An Overview of the BLITZ System

Harry H. Porter III
Department of Computer Science
Portland State University

Introduction

The BLITZ System is a collection of software designed to support a university-level course on Operating Systems. Over the course of one or two terms, students will implement a small, but complete, operating system kernel. The BLITZ software provides the framework for these student projects.

The Components of BLITZ

The BLITZ System is composed of the following components:

- BLITZ Processor Architecture
- Virtual Machine (Emulator)
- Assembler
- Linker
- KPL Programming Language
- KPL Compiler
- Specifications for the Student Project
- Support Code for the Projects
- Documentation

Background and Context

OS kernels execute on bare machines, running no other software. However, developing a program to execute on a bare machine is impractical for a university-level course for several reasons.

First, modern processors are too complex for students to master on the fly and require courses in their own right. In particular, an OS kernel must be concerned with the details of I/O and memory-management hardware, which require detailed, specialized expertise.

Second, it is difficult to develop and debug software on a bare machine. Without a sophisticated cross-platform development environment, the basic steps of compiling, loading, executing, and debugging become very difficult.

Third, most students do not have an extra bare machine to experiment with. Computers today are already fully loaded with operating systems and important user data. Experimenting with software that reads and writes directly to the disk is not a reasonable option.

In light of these difficulties, OS courses take several alternate approaches. First, students are asked to create and/or experiment with a "shell" program. Second, students are asked to simulate and measure in isolation various OS algorithms (such as scheduling algorithms). Third, students are asked examine and/or modify some small section of a real operating system (e.g., modify Linux device drivers). Nonetheless, many students learn by doing and there is no substitute for implementing a complete—if minimal—kernel.

The BLITZ Approach

The BLITZ software provides a virtual machine for students to use. The system includes a CPU architecture. This virtual CPU is similar to familiar processors like the PowerPC, Pentium, Sun Sparc, etc., except that it is somewhat simplified.

There is no real BLITZ computer. Instead, the machine is virtual. A piece of software, called the BLITZ "emulator" or "virtual machine", simulates the BLITZ computer. The emulator runs on a host machine, such as a Unix box or an Apple Macintosh.

Students will edit, compile, and link their OS kernel projects on the host machine. Then they will execute them on the virtual machine, using the emulator. When bugs occur in the students' code, the emulator will display various error messages and the students can use the emulator tools during debugging.

The BLITZ Architecture

The CPU is a RISC design, modeled loosely on Sun's Sparc architecture. The BLITZ processor contains 32 general purpose integer registers, of 32-bits each. The processor also contains 32 double precision floating point registers, and several other special purpose registers, such as the "status register", the "program counter", and some page table registers.

The processor executes in one of two modes, called "system mode" and "user mode". Kernel code will run in system mode, while application programs will run in user mode. The use of modes allows the kernel code to be protected from malicious user programs; any attempt by a user program to execute one of the privileged instructions will be caught by the CPU and will cause a trap into the kernel.

An OS kernel will provide each user program with a virtual address space. Since each user program runs in its own virtual address space, several different user programs can run simultaneously without interacting or interfering with each other. To support the implementation of virtual address spaces by a kernel, the CPU includes a memory management unit with page tables. Real page table hardware can be quite complex. While the BLITZ design is simpler, it nonetheless accommodates “dirty bits”, “referenced bits”, “writeable bits”, “valid bits”, and logical-to-physical mapping. The BLITZ architecture can accommodate virtual address spaces of up to 16 MBytes and physical memory of up to 4 GBytes.

The BLITZ machine includes 2 input-output devices. The first is a disk. The emulator simulates the disk by using a file on the host machine. In other words, the disk data is stored in a host file. When a program running on the BLITZ machine reads or writes a sector from/to the BLITZ disk, the data is transferred from/to a file on the host’s file system. The BLITZ disk includes tracks and sectors and the emulator models seek, rotational, settle, and transfer times, as well as transient disk errors. The disk data is transferred using DMA (direct memory access).

The second I/O device is a terminal, which the kernel and user programs can use to communicate with a human user. The BLITZ terminal is a character-oriented serial device, such as an RS-232 interface. ASCII character codes are sent and received asynchronously, one at a time. The emulator normally passes the terminal I/O directly through to the host computer’s user interface, so the students can interact directly with the running BLITZ code, but the terminal I/O can also be directed to host files, to facilitate automated testing.

Both serial and disk units are controlled through memory-mapped I/O registers. Both devices signal completion of I/O with hardware interrupts.

The CPU supports a number of interrupts, including asynchronous hardware interrupts (such as disk, terminal, and timer interrupts) and synchronous program exceptions (such as “page invalid”, “privileged instruction violation”, “alignment exception”, and “address/bus error”). In addition, the architecture includes a software “trap” instruction, which a user program can execute. The trap instruction allows a user process to enter the kernel, as would happen during the invocation of a kernel routine.

The BLITZ architecture is designed to model existing processors realistically. Since the goal is for students to learn how a kernel interacts with the underlying hardware; the BLITZ model is realistic enough to allow the implementation of an OS kernel. In theory, the BLITZ processor could be fabricated in silicon and the student OS code would run directly on this machine, instead of being emulated.

Nevertheless, when compared to real processors, the BLITZ architecture has been simplified in ways that don’t detract from the project’s goals. Real processors are highly optimized for execution performance and this adds tremendous complexity. In contrast, the BLITZ architecture does not include any hardware memory caching or instruction pipelining, since these features don't impact the understanding of basic OS concepts. The BLITZ system includes a disk and a terminal device, but the vast complexity of I/O devices and drivers is avoided. The BLITZ architecture has a word alignment requirement, but no half or double word alignment. Although the BLITZ instruction set is simpler than most real processors, the BLITZ processor has 113 distinct machine instruction op-codes, representing 49 different instructions.

The BLITZ Virtual Machine

The BLITZ virtual machine is emulated. The emulator tool supports the full architecture, including the disk and terminal I/O devices, the page table memory mapping hardware, and the interrupts. The debugger is built into and is an integral part of the emulator.

The emulator can be run in command-line mode, where the students can type commands one at a time. These commands can be used to debug their BLITZ kernel code. At a low-level, the students can, for example, single-step CPU execution and they can examine and modify registers and memory. At a higher level, students can look at a trace of the activation frame stack and look at the values of program variables, displayed in source-code form.

The emulator also maintains some statistics, such as number of instructions executed, disk reads and writes, and so on. The emulator uses a simple model of time: the clock advances in discrete fixed-size units. Each instruction takes exactly one clock unit to execute.

The BLITZ Tools

The following pieces of software comprise the BLITZ system.

blitz
The BLITZ virtual machine emulator and debugger

asm
The BLITZ assembler

lddd
The BLITZ linker

dumpObj
A tool to examine BLITZ object and executable files

diskUtil
A tool to manipulate the initial file system on the emulated BLITZ disk

kpl
The KPL compiler

In addition, some of the code for the OS kernel project is provided. Writing a complete OS kernel is a large undertaking. Students will write additional code and will link it with the provided code to produce their kernels.

KPL: A Kernel Programming Language

The BLITZ system uses a high-level programming language called “KPL”. This language was designed specifically for use by students in the BLITZ OS project.

In some respects, the KPL language is low-level like “C” or “C++”. The language allows programmers to work directly with pointers and to manage and manipulate memory directly. The language is also facilitates cross-linking with BLITZ assembly code programs. KPL allows the programmer to work with pointers to functions, which is useful when implementing threads.

In other respects, the KPL language is high-level like Java. It includes objects, classes, and interfaces. The language also includes a mechanism for parameterized classes. (Parameterized classes are called “template classes” in C++.)

KPL includes a try-throw-catch mechanism, similar to Java’s try-throw-catch mechanism.

KPL supports separately-compiled components, called “packages”. Each package consists of a specification and an implementation. A package may use other packages, in which case the components of one package become available in another package. Each compilation processes a single package implementation, although the specification parts of other packages will be consulted. The system includes a mechanism to ensure that each implementation matches its specification and that all packages have been compiled using current, up-to-date specifications, preventing the use of out-of-date modules and enforcing the requirement that an executable must be composed of consistently-compiled parts.

The philosophy of the KPL language derives from its use by students creating their OS kernel project code. Ease of programming and debugging are emphasized above all. As such, great effort has been made to catch all programming errors, to catch them early, and to produce clear and meaningful error messages. For example, at runtime, the system checks all pointer usage (pointers must be non-null) and all array accessing (no out-of-range indexing). Additional effort is taken at runtime so that error messages can provide accurate source code line numbers and so that variables values can be displayed clearly.

As students create their OS kernel code, they will make use of existing support code, which is provided in the form of pre-written KPL packages. The package specifications of the KPL language are especially important in helping students understand existing code as well as providing a clear and unambiguous requirement of what is expected of the implementations the student are to create. Since students will need to read and understand existing KPL code, the language syntax has been designed to promote readability and clarity to every extent possible.

The OS Project

During the course of an operating systems class, students will be expected to create an OS kernel for the BLITZ architecture. The complete OS design and some support code will be provided for them. Their task will be to (1) understand the overall design, (2) understand the existing support code, and (3) create new code to complete the kernel.

The OS project is broken into several programming projects. Each programming project will take about 2 weeks and will build on earlier projects.

The first project concerns threads and concurrent programming. As this is the students' first introduction to the BLITZ system, this project consists of several pieces. A "thread package" is provided to the students; in the first piece, students get the package running and make small incremental changes. Then, students will implement solutions to several common concurrent programming problems like the "dining philosophers" and "producer-consumer". In this project, students get experience with semaphores, locks, and the monitor concept.

The second project concerns the implementation of user-level threads. In this project, the distinction between kernel code (running in system mode) and user code (running in user mode) is introduced.

In the third project, virtual memory is implemented. User-level programs will now run in their own virtual address spaces. This project will require the kernel to maintain the page tables and service page faults.

In the fourth project, a file system is added to the growing OS. User-level programs will now be able to invoke kernel routines to read and write to files. The OS will turn these requests in disk operations and will schedule them.

Documentation

The following documents describe the BLITZ system:

An Overview of the BLITZ Computer Hardware (8 pages)
This document introduces the architecture of the emulated CPU and I/O devices.

The BLITZ Architecture (67 pages)
This document describes the BLITZ processor hardware. It includes information about the CPU registers, the instruction set architecture, and the BLITZ assembly language.

Example BLITZ Assembly Program (7 pages)
This is a compete, stand-alone BLITZ assembly program. This program can serve as a test of the BLITZ tools and an introduction to using the emulator.

BLITZ Instruction Set (3 pages)
This document contains detailed information on each of the BLITZ machine instructions. It will be of interest primarily to assembly language programmers.

The BLITZ Emulator (46 pages)
This document describes the BLITZ virtual machine emulator and debugger. It shows how to run a BLITZ program and describes all of the available debugging commands. It also includes detailed information about the disk and terminal I/O devices.

An Overview of KPL, A Kernel Programming Language (66 pages)
This document describes the "KPL" high-level programming language.

Context-Free Grammar of KPL (7 pages)
This document contains a formal specification of the grammar of KPL and can be used as a quick reference, for syntactical questions.

The Format of BLITZ Object and Executable Files (13 pages)
This document describes the files produced by the assembler and the linker. This document may be of some interest to students, but is not necessary.

BLITZ Tools: Help Information (13 pages)
Each of the BLITZ tools will produce some "help" information; this document collects such information from each of the tools. This document also includes detailed information about the syntax of BLITZ assembly language programs.

BLITZ Instruction Set Summary (1 page)
This document provides a quick overview of the different BLITZ instructions and their formats. It will be of interest primarily to assembly language programmers.

BLITZ Instruction Set – Alpha List (2 pages)
This document lists all BLITZ machine instructions. This list is sorted alphabetically by instruction name. It will be of interest primarily to assembly language programmers.

BLITZ Instruction Set – By Op-Code (2 pages)
This document lists all BLITZ machine instructions. This list is sorted numerically by op-code. It will be of interest primarily to assembly language programmers.

BLITZ Misc. Technical Notes (7 pages)
This document contains a number of miscellaneous comments and notes. This document may be of some interest to students, but is not necessary.