AN EXPERIMENTAL OPERATING SYSTEM

An operating system project for junior year undergraduate computer science students.

Dr. K. Muralikrishnan
kmurali@nitc.ac.in

Sreeraj S
sreeraj.altair@gmail.com

Shamil C M
shamil.cm@gmail.com

Vivek Anand T Kallampally
vivekzhere@gmail.com

http://xosnitc.github.com

Department of Computer Science and Engineering
National Institute of Technology Calicut

What is XOS?

- A platform for students to build a simple operating system.
- XOS is to be built from scratch by the students according to the specification provided.
- Runs on a simulated machine hardware called XSM (Experimental String Machine).
- The hard disk is simulated and uses a file system called XFS (Experimental File System)
- Package includes machine simulator, system and application programming language compilers, file system interface, specification of components and a roadmap.

Why XOS?

- **Simple and Easy-to-understand:** Only essential and important features for giving an elementary understanding and practical feel.
- **Conceptual Clarity:** The operating system kernel completely resides in the machine memory, unlike Nachos (Christopher, Wayne A., Steven J. Procter, and Thomas E. Anderson. "The Nachos instructional operating system." Proceedings of the USENIX Winter 1993 Conference Proceedings on USENIX Winter 1993 Conference Proceedings. USENIX Association, 1993.)
- **Complete Package:** A complete simulation environment, interfaces and programming language compilers are provided.
- **Well Documented:** Every component is documented in detail and an implementation roadmap is provided.

What is not there?

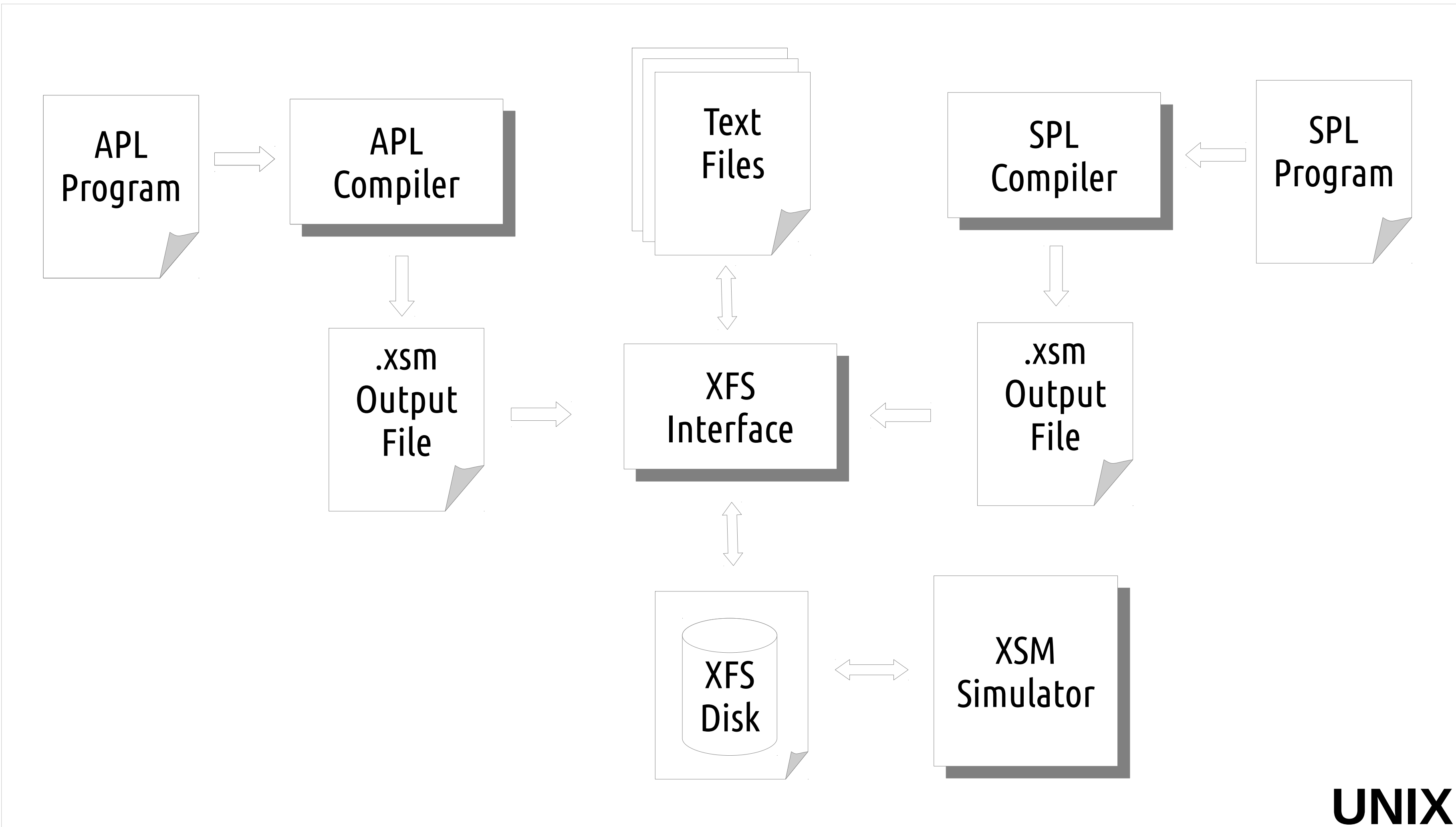
- Device Management or asynchronous devices
- Interprocess Communication
- File Caching and File Permissions
- Limited support of process synchronization has been suggested as enhancements to XOS. This includes process sytem calls like Wait, Signal, Getpid and Getppid.

Department of Computer Science and Engineering
National Institute of Technology Calicut

Components

- Experimental String Machine (XSM) Simulator
- Experimental File System (XFS) Interface
- System Programmer's Language (SPL) Cross Compiler
- Application Programmer's Language (APL) Cross Compiler
- Experimental Operating System Specification and Roadmap

How they interact?



Experimental String Machine (XSM)

- XOS runs on a simulated machine hardware called XSM.
- Smallest addressable unit is a word which is string of 16 characters.
- Native 2-address instruction set architecture.
- Components include timer, registers, memory and disk.

Experimental String Machine (XSM)

- **Timer** in XFS occurs in fixed number of instructions.
- **Registers** in XSM include
  - Program Registers (R0 – R7) to be used by application programs
  - Kernel Registers (S0 – S15) to be used by system routines
  - Temporary Registers (T0 – T4) for compiling system programs
  - Special Purpose Registers, BP, IP, SP, PTBR, PTLR and EFR.
  - IP cannot be modified other than through IRET and jump / branching instructions
- **Memory** of XSM has 64 pages, with page size 512 words.
- **Disk** is simulated using a UNIX file.

Experimental String Machine (XSM)

- Operation is simplified by using **LOAD** and **STORE** instructions and not using a DMA controller.
- Machine supports multiprogramming and virtual memory.
- Exceptions occur during page fault, invalid instructions or arguments, and invalid operations.
- Upon encountering an exception, control is transferred to Exception Handler Routine.
- Machine has two privilege modes, USER and KERNEL modes.

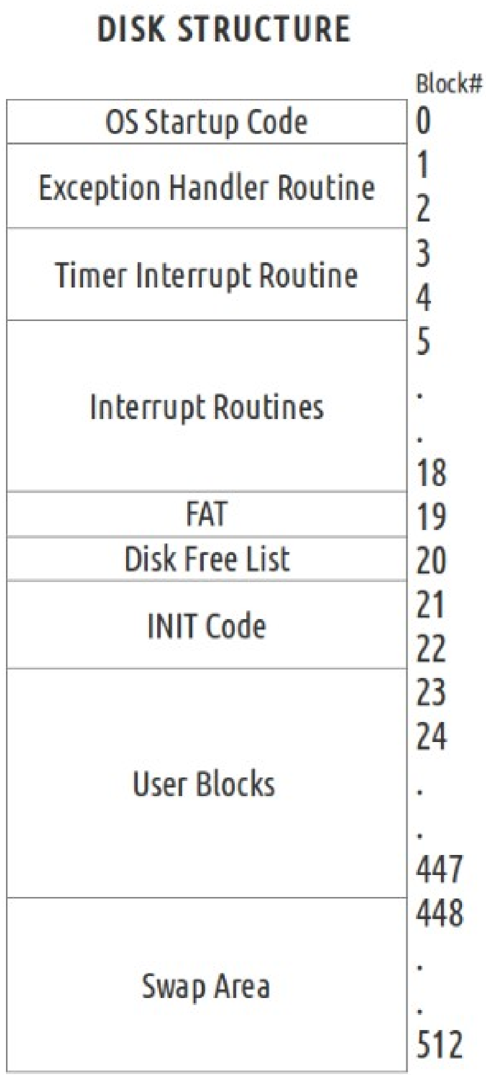
Experimental String Machine (XSM)

- Privilege Modes
  - Mode switching from USER to KERNEL occurs during interrupts and exceptions
  - Mode switching from KERNEL to USER occurs with **IRET** instruction
  - All addresses are physical or directly-mapped in KERNEL mode.
  - All addresses are logical in USER mode. Address translation from logical to physical is done using page tables following a basic and simple address translation scheme.
  - Privileged instructions can be invoked only in KERNEL mode.
  - Interrupts are disabled in KERNEL Mode. Machine halts on exceptions in KERNEL mode.

XSM Simulator

- A C program simulates XSM
- The interval of timer can be varied in the simulator
- The simulator can be run in the **DEBUG** mode which invokes a GDB-like debugger
  - Pauses execution upon encountering a **BRKP** instruction
  - Memory and register contents can be viewed.
  - A register value can be watched whenever changes are made
  - Instructions can be single stepped or execution can be continued at breakpoints.

Experimental File System (XFS)

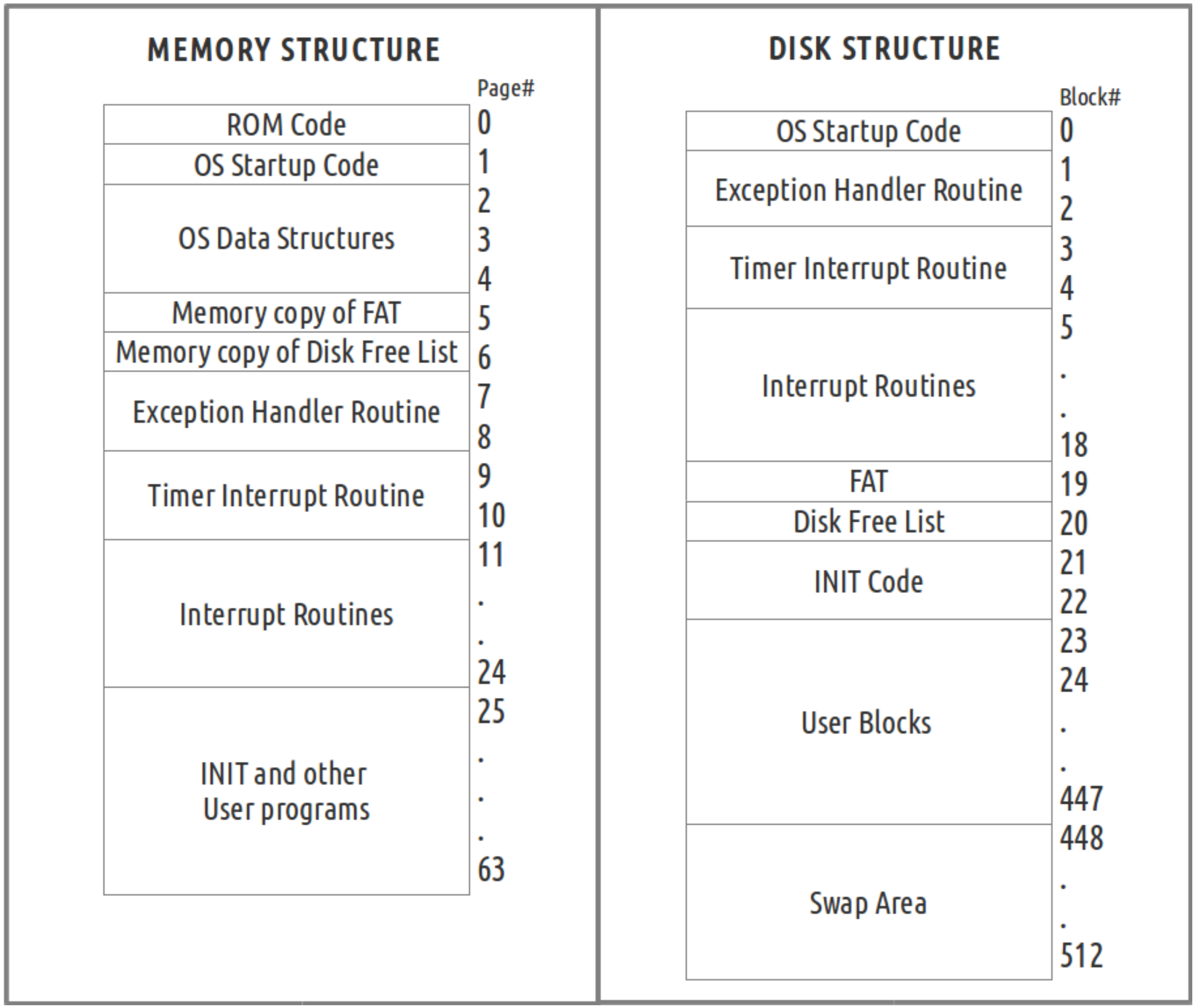


- XFS is a simple filesystem with no directory structure.
- Disk size is 512 blocks, with block size same as the page size in XSM (512 words).
- Every file contains a basic block and a data block.
- Two kinds of files, Executable and Data Files
- INIT Code has no FAT entry

XFS Interface

- XFS disk is simulated using a UNIX file, **disk.xfs**
- XFS interface is provided as part of the development tools
- XFS interface can be used to
  - Create and Format the XFS disk
  - Move files from UNIX machine to the XFS disk.
  - Copy range of blocks from XFS disk and put it in a UNIX file.
  - Commands like **ls** and **cat** are provided to display the list of files and the content of a particular file within the XFS disk.

Experimental Operating System (XOS)



http://xosnitc.github.com

Department of Computer Science and Engineering
National Institute of Technology Calicut

Experimental Operating System (XOS)

- The various functionalities to be implemented in XOS are process management, file management and memory management.
- The functionalities are implemented as system calls, scheduler and page fault handler.
- XOS is capable of multiprogramming and demand paging.
- In this project, XOS routines like OS Startup Code, the various interrupt routines including timer interrupt routine and the exception handler routine is to be programmed.

XOS Data Structures

- **Per-process page tables**: used for address translation for each process in memory.
- **Memory Free List**: Indicates if a memory page is used or not.
- **System Wide Open File Table**: list of files opened by processes
- **Ready List of PCBs**: list of task structures (PCBs) of processes in memory
- **PCB** of a process contains its Process Identifier (PID), STATE information, register values, details about files opened by the process.
- **File Allocation Table**: Disk data structure storing details of files on the disk. A memory copy is maintained by XOS.
- **Disk Free List:** Disk data structure that indicates if a disk block is used or not. A memory copy is maintained by XOS.

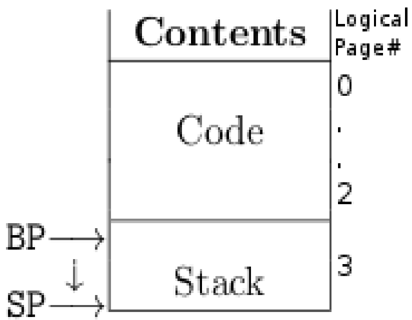
Process Structure

Code

- These are pages of the memory that contain the executable code loaded from the disk.

Stack

- This is the user stack used for program execution.
- The variables and data used during execution of program is stored in the stack.
- It grows in the direction of increasing word address.
- The location of the stack is fixed at the 4th page of the process.



OS Startup Code

- The OS Startup Code should be programmed to load and initialize data structures like Page Tables, Memory Free List and memory copy of Disk Free List.
- The OS Startup Code should also load the interrupt routines and exception handler routine from disk to the memory.
- It must load disk data structures like FAT and Disk Free List from disk to memory.
- It must setup the PCB and Page Tables of the INIT program, load it from disk to memory and start its execution.

Department of Computer Science and Engineering
National Institute of Technology Calicut

Timer Interrupt Routine

- Timer interrupt routine is invoked when a timer interrupt occurs at specific intervals of instructions
- The scheduler is to be implemented within the timer interrupt routine.
- The scheduler of XOS follows a *round-robin* scheduling technique.

Interrupt Routines 1 - 7

- Implementation of various system call interfaces in XOS is to be done within software interrupt routines 1 – 7.
- These are invoked using INT instructions.
- The system calls to be handled by XOS include
  - File System Calls like Create, Delete, Open, Close, Read, Write, Seek
  - Process system calls like Fork, Exec and Exit
  - System calls like Wait, Signal, Getpid and Getppid are suggested as enhancements.

Department of Computer Science and Engineering
National Institute of Technology Calicut

Exception Handler Routine

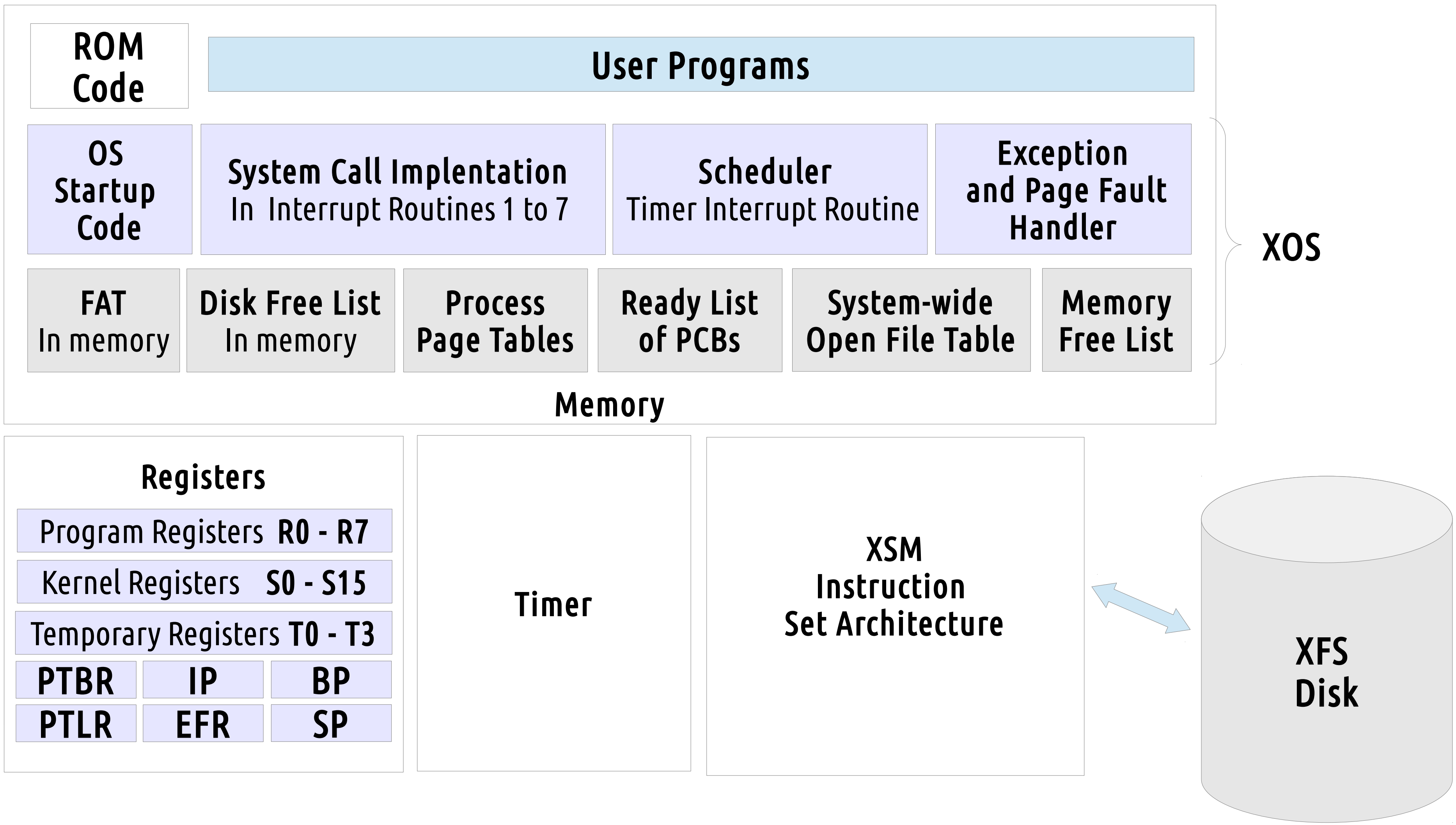
- The Exception handler routine must be programmed to exit the process on all exceptions except the page fault exception
- Page Fault exception occurs when translation is attempted on an invalid logical page.
- Upon a page fault exception, if its an accessible page, the page must be loaded from disk to memory.

Department of Computer Science and Engineering
National Institute of Technology Calicut

Virtual Memory and Demand Paging

- If memory for loading a page from disk to memory is not available, page replacement must be done to make space.
- A page is swapped from the memory to the swap area of the disk and the required page is loaded.
- For simplicity, XOS avoids swapping out the stack page and code pages shared by more than one process.
- Page replacement is done using reference bits, by following the *Second Chance Algorithm*.

Structure



Department of Computer Science and Engineering
National Institute of Technology Calicut

System Programmer's Langauge (SPL)

- The System Programmer's Language is a programming language to write the operating system routines inorder to build XOS.
- It is very closesly related to XSM instruction set, and has statements to access machine registers and memory directly.
- It is provided for convenience of the system programmer instead of directly using the machine instructions.
- Registers can be aliased using meaningful identifiers.
- Allows defining constants and has predefined constants for XOS.
- SPL cross compiler is provided as part of development tools to generate .xsm output files. These files are loaded as system routines to the XFS disk.

Application Programmer's Langauge (APL)

- The Application Programmer's Language is a high level language used to write user programs to be run on top of XOS.
- It supports both string and integer data types.
- It has system call interfaces corresponding to the various system calls available in XOS.
- APL cross compiler is provided as part of development tools
- Programs written in APL is compiled using the cross-compiler to XSM machine instructions.

ROADMAP

- The roadmap helps to build XOS from scratch sequentially by carefully understanding every concept.
- The roadmap is divided into stages including a stage to implement a few enhancements on XOS.
- The roadmap contains theory points and explanations at every stage of implementation including links to other documents.
- Its a source of obtaining elementary understanding to operating system concepts, by parallely building the operating system.

Department of Computer Science and Engineering
National Institute of Technology Calicut

ROADMAP

- STAGE 1: Setting up the system
  - Helps set up and test the development tools.
  - Elementary understanding of the various components of the project.
- STAGE 2: Understanding the filesystem
  - Introduces the XFS filesystem including disk data structures.
  - Teaches how to use the XFS interface.
- STAGE 3: Starting the machine
  - Start the machine in kernel mode
  - Run a kernel program to print odd numbers and run it as OS Startup Code.

ROADMAP

- STAGE 4: Running a user program
  - A user program to print primes is to be written in APL, compiled and loaded to disk.
  - The OS Startup code is to be programmed to set up the INIT process and load the newly written program as the INIT program.
  - The OS now runs in a single process mode.
- STAGE 5:Interrupt Routines
  - Introduces the concept of interrupts.
  - A sample implementation of a software interrupt routine and timer interrupt routine is done.
  -

ROADMAP

- STAGE 6: Getting started with multiprogramming
  - OS is made to run two programs concurrently.
  - The scheduler is implemented in the timer interrupt routine to switch between the two programs.
- STAGE 7: Creating Files
  - The first system call 'Create' is to be implemented in the stage.
  - System Calls and corresponding stack operations are explained in detail in this stage.
- STAGE 8: Playing with Files
  - This stage includes the implementation of the remaining file system calls.
  - Test cases are provided for checking if the system calls are working properly.

ROADMAP

- STAGE 9: Process System Calls
  - Process system calls which includes Fork, Exec and Exit are to be implemented in this stage.
- STAGE 10: Exception Handling and Demand Paging
  - Virtual Memory managment is implemented in this stage
  - Page replacement is also done in this stage.
  - The process system calls are modified to incorporate demand paging and page replacement.
- STAGE 11: Enhancements to XOS
  - This stage includes implementing Wait, Signal, Getpid and Getppid system calls
  - This stage also includes making a shell for XOS

Conclusions

- The project is easier to implement compared to the existing instructional operating systems.
- Simple system which is buildable from first principles
- It will help to
  - better comprehend the textbooks on operating systems
  - get a practical feel of operating systems
  - move on to more complex platforms later.
- Limited set of features to be implemented in a 14 – 16 week semster.
- Not intended to be scaled to implement complex features. Real platforms are suggested for advanced understanding of operating systems.
-

Department of Computer Science and Engineering
National Institute of Technology Calicut

Resources

- Website: http://xosnitc.github.com
- Mailing lists:
  - Users' Mailing List: xos-users@googlegroups.com
  - Developers' Mailing List: xos-developers@googlegroups.com