This paper illustrates one aspect of the work performed at the Naval Surface Weapons Center/Dahlgren Laboratory emulation facility and represents the culmination of much prior research into the use of emulators, virtual machines, and virtual machine monitors at the Center. A description of an experimental Virtual Machine Monitor capable of controlling heterogeneous machines is presented. Future work into the uses of virtual machines and virtual machine monitors with respect to military applications as well as commercial applications involving network configurations of small machines is discussed.

Introduction

Several papers have been written concerning the use of emulation and/or virtual machines within the U.S. Navy. [RUSSELL 79, PERRY 76, GOLDBERG 78]. Virtual machine concepts as applied to research done at the emulation facility can be considered in the class of "integrated emulation" [GOLDBERG 79]. This class implies user microprogrammable machines without a preferred native mode.

Prior to CPX (Experimental Control Program), only a single emulation was able to run under the control of the Emulation Aid System (EASY) [FLINK 77]. EASY is an operating system which runs on the Nanodata QM-1 microprogrammable computer. [NANODATA 76, SALISBURY 76]. The system was developed to support the construction of computer emulators (i.e., it permits a high level transformation of the QM-1 computer to a functional duplicate of another computer).

Specialized user-interface programs or "control programs" (CP's) were written for each emulator to permit them to run under EASY. Control programs developed include those for the CDC 200 UT emulator, the UCSD (University of California at San Diego) Pascal-P machine emulator, the AN/UYK-20 military computer emulator, the AN/UYK-7 military computer emulator, the Trident Basic Processor (TBP), and the E-Machine (Intermediate Language Machine for the SIMPL-Q programming language) [PERRY 78].

A shortcoming of these CP's was their design as mechanisms to support single target emulators with no provision for communication among future multiple emulation experiments. This paper describes the features of a "generalized" control program which has been termed as CPX. It should be noted that the design issues addressed by the CPX system have important implications beyond this specific implementation on the QM-1 computers.

Design Goals

CPX, together with EASY provides integrated support for: interactive control of multiple concurrently resident emulators; virtual, as well as, real input/output of the emulators to the various real peripherals on the QM-1 computer; and diagnostic displays for use in debugging of both emulators and software running on the emulated computers.

As a Virtual Machine Monitor (VMM), CPX may be perceived of as a resource manager which can provide to each virtual machine or "processor" a part of the real machines resources. The virtual machine behaves as if he is the sole user of the machine. Each processor running under CPX is expected to furnish its own dedicated operating system (e.g., fault handling, interrupt structure). Thus, a given virtual machine (e.g., AN/UYK-7 machine), controlled and monitored by CPX, is capable of executing software native to the real machine as though it were on a dedicated real machine.

The design goals of CPX are to provide an environment and mechanisms that allow:

1. experiments with the emulations of various network configurations to be efficiently run.
2. allocation of the real machine resources (e.g., memory, virtual and real devices).
3. protection.
4. interprocessor communication.
5. "real-time" software to be run in "simulated" real time.

These goals are illustrated further in the Applications section.
Major Functions of CPX

The specific functions of CPX are the following:

1. control and monitor up to 7 concurrently executing virtual "processors",
2. provide mechanisms for the allocation of a target processor's memory independent of the real machine's word size,
3. support of virtual devices (e.g., printer, consoles),
4. allocation of real devices (e.g., tape drive, disk, printer),
5. support for a "synchronizing" clock to maintain synchronization among selected processor emulations,
6. provide mechanisms to support interprocessor communications,
7. provide mechanisms to support memory protection among all running processors,
8. display state information about all processors as well as status of interprocessor communications,
9. provide a checkpoint/restart facility to capture and restore the state of a given network configuration.

Overview of CPX

The design of all system software developed on the QM-1 computer has been conceived of and implemented in a strictly hierarchical manner using current software engineering principles. The structure of the CPX-based system consists of the following components:

**Figure 1 - System Elements**

All system software runs on the Nanodata QM-1 Computer. The QM-1 has a two-level memory hierarchy consisting of a fast (150 nanosec.) read-write semiconductor memory known as "control store" and also a slower (750 nanosec.) core memory known as "mainstore." Emulators developed for the QM-1 reside in the fast control store whereas mainstore may be used as memory space for target machines or special system software.

TCP [NANODATA 78] is a totally microprogrammed executive developed by Nanodata. TCP resides in control store and provides the necessary support for the managing of "tasks" or emulators as well as performing basic resource mapping, low-level I/O functions and service functions of value to the emulator designer. However, TCP does not provide any of the functions usually associated with an operating system such as console interface, command interpreting, status or error reporting, etc.

A high level language, SIMPL-Q (extended to utilize special features of TCP), is used as the standard systems programming language. The SIMPL-Q compiler produces a type of source program known as an "intermediate language." The E-Machine, a microprogrammed emulator also residing in control store interprets or executes the "intermediate language." The E-Machine is specifically tailored to support the SIMPL-Q architecture.

Thus all user interface functions, as discussed above, are written in the SIMPL-Q language and make up what is known as the EASY Operating System. It may be seen now that functions which are executed frequently are implemented via microcode in TCP, but functions more easily implemented via a high level language are performed by mainstore resident modules. All CPX software resides in mainstore and is written in the SIMPL-Q language.

The CPX Virtual Machine Monitor operates as a subsystem of the EASY Operating System. To enter the CPX system, the user merely types "CPX" at his terminal. At this point a number of commands become available to configure, monitor, and control an "emulated multiprocessor" environment. These commands are known as the "User/CPX Interface."

CPX knows of emulators running under its control only through specialized software known as the "CPX/Emulator Interface." In order for an emulator to run under CPX, the interface module must conform to a predefined set of rules.

Thus all functions done previously by the emulator designer such as "initializing", "starting", and "halting" of tasks or emulators are now handled by CPX. In addition, CPX allocates control store and mainstore memories for target processors, as well as, various virtual and real devices. It should be noted that multiple emulators may reside concurrently in control store with the multiplexing of cpu cycles accomplished by TCP.
It can be readily seen that CPX can run different machine architectures (e.g., UNIVAC, INTEL 8080, DEC PDP 11, UCSD PASCAL and others) as long as an Emulator Interface is written for each emulated machine. Thus, a large assortment of multiprocessor configurations may be run concurrently.

For example, the user may partition his resources (e.g., memory and three consoles) into two or more different virtual machines (e.g., UNIVAC AN/UYK-20 and AN/UYK-7). Each real console corresponds to the operator's console of the real machine.

**Novel Features of CPX**

1. **Interprocessor Communication**

   Mechanisms exist within CPX to allow processors to communicate with other processors as long as an "I/O protocol" can be defined. Basically, the emulator designer must define what functions are necessary to allow the emulated processor to "talk" with another processor employing the same I/O protocol (e.g., Naval Tactical Data System protocol or "NTDS").

   Each processor communicates to other processors via a "virtual I/O" interface to CPX. The interface is handled by CPX and is transparent to each processor. The mechanisms used to implement the protocol is called the "specialized" path handler. From Figure 2, it can be seen that VM1 and VM2 can talk to each other via the "A type path handler", while VM3 talks via the "B type path handler" to VM4.

![Figure 2 - Example of Multiprocessor Configuration with different I/O protocols](image)

The path tables illustrated in the previous figure are general enough to be used by any "specialized" path handlers.

2. **Processor Synchronization**

   Much criticism of virtual machine systems for use in tactical applications centers on the overhead incurred by the virtual machine monitor and the inability to run what is known as "real time" software.

   A device implemented via CPX and known as the "synchronization clock" allows designated processors to be synchronized. Each processor in the subset is given a number of time units (1 time unit = 100 μsec). The processor runs until its time units are exhausted, at which time it waits until all other "synchronized" processors use their CPX "time slices."

   Thus, it may be seen that the speed of the emulated processor will potentially have a minimal effect on the running of "real time" software, since all processors run in "simulated" real time (i.e., real clock time is simulated).

   The following figure illustrates two processors running with synchronization and one processor not in synchronization. Thus, each processor will get CPU cycles according to the implemented scheduling algorithm. When processors A and B are "paused" waiting for new time slices, processor C will get all CPU cycles temporarily.

![Figure 3 - Example of Processor Synchronization](image)

**Example of Network/Multiprocessor Configurations**

The design of CPX was conceived of in such a way that experiments with a multitude of network configurations can be run. Potential uses of this implementation of CPX are limited only by the available resources of the Nanodata QM-1 (e.g., only seven sons under each copy of CPX) and time constraints. Military as well as commercial applications can be run via CPX. Software (real time or otherwise) will run in the CPX network configuration just as it would run in the "real" configuration.
The following example illustrates a possible use of CPX:

![Diagram of CPX](image)

Figure 4 - Military configuration involving a Command and Control System (C&CS) on a multiple bay UYK-7 and a Weapon System Simulation Program running on two UYK-20 computers.

**Applications**

The work presented in this paper is of an experimental nature and as such there is much more to be learned in how to optimally control real time system configurations. Problems addressed by CPX with respect to interprocessor communications and synchronization of processors must be successfully confronted in order to utilize the concepts of virtual machines with Navy tactical systems. CPX can provide a good first look at difficulties which may be encountered with real time systems. Through the use of emulation, it is not necessary to have actual real hardware (e.g., AN/UYK-7 computer) in order to experiment with different configurations of tactical systems.

Two experiments are currently being planned which will utilize the unique capabilities of CPX. The first is to actually run a Navy tactical fire control system on the Nanodata QM-1. The configuration is similar to that shown in Figure 4. The second experiment involves the Nanodata QM-1 running tactical software on the AN/UYK-7 emulator communicating with a second physically separated computer running the AN/UYK-20 emulator using a "virtual NTDS" I/O protocol. (i.e., NTDS protocol via the ARPANET protocol).

It is expected that this work will provide some data needed to successfully analyze what other features are needed in order to construct a virtual machine monitor which is capable of controlling actual real time Navy systems. A major problem expected to be encountered are the real time constraints of tactical systems. If it can be proved that our synchronizing mechanisms are sufficient to run real time software, then the CPX design implemented on faster machines (e.g., the Lawrence-Livermore S1 Computer) can surmount the overhead problems presented by a virtual machine monitor and thus the software will be capable of running in actual real time. Another area that could present difficulties is the "sharing" of memory by different processors. Again timing and control play a key factor in eliminating this problem.

The use of CPX-like virtual machine monitor is not limited to military applications but could also be used to control networks of real time process control machines.

**Acknowledgments**

A number of people have made significant contributions to CPX and to this paper. Special thanks go to Mr. Charles Flink of the Naval Surface Weapons Center for providing the essential technical expertise which enabled CPX to actually be developed. The authors would like to express their appreciation to Mr. Walt Warner, Mr. Harmon Thoms and Mr. John Perry of the Naval Surface Weapons Center for their helpful comments and review of this paper; and finally to Mr. Ralph Niemann, without whom there would have been no virtual machine projects.

**References**