CSC-203 Operating Systems[3]

Lecture: 4

Instructor: Sanjog Sigdel

sigdelsanjog.com.np

Date: May 11, 2019

Lecture 4

Historical Background

System Calls:

Definition, Handling System Calls, System calls for Process, File and Directory Management, System Programs, The SHell, Open Source Operating Systems

Historical Background

System Calls

- operating systems have two main functions: **providing abstractions to user programs** and **managing the computer's resources**
- Interface between a process and an operating system is provided by system calls
- Available as assembly language instruction
- System calls are usually made when a process in **user mode requires access to a resources**

Lecture 4

Historical Background

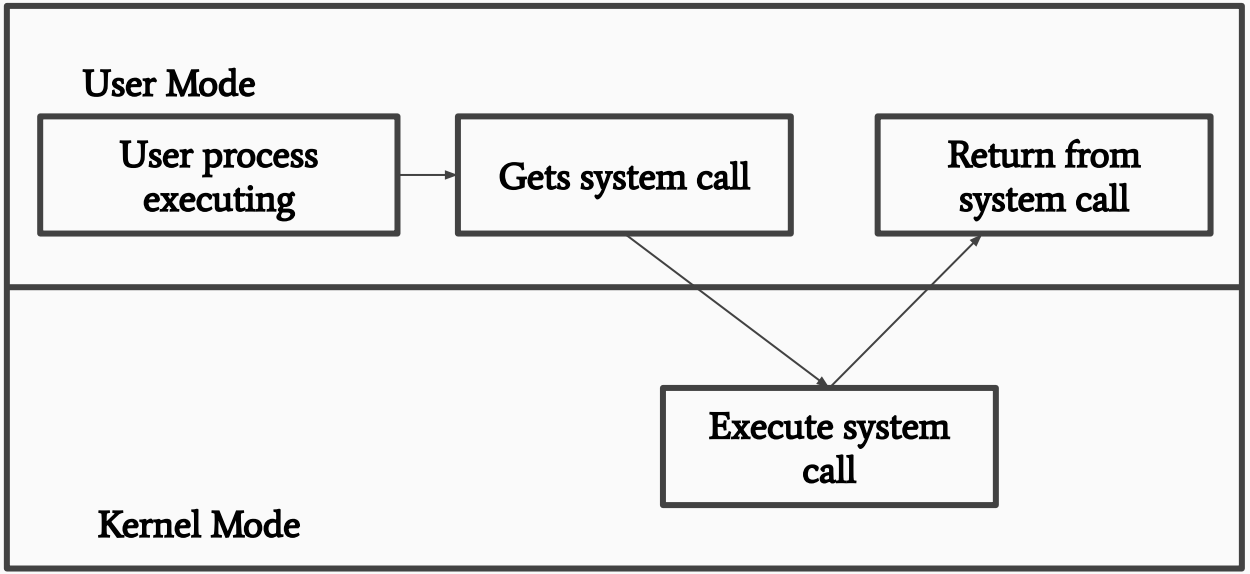


Fig: System Calls

Historical Background

- The system call is executed on a **priority basis** in the Kernel Mode.
- After execution of the system call, the control **returns to the user mode** and **execution of user processes can be resumed**.
- In general system calls are required in the following situations:
  - If a **file system requires** the creation or deletion of files. Reading and writing from files also require a system call
  - **Creation and management** of new processes
  - **Network connections** also require system calls. This includes sending and receiving packets
  - **Access to a hardware device** device such as a printer, scanner etc. requires a system call.

Historical Background

Types of System Calls

There are mainly five types of system calls. These are explained in detail as follows:

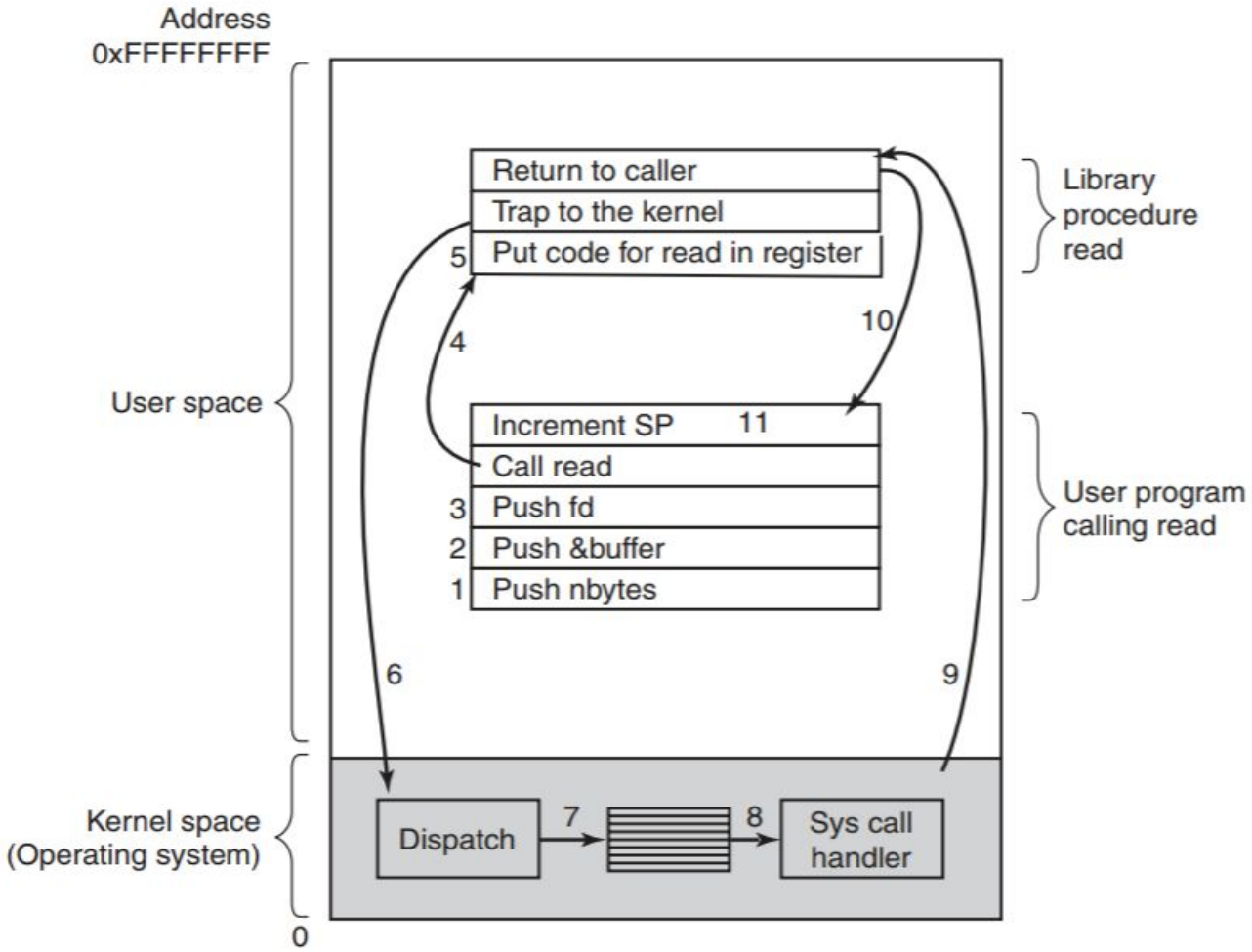
- **Process Control:** These system calls deal with processes such as process creation, process termination etc.
- **File Management:** These system calls are responsible for file manipulation such as creating a file, reading a file, writing into a file etc.
- **Device Management:** These system calls are responsible for device manipulation such as reading from device buffers, writing into device buffers etc.
- **Information Maintenance:** These system calls handle information and its transfer between the operating system and the user program
- **Communication:** These system calls are useful for inter process communication. Thy also deal with creating and deleting a communication connection.

Historical Background

a quick look at the **read system call.**

- it has three parameters:
  - the first one **specifying the file,**
  - the second one **pointing to the buffer,**
  - the third one **giving the number of bytes to read**

```
count = read(fd, buffer, nbytes);
```


Address
0xFFFFFFFF
User space
Kernel space
(Operating system)
0
Return to caller
Trap to the kernel
5
Put code for read in register
4
10
Increment SP
11
Call read
3
Push fd
2
Push &buffer
1
Push nbytes
6
9
Dispatch
7
8
Sys call
handler
Library
procedure
read
User program
calling read

Historical Background

- The calling program first **pushes the parameters onto the stack**, as shown in **steps 1–3**
- Then comes the actual call to the library procedure (**step 4**)
- The library procedure, possibly written in assembly language, typically puts the system-call number in a place where the operating system expects it, such as a register **(step 5)**
- executes a TRAP instruction to **switch from user mode to kernel mode** and start execution at a fixed address within the kernel **(step 6)**
- Kernel code that starts following the TRAP **examines the system-call number and then dispatches to the correct system-call handler**, usually via a table of pointers to system-call handlers indexed on system-call number **(step 7)**
- At that point the system-call handler runs **(step 8)**
- Once it has completed its work, control may be returned to the user-space library procedure at the instruction following the TRAP instruction **(step 9)**
- This procedure then returns to the user program in the usual way procedure calls return **(step 10)**
- To finish the job, the user program has to clean up the stack, as it does after any procedure call **(step 11)**

Lecture 4

Historical Background

System Call for Process Management:

Fork is the only way to create a new process in POSIX. It creates an exact duplicate of the original process, including all the file descriptors, registers—everything.

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

Lecture 4

Historical Background

System Calls for Process Management:
Fork is the only way to create a new process in POSIX. It creates an exact duplicate of the original process, including all the file descriptors, registers—everything.

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

Lecture 4

Historical Background

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

Lecture 4

Historical Background

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

Lecture 4

Historical Background

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

Lecture 4 Historical Background

System Calls

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

Lecture 4

Historical Background

System Programs;

- Provide an environment where programs can be developed and executed
- System Programs also provide a bridge between the user interface and system calls
- Used to program the operating system software
- Most system programs are created to have a low runtime overhead
- Have Small runtime library
- Some part of system programs may be directly written in Assembly language by the programmers
- Debuggers cannot be used mostly
- Examples: operating system, networking system, web site server, data backup server etc.

Lecture 4 Historical Background

System Programs

- System program serves as a **part of the operating system**.
- It traditionally **lies between user interface and the system calls**.
- The user view of the system is actually defined by system programs and not system calls because that is what they interact with and system programs are closer to the user interface

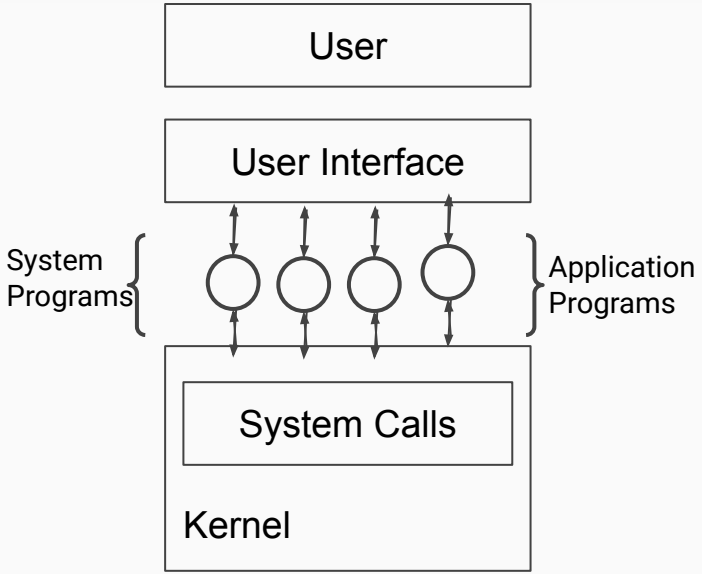


Fig: System Program Structure

Lecture 4

Historical Background

Types of System Programs

Status Information:

- provides required data on the current of past status of the system
- System date, system time, available memory in system, disk space, logged in users

Communications:

- Needed for system communications such as web browsers.
- Web browsers allow systems to communicate and access information from the network as required

File Manipulation:

- Used to manipulate system files
- Done using commands like create, delete, copy, rename, print, etc.

Lecture 4

Historical Background

Types of System Programs

Program Loading and Execution:

- Make sure that programs can be loaded into memory and executed correctly
- Examples: Loaders and Linkers

File Modification:

- Change data in the file or modify it.
- Text editors (GNU text editors, Nano text editor)

Lecture 4

Historical Background

Types of System Programs

Application Program:

- Perform a wide range of services as per the users needs.
- Program for database systems, word processors, plotting tools, spreadsheets, games, scientific applications etc.

Programming Language Support:

- Provide additional support features for different programming languages
- Compilers, debuggers

Lecture 4

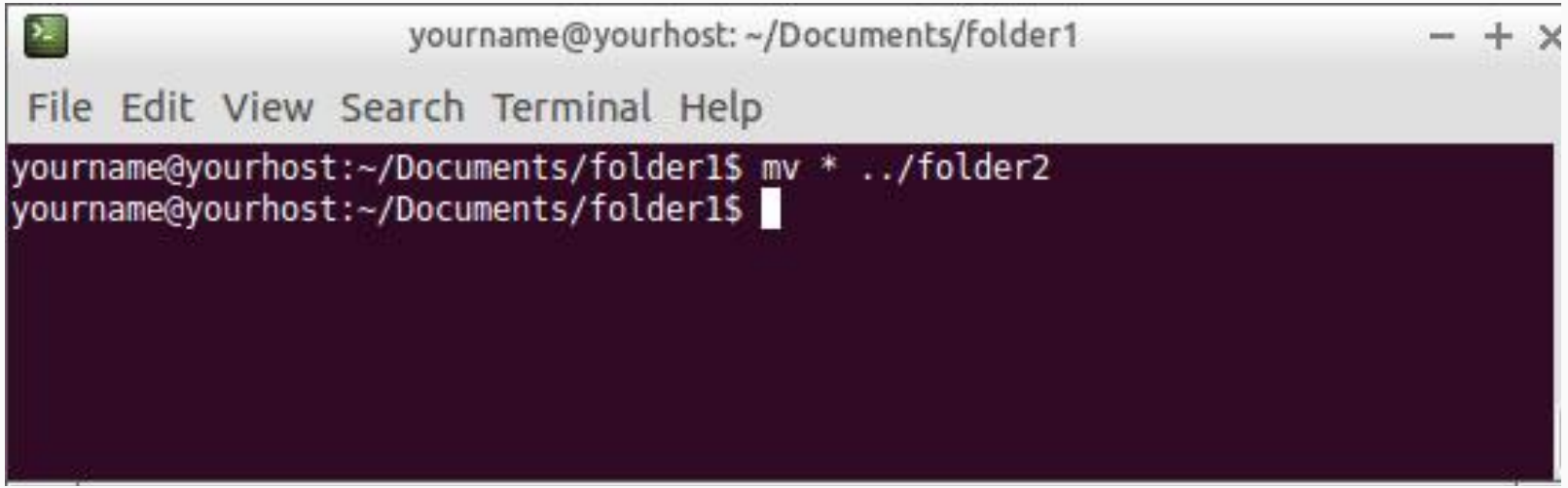
Historical Background

The Shell

- Operating system is a set of complex program that carries out the system calls
- Editors, compilers, assemblers, linkers, and command interpreters definitely are not part of the operating system, even though they are important and useful
- **Shell is a command interpreter which is used to execute system calls and receive response from the Operating System**
- The **main interface** between a **user** sitting at his terminal and the **operating system**, unless the user is using a graphical user interface
- Examples: **MS-DOS, PowerShell, sh, csh, ksh, and bash.**

Lecture 4

Historical Background



Lecture 4

Historical Background

The original shell (**sh**) starts with typing **prompt**

Commands:

- **date**
- **date > file**
- **sort <file1> file2**
- **cat file1**
- **cat file2 | sort >/dev/lp**
- **locate file.gz**
- **cat file5 | grep home**

- **Pwd**
- **ls**
- **cd**
- **mkdir**
- **rm**
- **cat**
- **less filename**
- **nano**
- **sudo**
- **shutdown**
- **restart**

Lecture 4

Historical Background

```
#define TRUE 1

while (TRUE) {                                  /* repeat forever */
        type_prompt( );                         /* display prompt on the screen */
        read_command(command, parameters);      /* read 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 */
        }
}
```

Lecture 4

Historical Background

Open Source Operating Systems

- **Open Source refers to any program whose source code is made available for use or modification as users other developers see fit.**
- Usually developed as a public collaboration and made freely available
- Released under a license where the copyright holder allows other to study, change as well as distribute software to other people

Cosmos

- **C# Open Source Managed Operating System**
- Open Source OS written mostly in programming language C#
- Till 2016, Cosmos didn't intend to be a fully-fledged operating system but a system that allowed other developers to easily build their own operating systems
- Hid inner working of the hardware from the developers' thus providing data abstraction

Lecture 4

Historical Background

Open Source Operating Systems

FreeDOS

- Free OS developed for systems compatible with IBMPC computers
- Provides a complete environment to run legacy software and other embedded systems
- Can boot from a floppy disk, or USB flash drive
- Licenced under GNU General Public license and contains free and open source software
- No license fees required for its distribution
- changes to the systems are permitted

Lecture 4 Historical Background

Open Source Operating Systems

Genode

- Free as well as open source
- **Contains a microkernel layer and different user components**
- Can be used as an operating system for computer, tablets
- Used as a base for virtualization, inter-process communication, software development etc.

GhostOS

- Free. open source OS developed for personal computers
- Started as a research project and developed to contain various advanced features like GUI, C libraary etc.
- Features multiprocessing and multitasking and is based on Ghost Kernel
- Programming done in C++

Lecture 4

Historical Background

Open Source Operating Systems

ITS

- Incompatible time-sharing system developed by MIT AI library
- Remote login facility
- Device independent graphics terminal, inter machine file system access

Phantom OS

- OS based on the concepts on persistent virtual memory and is code oriented
- It's main goal is simplicity and effectiveness in process management

Lecture 4

Historical Background

Some useful terminologies

- Mount
- Link
- pipeline
-

UNIT 1

Summary

Topics studied in UNIT 1

- **Operating System Definitions**
  Collection of software that directs a computer's operation, controlling and scheduling the execution of other programs, and manage resource, input/output, and communication resources
- **Operating System acts as an extended machine**
  Users are not concerned about where the data is written, OS manages the hardware resources hiding the truth about hardware and presents a nie simple view of named files
- **Operating System acts as a Resource Manager**
  Multiple users call for single resource to get their job done. It's OS responsibility to manage the resources
- **HIstory of OS(Generations)**
  Vacuum Tubes, Transistors & Batch systems, ICs and Multiprogramming, Personal Computers

UNIT 1

Summary

Topics studied in UNIT 1

- **Computer Hardware Review**
  Processors, Moore's Law, Thread, Multicore and Multithreaded Chips, I/O Device, Busy Waiting & Interrupt
- **Bus**
- **The boot Process**
  BIOS-> Boot Medium-> Boot sector->Partition Table->Configuration from BIOS-> Loads Kernel->GUI
- **Variants of Operating Systems**
  Mainframe, Server, Multiprocessor, personal, Handheld, Embedded, Sensor node, Real-time, Smart Card
- **System Calls, Shell & Open Source Operating Systems**

Q&A

For Further Queries:
sigdelsanjog@gmail.com

[NISTBanepa]

THANK YOU