Lecture 03 Operating System Components

Dr. Tushar, Mosaddek Hossain Kamal
Professor

Computer Science and Engineering, University of Dhaka,

BSc Third Year, Semester 2 (July – Dec), Academic Year: 2024

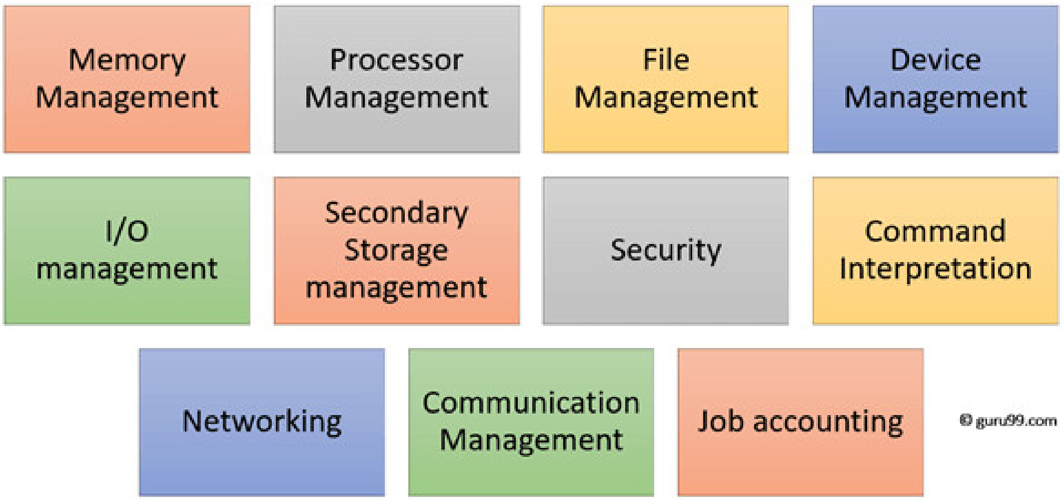
CSE3201: Operating Systems

October 08, 2024

Outline

Learning Outcome

- Understand the basic components of an Operating System
  - Process, Memory, File System, System Calls, Operating System Structure, Resource management
- Understand the concepts of Operating System components and their interactions



Operating System Components

Process

- A program in execution
- An instance of a program running on a computer
- The entity that can be assigned to and executed on a processor
- A unit of resource ownership
- A unit of activity characterized by a single sequential thread of execution, a current state, and an associated set of system resources
    - Nowadays the execution abstraction is separated out: Thread
    - Single process can contain many threads

Process

Consist of three segments

- Text
  - contains the code (instructions)
- Data
  - Global variables
- Stack
  - Activation records of procedure
  - Local variables

Note

- data can dynamically grow up
- The stack can dynamically grow down

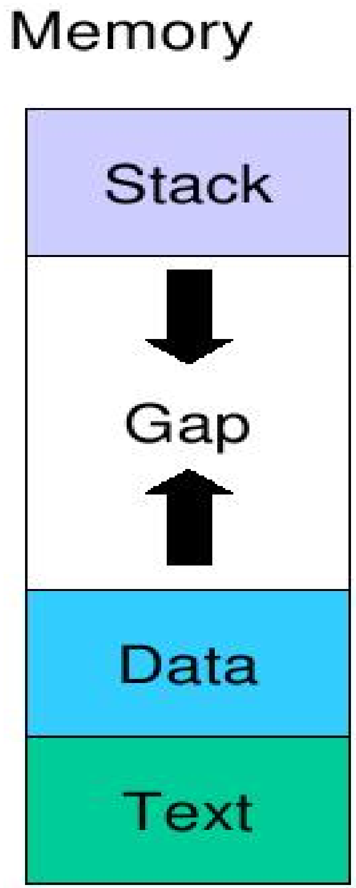
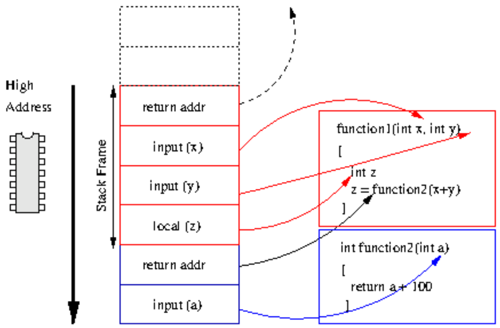


Figure 1: Process Memory

Process Consists of three components

- An executable program
  - Text
- Associated data needed by the program
  - Data and stack
- Execution context of the program
  - All information the operating system needs to manage the process
    - Registers, program counter, stack pointer, etc...
  - A multithread program has a stack and execution context for each thread



Multiprocess Creates Concurrency issues

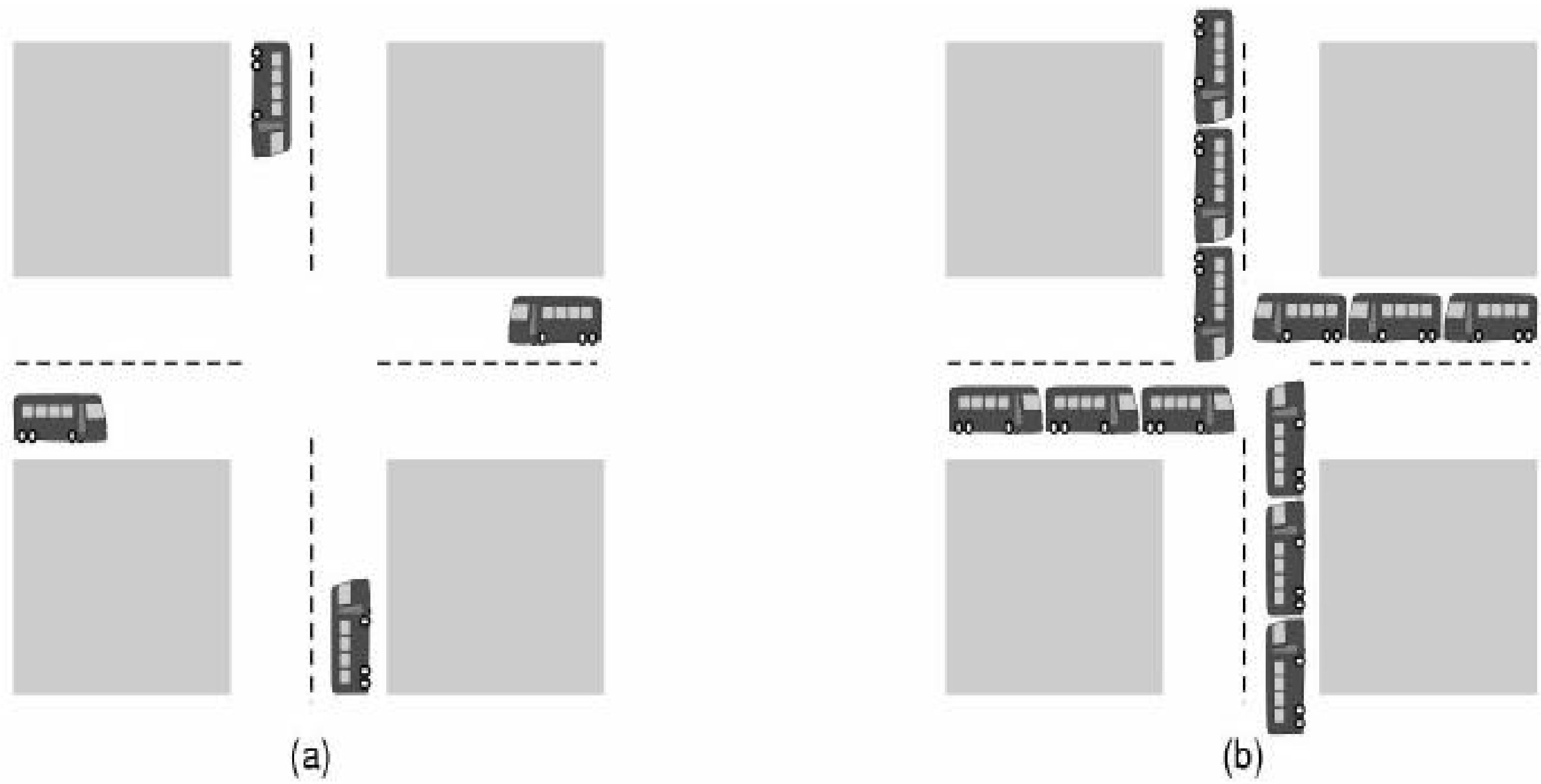
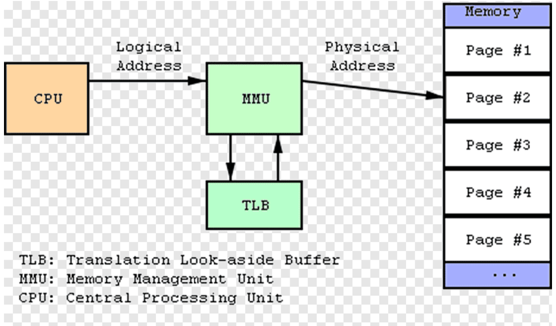


Figure 2: (a) Potential Deadlock, (b) Actual Deadlock

The view from thirty thousand feet

- Process isolation
  - Prevent processes from accessing each others data
- Automatic allocation and management
  - Don't want users to deal with physical memory directly
- Support for modular programming
- Protection and access control
  - Still want controlled sharing
- Long-term storage
- OS services
  - Virtual memory
  - File system



Virtual Memory

- Allows programmers to address memory from a logical point of view
  - Gives apps the illusion of having RAM to themselves
  - Logical addresses are independent of other processes
  - Provides isolation of processes from each other

  Can overlap execution of one process while swapping in/out others.

Virtual Memory Addressing

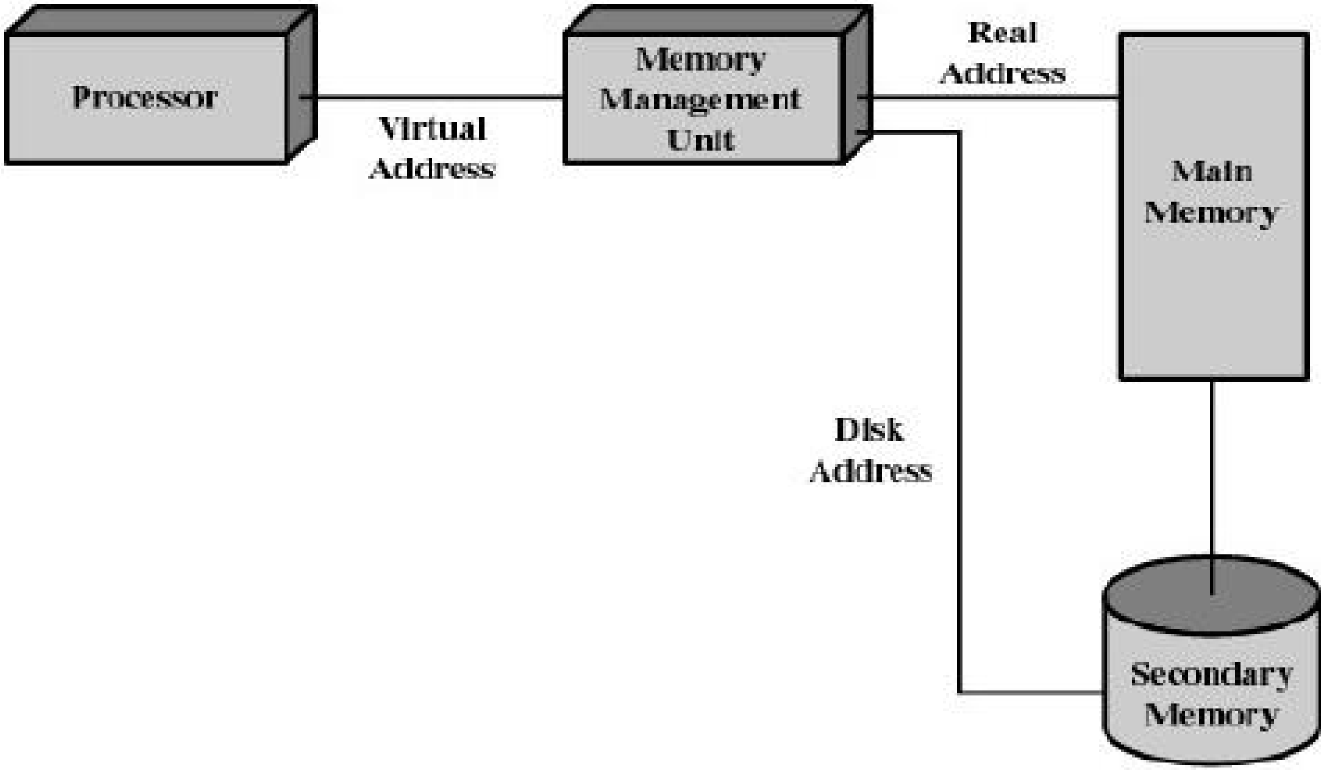
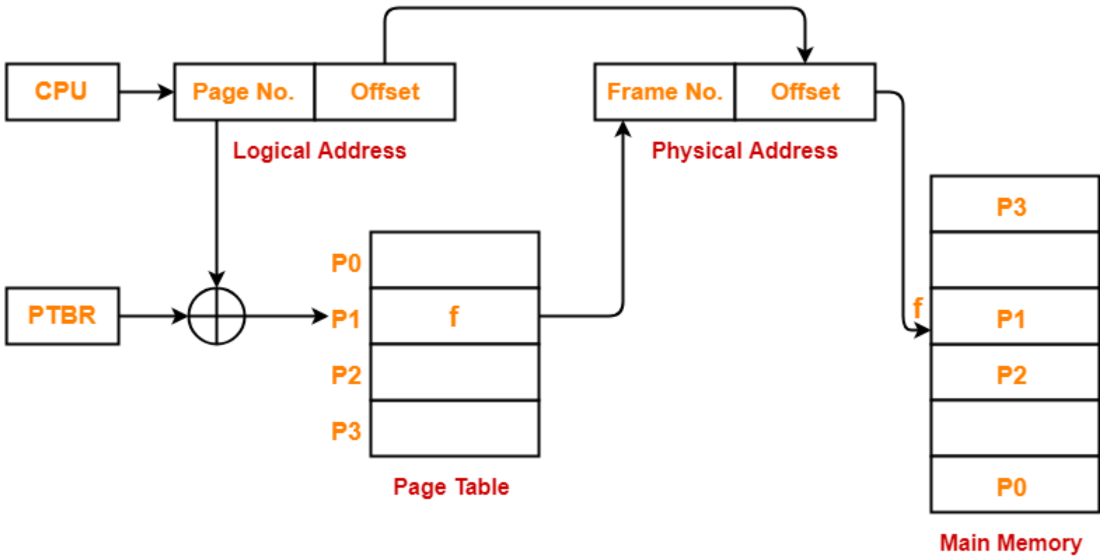


Figure 3: Virtual Memory Addressing

- Allows process to be comprised of a number of fixed-size blocks, called pages
- Virtual address is a page number and an offset within the page
- Each page may be located any where in main memory
- A page may actually exist only on disk



Translating Logical Address into Physical Address

Virtual Memory

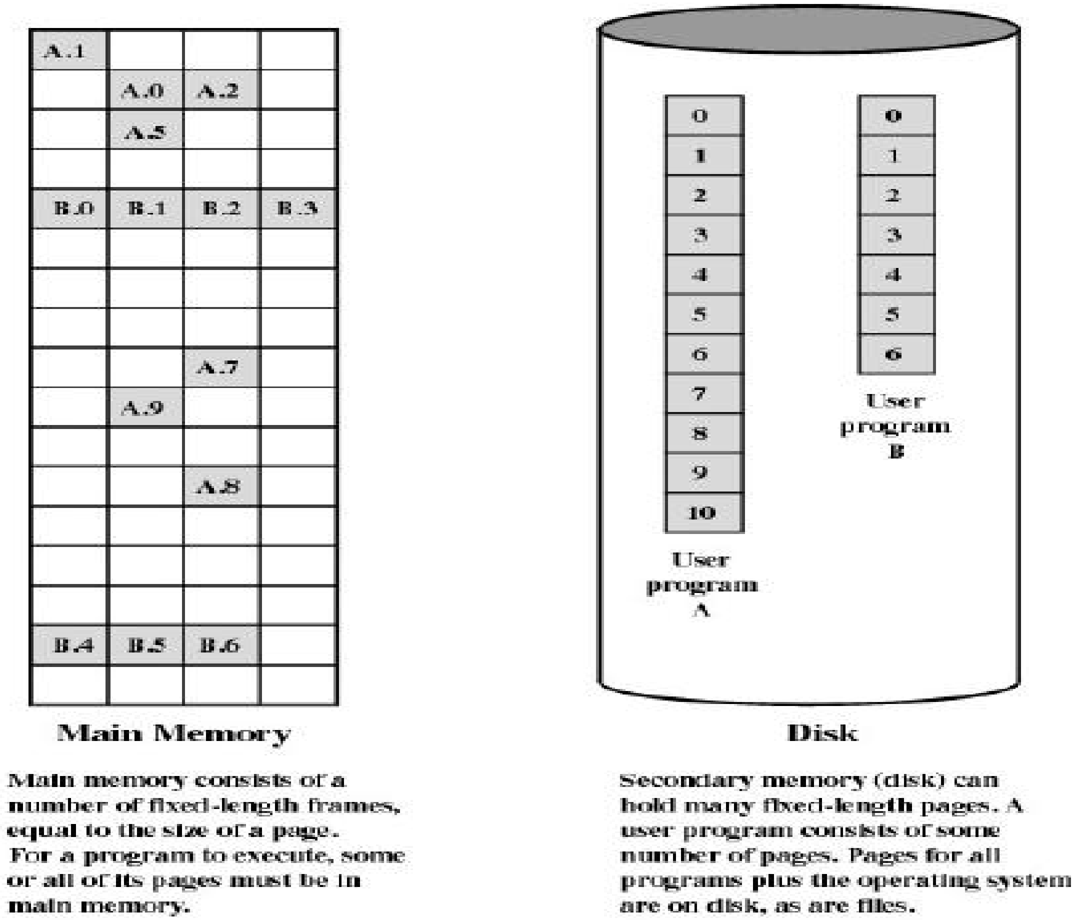


Figure 4: Virtual Memory Concept

File System

- Implements long-term store
- Information stored in named objects called files

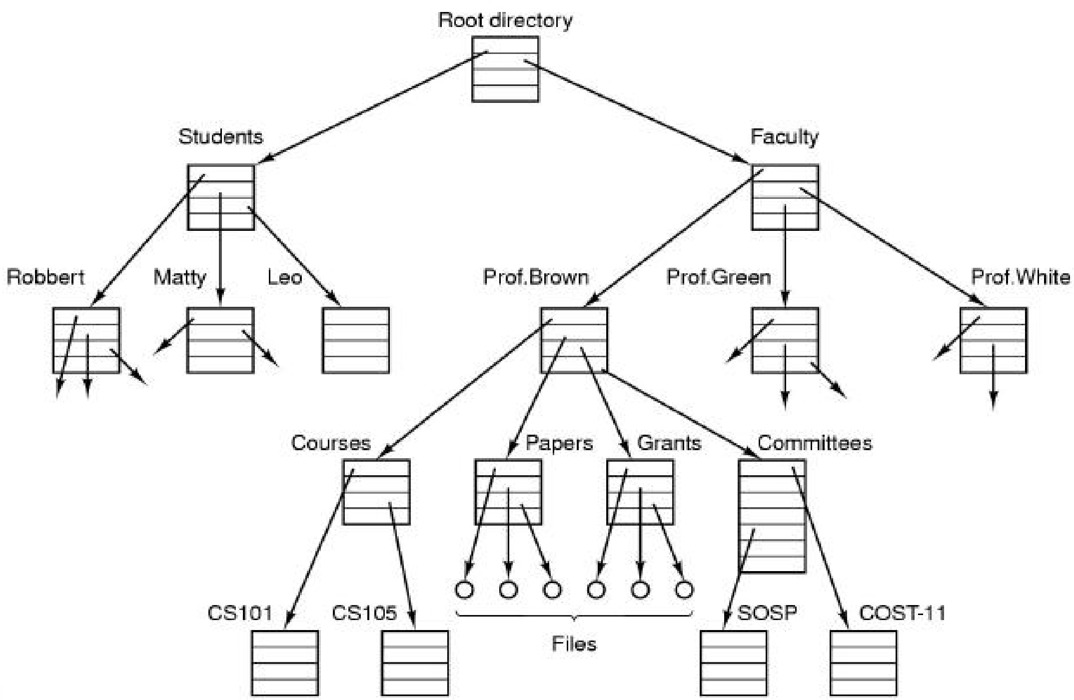


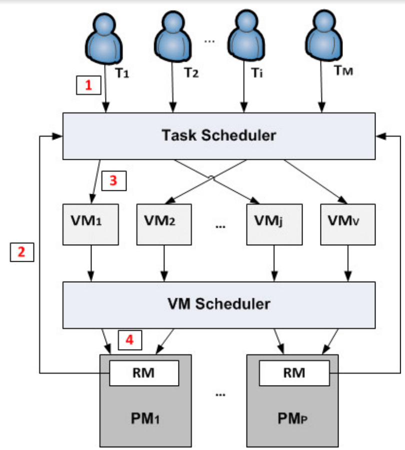
Figure 5: File System Example

Information Protection & Security

- Access control
  - regulate user access to the system
  - Involves authentication
- Information flow control
  - regulate flow of data within the system and its delivery to users

Scheduling & Resource Management

- Fairness
  - give equal and fair access to all processes
- Differential responsiveness
  - discriminate between different classes of jobs
- Efficiency
  - maximize throughput, minimize response time, and accommodate as many uses as possible



Operating System Structure

The layered approach

a. Processor allocation and multiprogramming
b. Memory Management
c. Devices
d. File system
e. Users

Each layer depends on the the inner layers

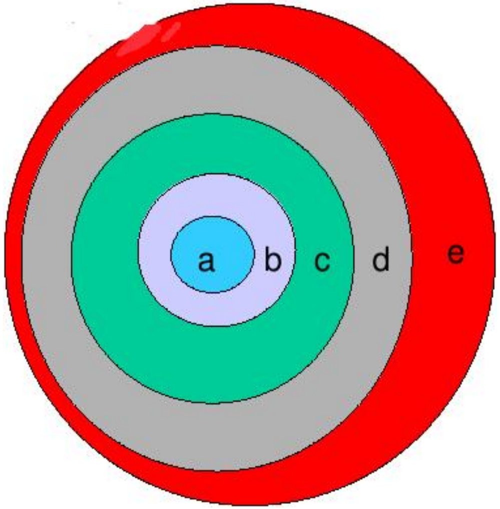


Figure 6

In practice, layering is only a guide

- Operating Systems have many inter-dependencies
  - Scheduling on virtual memory
  - Virtual memory on I/O to disk
  - VM on files (page to file)
  - Files on VM (memory mapped files)
  - And many more…

Monolithic Kernel

- Also called the "spaghetti nest" approach
- Everything is tangled up with everything else.
- Implement user services and kernel services separately
  - How use same address space
  - Make operating system faster
    - Switching between user mode to kernel mode
- Linux, Windows, ...

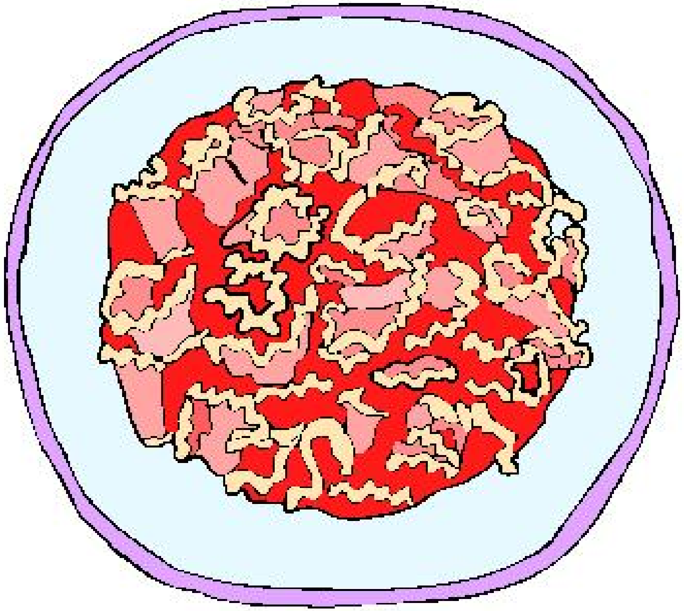


Figure 7: Spaghetti

The Monolithic Operating System Structure

- However, some reasonable structure usually prevails

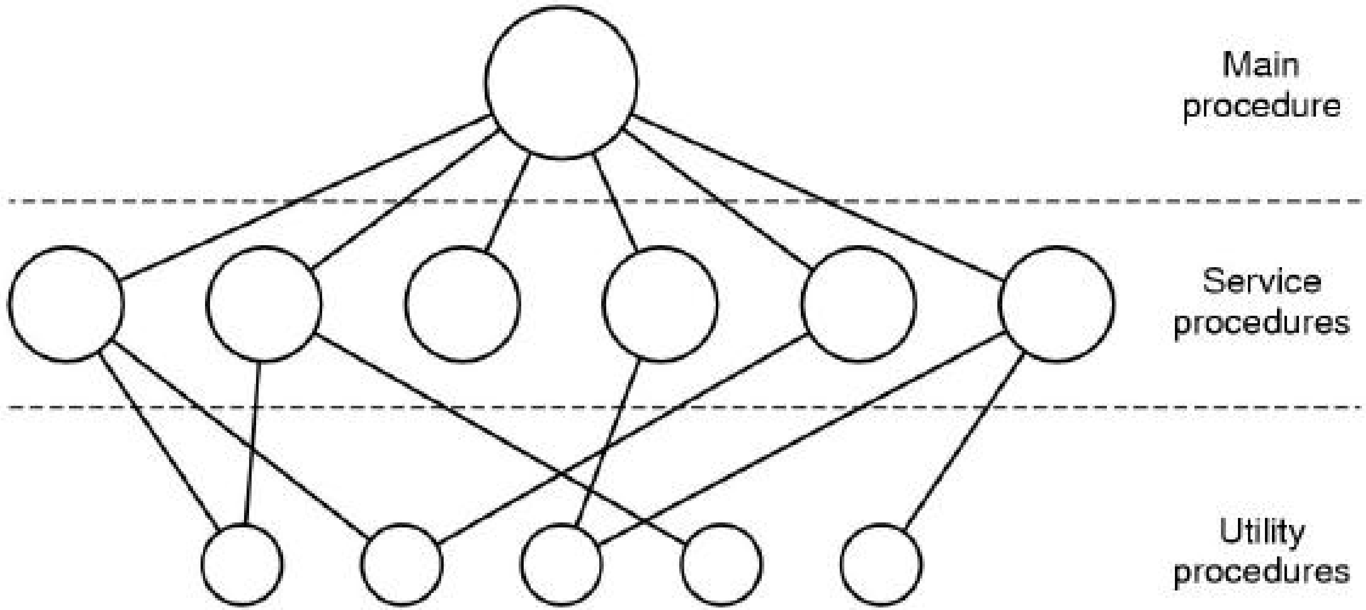
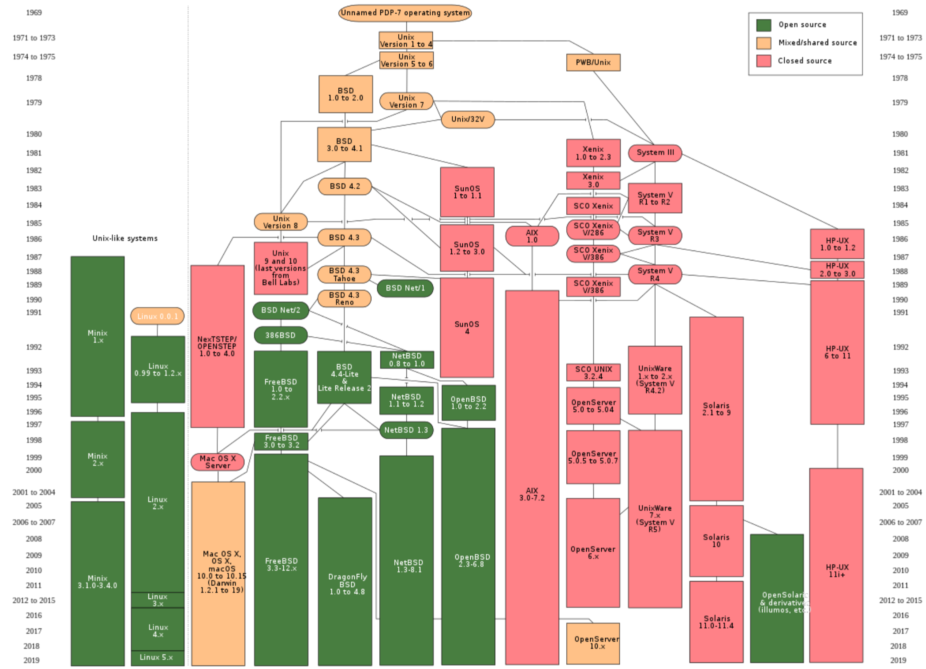


Figure 8: Monolithic Kernel Structure

Monolithic Kernel Example: UNIX

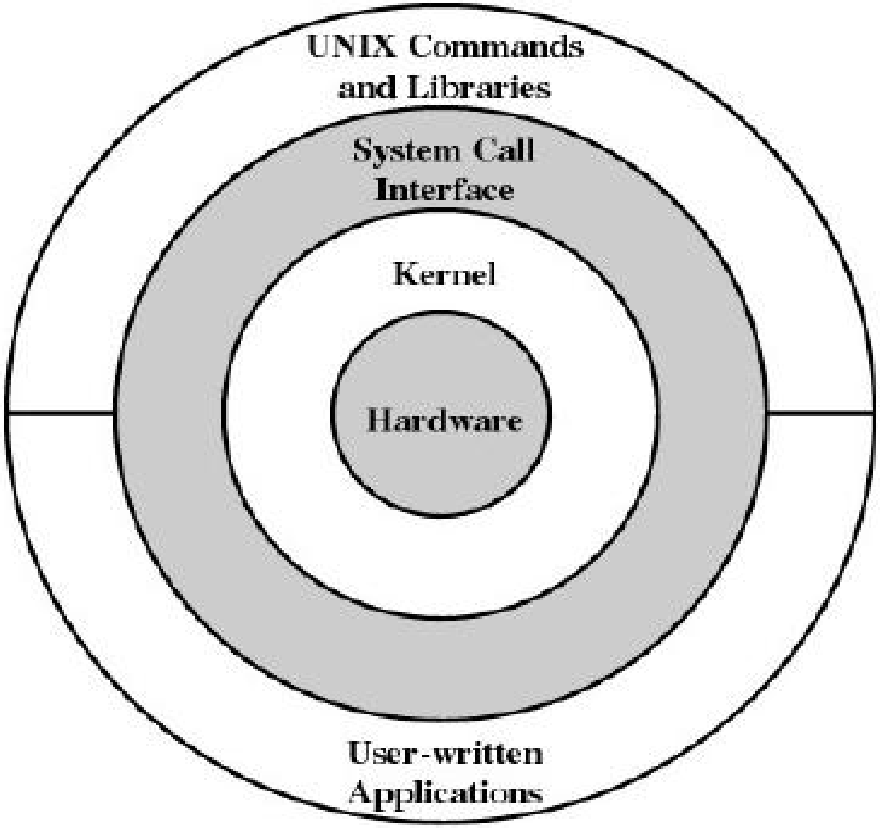
- Provides a good hardware abstraction
  - Everything is a file (mostly)
- Runs on most hardware
- Comes with a number of user services and interfaces
  - Shell
  - C compiler

Note: Unix is the OS that creates or motivates all modern OS (including Mobile OS, Android, and Linux)

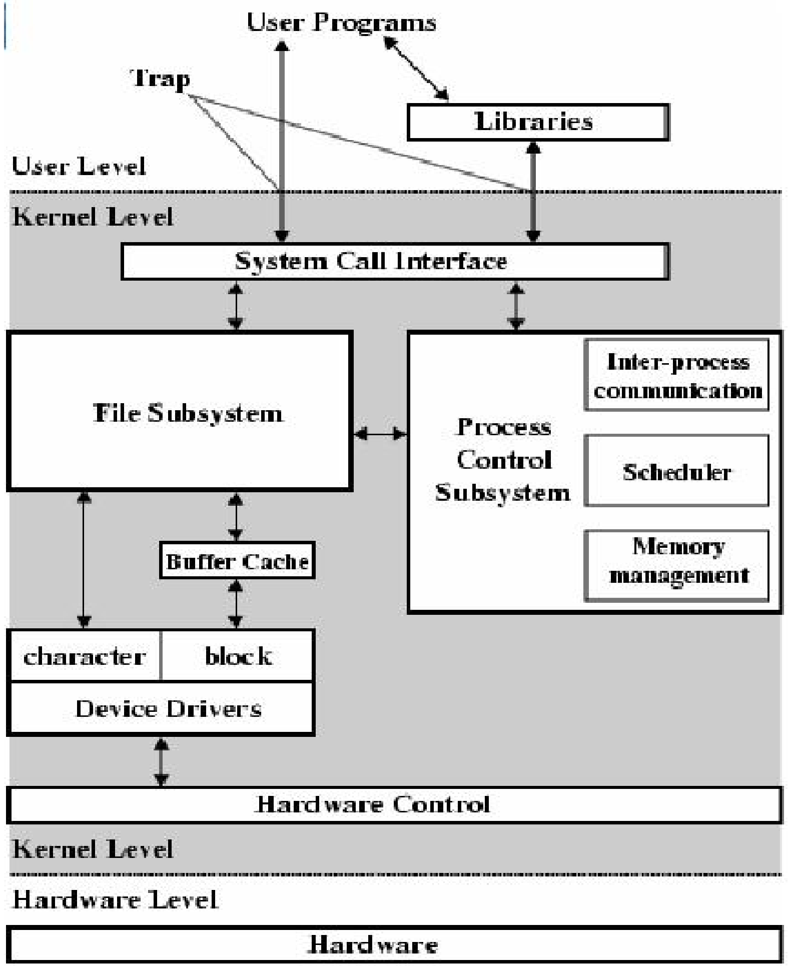


Traditional Unix Structure

Traditional Unix Structure



Traditional Unix Structure



Microkernel-Based System

Microkernel-Based System

- Assigns only a few essential functions to the kernel
  - Address space
  - Interprocess Communication (IPC)
  - Basic scheduling
  - Minimal hardware abstraction
- Other services implemented by user-level servers
- Traditional "system calls" become IPC requests to servers
- Extreme view of a micro-kernel
  - A feature is only allowed in the kernel if required for security
- expensive and poor in performance

Example: Minix – Andrew S. Tanenbaum and Linus Torvald – debate (Linux kernel structure???)
L4 Micro kernel – Unix like OS – L4Ka::Hazelnut, L4/Fiasco, L4Ka::Pistachio
Hyper-vision – Nano-Kernel

Monolithic OS structure

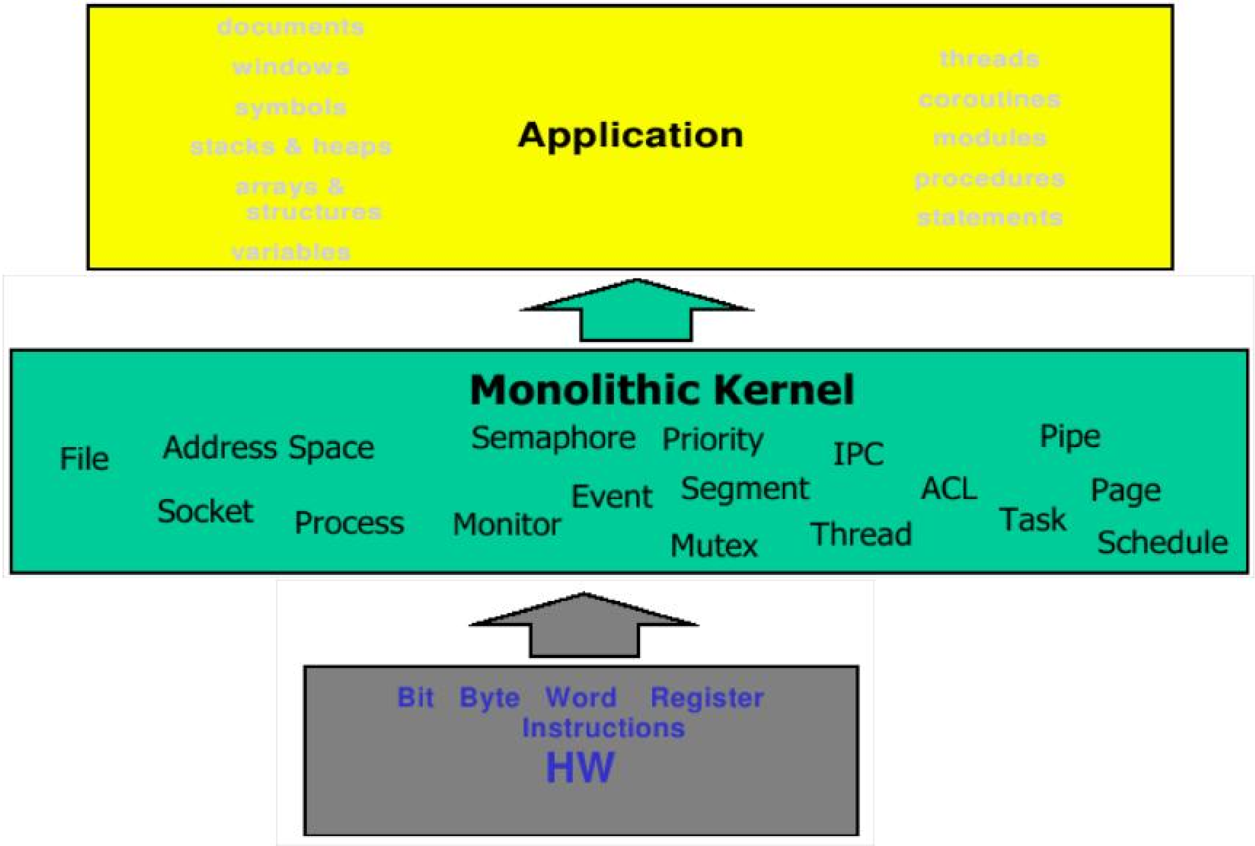


Figure 9: Monolithic Kernel

Microkernel OS structure

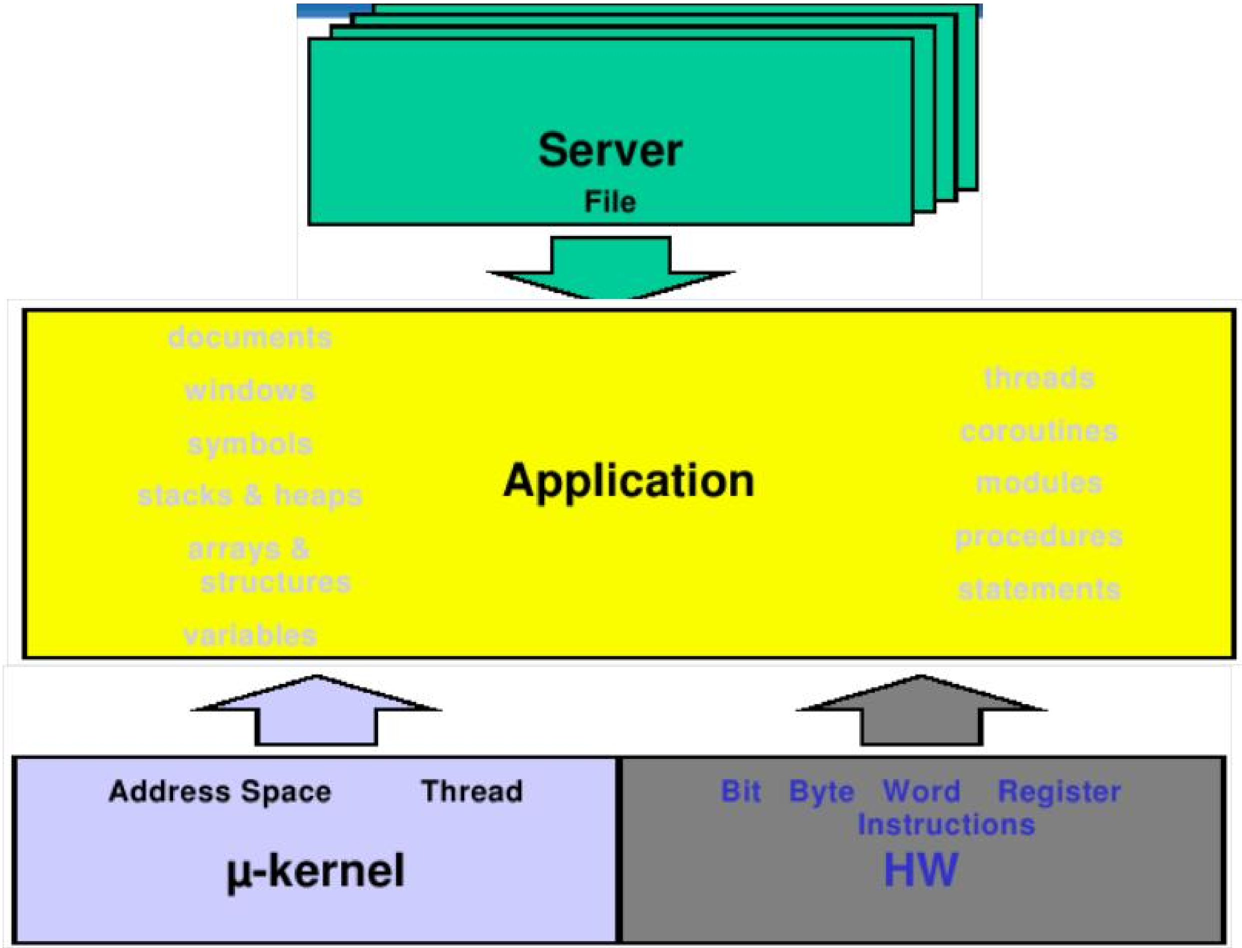


Figure 10: Micro-Kernel OS

Part2: System Calls

- A high-level view of System Calls
  - Mostly from the user's perspective – From textbook (section 1.6)
- A look at the Cortex-M4
  - A brief overview
  - Mostly focused on exception handling (Alert: URL may change)
    - From 'Book in Reference',
    - Data Sheet,https://www.st.com/resource/en/datasheet/stm32f446re.pdf
    - Reference Manual https://www.st.com/resource/en/reference_manual/dm00135183-stm32f446xx-advanced-arm-based-32-bit-mcus-stmicroelect pdf,
    - Programming Manual https://www.st.com/resource/en/programming_manual/pm0214-stm32-cortexm4-mcus-and-mpus-programming-manual-stmicroelec pdf
  - Allow me to provide "real" examples of theory
- System Call implementation
- Case Study: OS CSE and RTOS system call handling

Operating System: System Calls

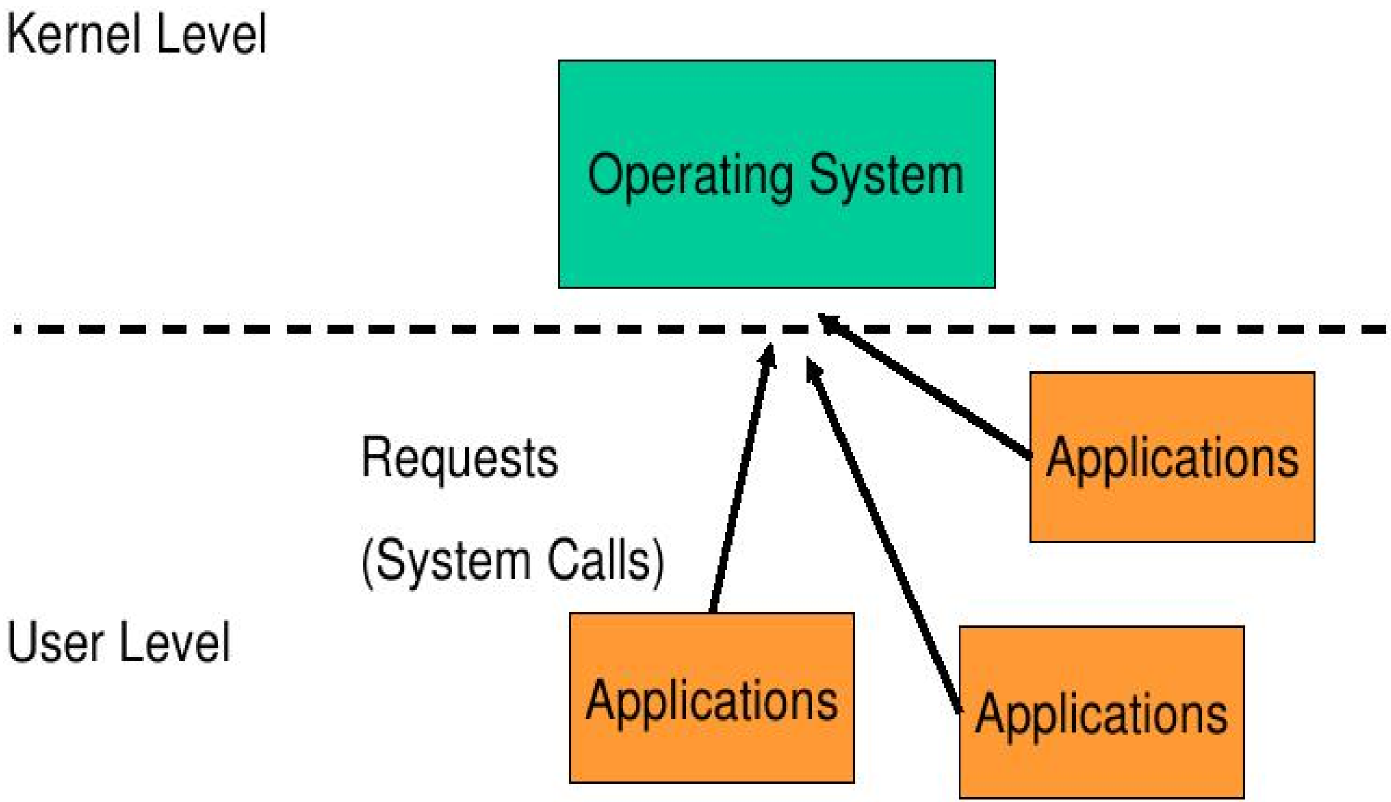


Figure 11: OS Syscalls

System Calls

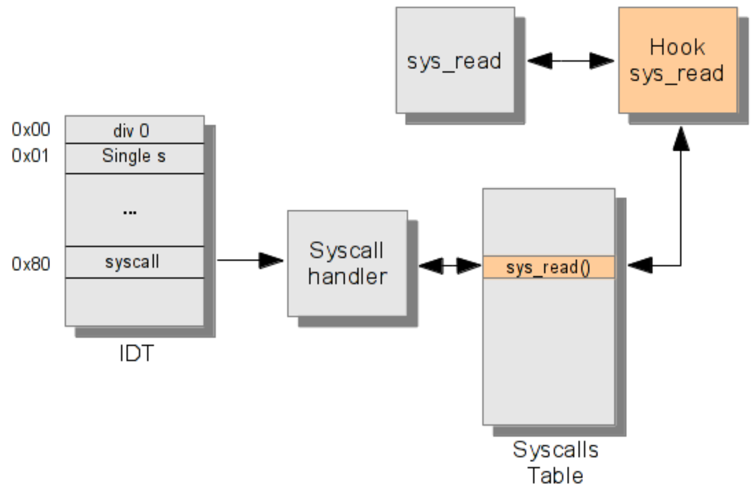
Operating System: System call

- Can be viewed as special procedure calls
  - Provides for a controlled entry into the kernel
  - While in kernel, they perform a privileged operation
  - Returns to original caller with the result
- The system call interface represents the abstract machine provided by the operating system.

A Brief Overview of classes of System Calls

From the user's perspective

- Process Management
- File I/O
- Directories management
- Some other selected Calls
- There are many more
  - On Linux, see man syscalls for a list



Some System Calls For Process Management

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

Figure 12: Process Management Syscalls

Some System Calls For File Management

File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

Figure 13: File management Syscalls

Some System Calls For Directory Management

Directory and file system management	
Call	**Description**
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

Figure 14: Directory Syscalls

Some System Calls For Miscellaneous Tasks

Miscellaneous

Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since Jan. 1, 1970

Figure 15: Misc. Syscalls

- A stripped down shell:

```
while (TRUE) {                              /* repeat forever */
    type_prompt( );                         /* display prompt */
    read_command (command, parameters)      /* input from terminal */

if (fork() != 0) {                          /* fork off child process */
    /* Parent code */
    waitpid( -1, &status, 0);               /* wait for child to exit */
} else {
    /* Child code */
    execve (command, parameters, 0);        /* execute command */
 }
}
```

Figure 16: Syscalls for Shell

System Calls

UNIX	Win32	Description
fork	CreateProcess	Create a new process
waitpid	WaitForSingleObject	Can wait for a process to exit
execve	(none)	CreateProcess = fork + execve
exit	ExitProcess	Terminate execution
open	CreateFile	Create a file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	Read data from a file
write	WriteFile	Write data to a file
lseek	SetFilePointer	Move the file pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory
rmdir	RemoveDirectory	Remove an empty directory
link	(none)	Win32 does not support links
unlink	DeleteFile	Destroy an existing file
mount	(none)	Win32 does not support mount
umount	(none)	Win32 does not support mount
chdir	SetCurrentDirectory	Change the current working directory
chmod	(none)	Win32 does not support security (although NT does)
kill	(none)	Win32 does not support signals
time	GetLocalTime	Get the current time

Figure 17: Unix/Win32 Syscalls

RTOS and Bare-metal System call

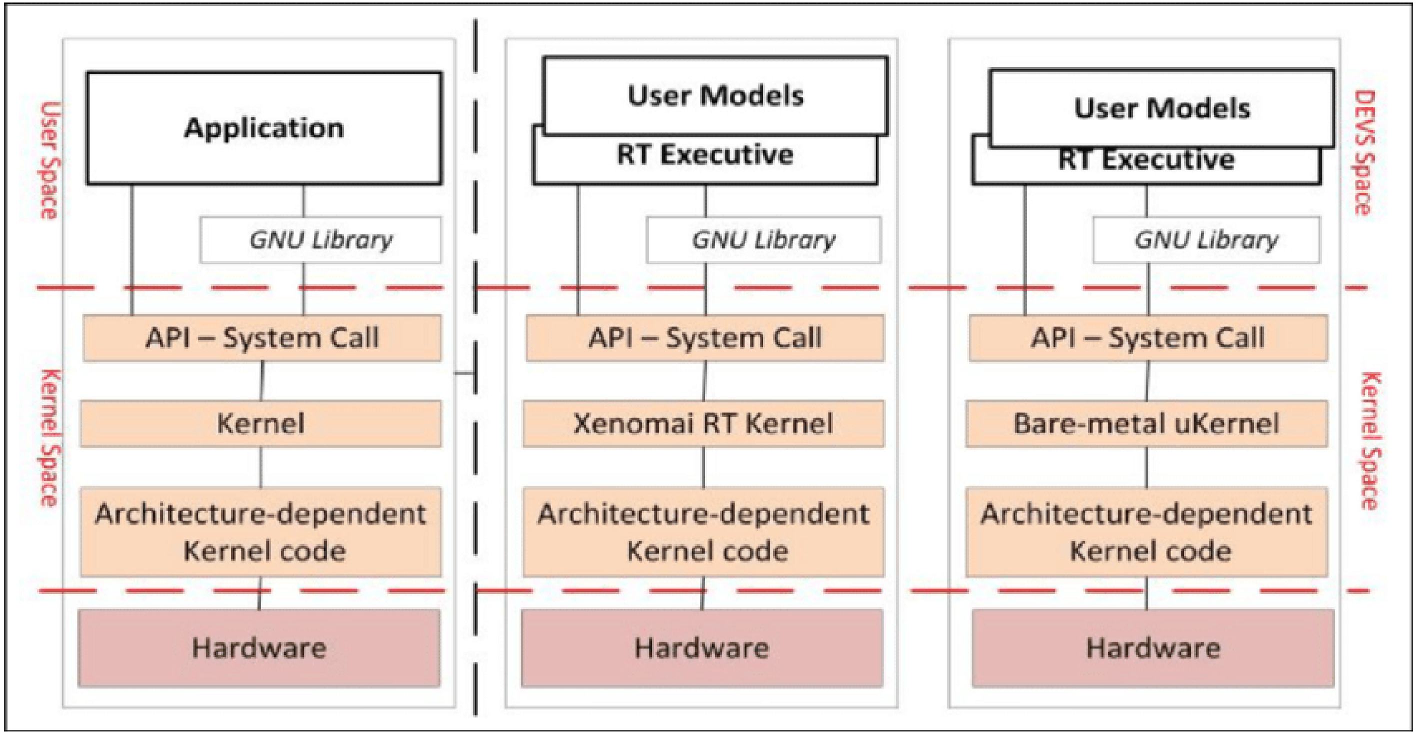


Figure 18: RTOS and Bare-Metal Syscalls

IOS Syscalls

```
kernel:FFFF1F24            starlet_syscall_handler              ; CODE XREF: start
kernel:FFFF1F24 E9 CD 7F FF             STMFA   SP, {R0-LR}^
kernel:FFFF1F28 E1 4F 80 00            MRS     R8, SPSR
kernel:FFFF1F2C E5 8D 80 00            STR     R8, [SP,#spsr_register_save]
kernel:FFFF1F30 E5 8D E0 40            STR     LR, [SP,#lr_register_save]
kernel:FFFF1F34 E5 1E A0 04            LDR     R10, [LR,#-4]   ; R10 = E600XXXX  (the invalid instruction)
kernel:FFFF1F38 E3 CA 9D 7F            BIC     R9, R10, #NOT 0xFFFFE03F
kernel:FFFF1F3C E5 9F 84 CC            LDR     R8, =0xE6000010 ; syscall base
kernel:FFFF1F40 E3 C9 90 20            BIC     R9, R9, #NOT 0xFFFFFFDF ; R9 = R10 & FFFFE01F
kernel:FFFF1F44 E1 59 00 08            CMP     R9, R8          ; Were any bits set other than the syscall number
kernel:FFFF1F48 1A 00 00 1F            BNE     invalid_syscall
kernel:FFFF1F4C E1 A0 A2 CA            MOV     R10, R10,ASR#5  ; R10 = R10 >> 5
kernel:FFFF1F50 E2 0A A0 FF            AND     R10, R10, #0xFF ; R10 = R10 & 0xFF
kernel:FFFF1F54 E3 5A 00 7A            CMP     R10, #0x7A      ; max index of syscall (can vary for each IOS)
kernel:FFFF1F58 CA 00 00 11            BGT     return_to_caller
kernel:FFFF1F5C E1 A0 80 0D            MOV     R8, SP
kernel:FFFF1F60 E3 A0 B0 1F            MOV     R11, #0b11111
kernel:FFFF1F64 E1 21 F0 0B            MSR     CPSR_c, R11     ; switch to system mode, disable irq \& fiq
kernel:FFFF1F68 E5 98 80 44            LDR     R8, [R8,#sp_register_save]
kernel:FFFF1F6C E5 9F B4 A0            LDR     R11, =syscall_stack_arg_counts
kernel:FFFF1F70 E7 9B B1 0A            LDR     R11, [R11,R10,LSL#2] ; number of args on stack for this syscall
kernel:FFFF1F74 E0 8D D1 0B            ADD     SP, SP, R11,LSL#2 ; SP += R11[R10 << 2]
kernel:FFFF1F78            get_stack_arg                        ; CODE XREF: start+1F8j
kernel:FFFF1F78 E3 5B 00 00            CMP     R11, #0
kernel:FFFF1F7C 0A 00 00 03            BEQ     find_syscall_and_jump
kernel:FFFF1F80 E5 3D 90 04            LDR     R9, [SP,#-4]! ; copy argument value
kernel:FFFF1F84 E5 28 90 04            STR     R9, [R8,#-4]!
kernel:FFFF1F88 E2 4B B0 01            SUB     R11, R11, #1
```