

XOS : Experimental Operating System

Shamil C. M.
National Institute of Technology
Calicut, India
shamil_bcs09@nitc.ac.in

Sreeraj S
National Institute of Technology
Calicut, India
sreeraj_bcs09@nitc.ac.in

Vivek Anand T.
Kallampally
National Institute of Technology
Calicut, India
vivekanand_bcs09@nitc.ac.in

K. Murali Krishnan
National Institute of Technology
Calicut, India
kmurali@nitc.ac.in

ABSTRACT

This paper introduces an operating system project that helps undergraduate computer science students acquire an elementary understanding of the practical aspects of an operating system. The specification of XOS (Experimental Operating System) has been laid out for students to build it from scratch in a bottom-up manner. XOS runs on a simulated machine hardware with a very simple instruction set and native filesystem. Unlike other common instructional operating systems, the complete development environment including custom programming languages, debugger, file system interface and a detailed implementation roadmap is provided.

Categories and Subject Descriptors

K.3.2 [Computers and Education]: Computer and Information Science Education

General Terms

Design, Experimentation

Keywords

XOS, instructional operating system, experimental operating system

1. INTRODUCTION

Teaching operating systems has been a challenge at the undergraduate level. To tackle this problem several instructional operating systems like Nachos[2], OS/161[4], Pintos[5], GeekOS [1] etc. have been developed by various universities. Nachos[2] has been one of the most popular instructional

operating systems available and is being used in many institutes across the world [1]. Implementing Nachos is simple and it uses a mixed mode approach, where the operating system kernel is co-resident with the machine simulator and fused together as a single program. Nachos[2] and OS/161[4] runs on top of MIPS machine simulator. For these systems, a user's machine running on other platforms require cross compilers to MIPS.

Instructional operating systems like Minix and Xinu [1], provide a functional operating system on which modifications are to be done by students. Almost every other instructional operating system provides a skeleton of an operating system. However in XOS, only the specification has been laid out, and students learn to implement XOS from ground up using the tools provided. In this project, a simple high level language called APL (Application Programmer's Language) and its cross-compiler to XSM instruction set is provided to write user programs to test XOS. An XSM dependant language called SPL (System Programmers Language) and its cross-compiler is provided to program the OS itself. Most instructional operating systems use the UNIX filesystem for file management by the operating system. Instead, XOS provides a native file system known as XFS (Experimental File System). An interface between the UNIX filesystem and XFS filesystem is also provided.

XOS has features like multiprogramming, process management, a primitive filesystem and virtual memory. A sequence of stages are provided in a detailed roadmap which helps students to build XOS sequentially. Several simplifications have been made in XOS specification to make the system simple and manageable as a short-term project. These include absence of blocking system calls, inter-process communication, process synchronization, device management, file caching, file permissions etc. Only the fundamental data structures and functions of a single-user, multiprogramming operating system has been retained. The current specification of the machine as well as the operating system does not support process synchronization. It is hoped that future versions of this project will rectify this omission. The other features are not intended to be done in this platform. The project has been designed to provide an understanding of operating system concepts for a student undertaking a core

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

WOODSTOCK '97 El Paso, Texas USA

Copyright 20XX ACM X-XXXXX-XX-X/XX/XX ...\$15.00.

theory course in operating systems at undergraduate level. XOS is not scalable and students intending to implement advanced features like device management, file caching etc. must move to real platforms like Minix, GeekOS[1] etc.

In our experience with Nachos at junior year undergraduate operating systems laboratory course, we observed that students did not have the necessary programming expertise to comprehend the large code base of Nachos. We also observed that students faced a conceptual difficulty in understanding the separation between the operating system kernel and the machine simulator which are fused together as a single program in Nachos. XOS addresses these issues by allowing the student to build the operating system from scratch without having to learn an existing code base. The roadmap is designed in such a way that the student can read the roadmap and complete the project without any supervision. The project is expected to provide the student with a better insight into the operating systems concepts described in standard text books like [6]. Building XOS can be a supplementary project for a junior level undergraduate course in operating systems spanning over a 12 to 16 week term.

The primary components of the project include a simulated machine hardware (XSM), file system (XFS) and the operating system (XOS). No code base for the operating system is provided and the operating system is completely implemented by the student. Apart from the primary components, various tools are provided as part of the development environment. They include languages like Application Programmer's Language (APL) and System Programmer's Language (SPL) and their cross compilers to XSM instruction set, XSM debugger, and a UNIX-XFS interface to transfer files between a UNIX machine and the XFS disk (the XFS disk is itself a UNIX file).

2. EXPERIMENTAL FILE SYSTEM

The disk for XSM (which is a UNIX file) can be formatted using XFS or Experimental File System. XFS is the file system compatible with XOS. Since file system management is done by students building XOS, the disk organization must be easily understood and at the same time must give an insight on the data structures used in real file systems. Hence we chose to have a native file system for XOS.

The disk is formatted with this file system using the interface provided as part of the development environment. XFS is a simple file system with no directory structure. The data is organized into blocks of size equal to the page size in XSM memory. There are 512 blocks in XFS which holds the file and disk data structures, OS routines, user programs and data files. The various data structures in XFS include the Disk Free List which maintains information about used and unused blocks on the disk and the FAT or the File Allocation Table stores details of the files in the disk.

3. EXPERIMENTAL STRING MACHINE

XOS runs on a simulated machine hardware called XSM or Experimental String Machine. XSM uses an easy-to-understand native 2-address instruction set. The various components of the machine include registers, memory, timer and the disk. A UNIX file simulates the disk for XSM.

XSM has a timer which triggers after fixed number of instructions as compared to a timer interval in real machines. An instruction triggered timer was preferred over a clock-triggered timer to ensure that the timer interrupt is not invoked in between the simulation of a single instruction. Thus, an instruction in XSM is always atomic. Instructions are executed one after the other in a non-pipelined manner.

XSM has a memory of 64 pages. Size of each page is 512 words. A word is the smallest addressable unit in XSM as compared to a byte in MIPS. XSM is a string machine, and each word is stored internally as strings of size 16 characters. However, the XSM supports two data types, integer and strings and has instructions for both the data types. There are two privilege modes in XSM, the user mode and kernel mode. Switching between modes is done by instructions.

XSM instruction set supports load and store instructions to load data from the disk to memory and to store data from memory to disk respectively. Real machines usually implement transfer using a DMA (Direct Memory Access) controller which transfers data directly between disk and memory and signals the processor after the transfer is complete. This happens without intervention of the processor. However XSM, provides machine instructions to do this. It is a deviation from real machines and has been provided to avoid the complexity associated with device management by the operating system when more than one process is running. We decided to avoid device management and DMA altogether because it is beyond the scope of the project. We feel that such features may be attempted on more advanced platforms like Minix after the completion of this project.

XSM supports virtual memory management. The machine will transfer control to a specified location in memory where the exception handler is expected to reside upon encountering an exception after setting an EFR or the Exception Flag Register. Unlike MIPS machines[3], which has 3 registers for exception handling, XSM combines all three registers into a single register for simplicity. Exceptions in XSM include a page fault, illegal instructions or operands, illegal memory access and arithmetic exceptions.

XSM machine has capability of running an operating system capable of multiprogramming, file management and virtual memory on top of it. XOS has been designed to exploit the complete capabilities of the XSM architecture, keeping in mind a simplistic design. The motivation behind the design of all components was sequentially building the concepts of operating systems and not on the intricate details associated with its implementation.

4. EXPERIMENTAL OPERATING SYSTEM

The specification for an Experimental Operating System or XOS is laid out to the students. In this project, students will build XOS to meet the specification. XOS is a simulated operating system which runs on top of XSM which is a simulated machine hardware. The OS kernel unlike Nachos [2] resides in the memory of the machine during runtime, and is not fused together with the simulated machine and compiled as a single program. The simulated disk formatted

using XFS, permanently stores the OS routines and data structures like in real systems. The disk is simulated using a UNIX file.

The various components of the operating system include routines like OS startup code, eight interrupt routines including the timer interrupt and the exception handler routine, and data structures like ready list of PCBs, per-process page tables, the system wide-open file table, memory free list, memory copy of disk data structures including FAT and the disk free list. The OS routines are to be programmed by the student building XOS and is loaded to the disk. The OS startup code which the student programs is responsible for loading all the other routines. The OS startup code which must be loaded in the 1st block of the XFS disk. When the system boots the bootstrap loader or the ROM code will copy this disk block to memory and pass control to it. The ROM code is assumed to be hardcoded in the machine memory, and need

The OS routines will modify the memory-resident data structures, and make changes in the disk as and when required. 10 system calls are provided by XOS which include file system calls like Create, Delete, Open, Close, Read, Write, Seek and process system calls like Fork, Exec and Exit. These 10 system calls are mapped to 7 interrupt routines provided by the machine. The scheduler is present in the timer interrupt routine. The page fault handler is present in the exception handler routine of the machine. For all other exceptions XOS will exit the process.

5. DEVELOPMENT TOOLS

5.1 Application Programmer's Language (APL)

5.2 System Programmer's Language (SPL)

5.3 XSM Debugger

5.4 XFS Interface

6. ROADMAP

7. CONCLUSIONS

8. ACKNOWLEDGMENTS

This section is optional; it is a location for you to acknowledge grants, funding, editing assistance and what have you. In the present case, for example, the authors would like to thank Gerald Murray of ACM for his help in codifying this *Author's Guide* and the `.cls` and `.tex` files that it describes.

9. REFERENCES

- [1] C. L. Anderson and M. Nguyen. A survey of contemporary instructional operating systems for use in undergraduate courses. *Journal of Computing Sciences in Colleges*, 21(1):183–190, 2005.
- [2] W. A. Christopher, S. J. Procter, and T. E. Anderson. The nachos instructional operating system. In *Proceedings of the USENIX Winter 1993 Conference*
- [3] J. Heinrich. *MIPS R4000 Microprocessor User's Manual*. MIPS technologies, 1994.
- [4] D. A. Holland, A. T. Lim, and M. I. Seltzer. A new instructional operating system. *ACM SIGCSE Bulletin*, 34(1):111–115, 2002.
- [5] B. Pfaff, A. Romano, and G. Back. The pintos instructional operating system kernel. In *ACM SIGCSE Bulletin*, volume 41, pages 453–457. ACM, 2009.
- [6] A. Silberschatz, P. B. Galvin, and G. Gagne. *Operating system concepts*. J. Wiley & Sons, 2009.