means that application programs can operate within the Merge 386 environment without any modification or recompilation. Thus, the Merge 386 system is said to be transparent to the application program.

This *program-interface transparency* is extremely valuable because it immediately widens the potential user-base of application programs written for one operating system to include the users of the other operating system, without requiring any new investment in development or testing of the application program.

If implemented carefully, the cost of providing this integrated environment is surprisingly small, both in the amount of code required to support it and the actual run-time overhead incurred.

This paper first explains the philosophy and technical objectives of the system. Following that is a discussion of the underlying architecture and how that architecture provides program-interface transparency, user-interface transparency, and the one-system view. Finally the implications of the design on the performance of the system are discussed. We conclude with comments on our actual experiences in building and using the system.

1.1 Program-Interface Transparency

The power of transparency and compatibility with an existing operating system can hardly be overestimated. It is commonly observed that the application base for the DOS operating system is orders of magnitude larger than the application base for the Unix operating system. It is also well understood that if the Unix operating system is ever to become as popular as MS-DOS, then the size of the application base will have to be increased.

One way to increase the Unix application base is to port all of the popular MS-DOS applications to run under Unix. However, the source code for most popular MS-DOS applications is not readily available, and even if it were, many DOS applications do not follow the programming rules that are normally employed when writing programs for a multitasking operating system. For example, some programs perform their own I/O instructions directly to the hardware, rather than making use of the operating system calls to perform these functions.

One of the objectives of the Merge 386 system, then, is to provide a level of transparency so high that MS-DOS application programs can run unmodified, without recompilation, in a protected environment on a Unix system. This is program-interface transparency.

1.2 User-Interface Transparency

As already mentioned, user-interface transparency reduces training costs and frustration. By the use of one integrated DOS/UNIX environment, the user can more easily and efficiently address the problems he has to solve. Therefore one of the objectives of Merge 386 is to provide this transparency and a one-system view.

1.3 High Performance

A third major goal of the system is to provide high performance response to the user and to external devices from a DOS application program. The user should see speedy response to his requests, whether they are CPU intensive, file I/O intensive, or external device intensive. An example of the latter would be a DOS program that is communicating with another system via an interrupt driven asynchronous interface. As we shall see, the architecture of Merge 386 is designed to address these performance issues without sacrificing any flexibility in the assignment of a particular external device to a DOS application program.

2 Merge 386 Architecture

The Intel 80386 supports several modes of operation. Most Unix applications execute in *80386 protected mode*, which is the processor's ''native'' 32-bit addressing mode. However, there is another processor mode called *Virtual 86 mode*. When a process is executing in this mode, instructions and addressing are interpreted by the 80386 in the same way that they were interpreted on an earlier Intel processor, the 8086. DOS programs are all written to execute on the 8086. By supporting the Virtual 86 mode, the 80386 provides the opportunity for software such as Merge 386 to run DOS programs in this mode, in a complete, protected *virtual machine* environment.

The Merge 386 software architecture is based upon a standard AT&T System V.3 Unix port for the Intel 80386 microprocessor. This Unix system has been enhanced to support the existence of processes that run in Virtual 86 processor mode. This support consists of code ''hooks'' that manipulate the data structures that instruct the processor to change modes under program control, or when interrupts or traps occur.

2.1 The Virtual Machine Monitor

When a process is executing in Virtual 86 mode, there are certain instructions that are considered ''privileged'' that cause a trap when they occur. When such a trap occurs, it causes control to be passed to a *Virtual Machine Monitor (VMM)*, which is a module that interprets and emulates all of the privileged instructions that could be executed by the Virtual 86 task. Examples of these are instructions that let the program enable, disable, or check the state of interrupts, and I/O instructions.

There are also certain addressing modes that cause a trap into the VMM; these must also be correctly emulated. By properly emulating all of these instructions and modes, the 8086 program can be made to run in an environment that appears to it to be an 8086.

2.2 The Virtual I/O Devices

One part of the VMM is the *I/O Monitor*. This module is responsible for decoding I/O instructions and determining what device the 8086 program was attempting to access. The I/O monitor builds a data structure containing this information and passes it to a module that knows how to emulate that particular device. This module emulates all aspects of a set of I/O ports, and is called a *Virtual Port Interface (VPI)*.

Each VPI is responsible for the emulation of some type of device in the system. For example there is an ''interrupt controller'' VPI and a ''COM'' VPI and a ''video'' VPI. A configuration table places particular VPIs at particular virtual port numbers. This table defines the configuration of the virtual machine. To emulate a particular machine requires the configuration table to attach VPIs to the same port numbers that the physical devices are attached to on the actual machine. For example, to emulate an IBM PC, there would have to be an ''interrupt controller'' VPI attached to virtual ports 0x20 and 0x21, because on the IBM PC the physical interrupt controller is attached to those port numbers.

The interface between the I/O Monitor and the VPI modules is well defined and documented, so that additional VPI modules may easily be written and added to the system at any I/O port number. This provides a high degree of flexibility in configuring virtual machines into the system.

Some VPI modules are self-contained and completely virtual. However, others require underlying physical support to be useful. For example, some 8086 programs communicate over "COM ports" which are connected to other computers via serial asynchronous lines. The COM port is a particular device at a particular address that is found on an IBM PC. However, the physical 80386 computer system may not have one of these devices; for example it might have an 8-port serial card of some sort.

Merge 386 defines an interface between the VPI and underlying Unix device drivers so that a virtual device can be emulated using a completely different physical implementation. In this case a virtual serial device of one type is being supported by a physical serial device of another type. However, one could build a driver that would allow the virtual COM device to communicate via a virtual circuit on a network, for example.

2.3 Direct Device Assignment

In the case where a physical device on the 80386 system exactly corresponds to the device that the Virtual 86 program wants to access, it is possible to directly assign the device to the Virtual 86 process, and disable the protection trapping for that device. In this case the trapping of the I/O instructions for that device is avoided and higher performance can be achieved. This is mostly useful for COM ports and floppy disk access.

2.4 File Service

The previous sections have discussed issues related to running any 8086 program under Unix using the Merge 386 VMM implementation. Any 8086 operating system or program can run in that system. In this section we will focus on the MS-DOS operating system in particular, and the integrated file system that Merge provides between Unix and DOS.

The file system is a standard Unix file system without any modifications. When a DOS application program issues a call to open a file, a software module (known as the *Bridge*) intercepts the system call and conditionally passes the request to a module running in 80386 protected mode (the *Server*). The Server issues a system call to the Unix kernel to open the requested file, and passes a file descriptor back to the DOS application via the Bridge.

Later, a *read* system call might occur on that file descriptor. When that happens the Bridge passes the request to the Server, which performs the *read*, and places the data into the address that the application program or the Bridge specified.

In this way, DOS programs can access files in the Unix file system. Together, the Bridge and the Server are responsible for translating all DOS file-system system calls into Unix system calls, and for translating the results back into DOS conventions.

3 The One-System View

One design goal for the system was to provide the ability for a user to execute Unix programs and DOS programs without necessarily knowing which were which. This section will address that goal from the Unix user's point of view. Under Merge 386, the user can type ''vi filename'' or ''ws filename'' to his Bourne shell (or other user interface) using the same syntax. The former causes the Unix editor *vi* to be executed, the latter causes the DOS editor *Wordstar* to be executed.

It should be noted that the user may also choose to use the system from a DOS point of view, or even a mixed DOS/Unix point of view using a ''hot key''. However this discussion will address the system from the Unix point of view.

3.1 Recognizing DOS Programs

The execution of DOS programs is transparent at the Unix system call level. That is, the Unix *exec()* system call accepts a filename, and the referenced file may contain Unix executable code or DOS executable code. The recognition is done within the *exec()* system call, by examining the load module. If the load module is determined to be a DOS executable file, then the Merge software is invoked to create the Virtual 86 environment and to start the DOS program running within it.

The advantage of having the *exec()* call recognize the DOS load module is substantial. Any Unix program that executes another program can transparently execute a DOS program as easily as it can execute a Unix program, with no knowledge that a DOS program is being started. This means that such Unix programs as the shells, *make, vi*, a windowing system, or any other program that *exec*s other programs can start DOS programs without being modified or recompiled.

3.2 Starting a DOS Application

When a DOS program is *exec*ed, the Virtual 86 environment is created and a copy of the DOS operating system is placed into it, along with the Bridge (discussed in a previous section) and some initialization code. This initialization code is passed information regarding the current environment of the user, such as his current working directory, his environment variables, and so forth. It is also passed the name of the DOS program that the user wanted to run, along with any parameters that were specified.

The initialization code uses this information to set up the DOS environment, and then executes the requested application program, passing it the parameters. In this way, the command that the user typed to his Unix system is able to cause execution of a DOS application program.

4 Performance

A critical design requirement for the Merge 386 system was that it perform well. Performance considerations become very important when running multiple virtual machines, because each machine can be running an operating system that ‘‘thinks’’ it has the entire machine to itself. Even without this problem, it is important that the most common privileged operations do not incur much overhead. Also, it is important when external interrupts occur that are to be delivered to a virtual machine, that they be delivered with a very short latency time and that there not be a lot of overhead associated with delivering them.

Merge 386 addresses all of these issues. When a privileged instruction occurs, a trap is taken through a very fast path to the VMM which decodes the instruction, emulates it, and returns directly to the code following the faulting instruction.

To keep interrupt latency to a minimum, and to allow very fast access between virtual devices and their underlying physical devices, the VMM and the VPI modules are located in the Unix Kernel (loaded as device drivers). This allows the VPI modules to communicate directly with the drivers that manipulate the physical I/O devices, without a system call interface and Unix device I/O layer between them.

The VMM and the VPI modules also keep information regarding the recent behavior of the Virtual 86 process to aid in changing the scheduling priorities. This is used to help solve the problem of Virtual 86 programs that poll waiting for user input.

5 Implementation Challenges

Implementing transparent DOS execution under Unix in the Virtual Machine environment presented several difficulties. One major area was in the use of the Unix file system by DOS programs. The program-visible file system interfaces of the two operating systems do not exactly match. For example, the length and character set of a filename in a DOS file system are more restrictive than those of a Unix file system. This means that there are some Unix filenames that cannot be directly used by DOS application programs. Merge 386 addresses this issue by providing a mapping for those filenames that would not otherwise be accessible from DOS.

Another example is that a DOS application program may open an arbitrary number of files. Some DOS applications search entire directory hierarchies, opening file after file and never closing any. Merge 386 addresses this problem also, by caching file descriptors. There are many other examples of such differences between DOS and Unix.

Another problem is that many many application programs make use of various undocumented side-effects of MS-DOS system calls. Each of these effects must be properly emulated in the Merge 386 file service routines, in order to allow those programs to run transparently.

Another major area important to performance is the set of issues regarding running a single-tasking, single-user operating system and its applications as a task under a multitasking multi-user operating system. The Merge 386 system has to address the problem of determining the proper scheduling parameters for a DOS process that may be polling an I/O device looking for input, or may be performing a computation, or both. These problems are very difficult in general, but specific common cases can be handled by algorithmic means.

6 History, Experiences and Conclusions

Work on the file-integration portion of Merge 386 began in 1983 and was first run as an Ethernet-based DOS/Unix networking system in that year. The problems associated with file system integration and transparency were discovered and solved at that time.

In 1984, a version of Merge was first built using the Intel 80286 processor. In that system only a single DOS process could run at a time, due to processor limitations. However, it was at that time that the software was designed that provides the one-system view to the user. Some of the virtual-device emulation code was also written for that system. By the completion of that project, program-interface transparency, user-interface transparency, and the one-system view were implemented.

The port of the existing software to the Intel 80386 and the implementation of the virtual machine monitor software was done in 1986. The problems that had to be solved in that effort were mainly related to performance in a multiple-DOS environment, since the previous work had already solved the other major problems. Mercifully, we didn't have to solve them all at once.

At Locus, we use the Merge system for our own program development. Some of our programs are compiled under the DOS system to run as part of the DOS virtual environment. We use Unix editors and the Unix *make* utility to do our program development. The *makefile* contains rules that know how to make a DOS format *.exe* file from a *.c* or *.asm* file, using the DOS compilers that run as DOS programs. These DOS compilers are *exec*ed from the make program transparently, without any knowledge by the make program that they are DOS programs. We find this far more convenient than having to use different tools for our DOS development than for our Unix development.

Our experience with DOS/Unix integration has been an unqualified success. We have successfully run hundreds of DOS application programs under the DOS/Unix file integration system, thus validating the correctness of the implementation. Feedback from users regarding the one-system view has been very positive. Unix users especially appreciate the widened application set that the system provides to them. By bringing together the Unix and the DOS worlds, Merge 386 provides a more powerful and complete computing environment to Unix and DOS users alike.