COMP 8567

Advanced Systems Programming

Introduction

Summer 2023

Dr. Prashanth Ranga

Where is Linux used?

- **Webservers**

  Linux is used to power **96.3%** of the world's top 1 million web servers. Windows (1.9%), and FreeBSD 1.8%) are the other players: https://www.enterpriseappstoday.com/stats/linux-statistics.html
- Single Board Computers (**Raspberry Pie**)
- **Android** uses a modified version of the Linux Kernel
- **macOS** is based on a BSD Unix kernel known as Darwin which is open-source.
- **iOS** is a variant of Darwin, derived from BSD, a UNIX-like kernel
- **Supercomputers**

  https://en.wikipedia.org/wiki/TOP500

  List of the fastest supercomputers in the world.

COMP 8567 – List of Topics

1. Introduction to UNIX
2. Advanced C programming techniques
3. Unix Input/Output
4. Process Management and Control
5. Signals
6. Inter Process Communication (Pipes)
7. Unix Shells
8. Multithreading
9. Client-Server applications

Introduction to Unix –Outline

- Connecting to the Remote Linux Desktop
- The Role of Operating Systems
- Architecture of the Unix Operating System
- GNU
- History of Unix
- Other implementations of UNIX
- Shells
- **File System**
- **Input Output**
- **Process and Process ID**
- **Signals**
- **Pipes**
- **System Calls and Library Functions**
- Conclusion and Summary

Remote Desktop

Remote Desktop via a Web Browser

http://nx.cs.uwindsor.ca

Since the School of Computer Science has a NoMachine Enterprise license, the **cs.uwindsor.ca** remote desktop can be accessed directly from the **browser** through the link provided!

This method of remote access requires neither the NoMachine client nor the VPN client.

Download the App (Recommended)

https://www.nomachine.com/download/download&id=8

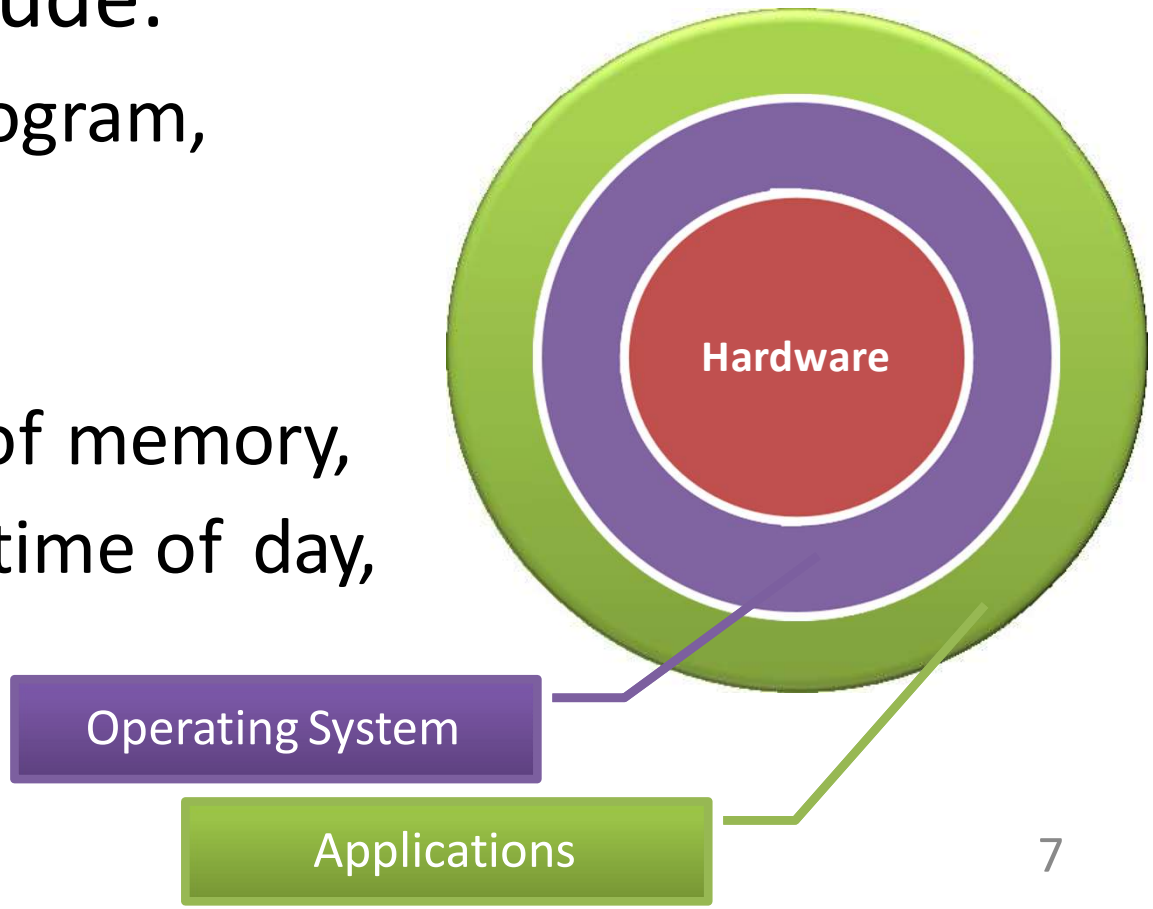
Sample C Program (welcome.c)

```
# include <stdio.h>  //welcome.c

main()
{

        printf("\nWelcome to COMP 8567\n");

}
```

```
$ gcc -o welcome welcome.c

$ ./welcome
```

Role of Operating Systems

- All operating systems provide services for programs they run.
- Typical services include:
  - executing a new program,
  - opening a file,
  - reading a file,
  - allocating a region of memory,
  - getting the current time of day,
  - and so on.

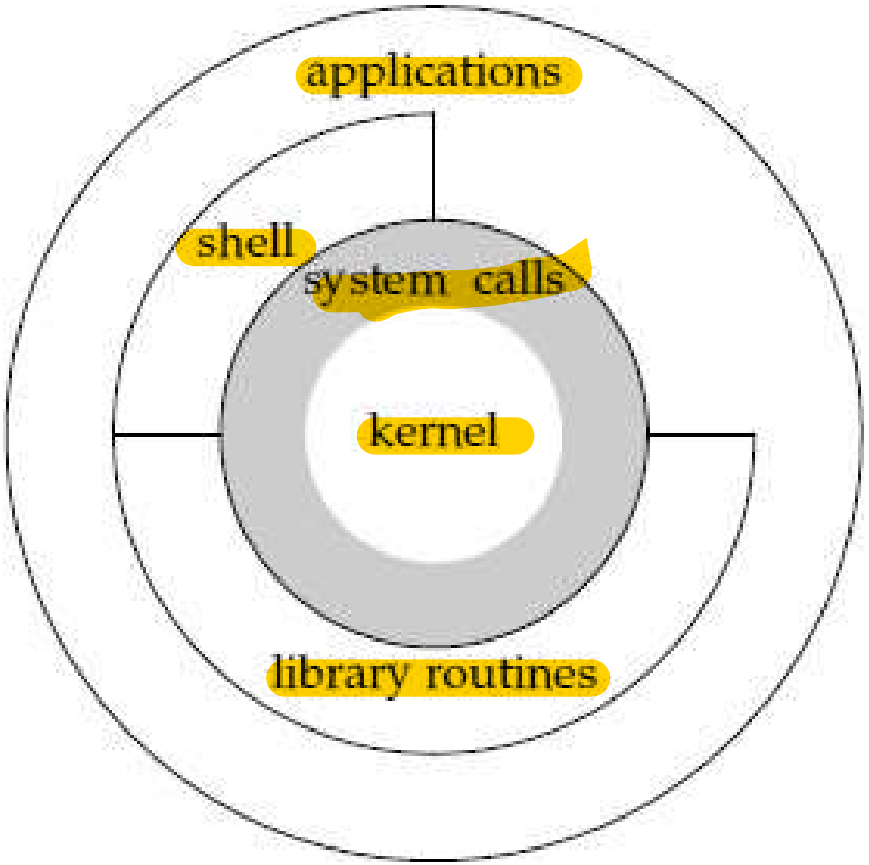


Major Software Components of OS

- Process Management
- Memory Management (Primary Memory)
- File Management (Secondary Memory)
- I/O Management
- Network Management
- Security Management

Architecture of the UNIX operating system

- The core of the UNIX OS is called the ***kernel.***
- The ***kernel*** directly interacts with the hardware and provides services to the applications
- The interface to the kernel is a layer of software called the ***system calls*** .
  - A system call is the programmatic way in which a computer program requests a service from the kernel of the operating system on which it is executed[1]
- **Libraries** of common functions are built on top of the system call interface, but applications are free to use both.
- The **shell** is a special application that provides an interface for running other applications.



For example, **Linux** is the kernel used by the GNU operating system.

History of Unix

- 1969-71 AT&T Bell Labs
- MULTICS: Multiplexed Information and Computing Service
  - Mainframe-TimeSharing- MULTICS was used to mange Time-sharing
- AT&T Stopped the project
- Side Project was developed by the employees who were interested
- MULTICS became a single user system UNICS (Uniplexed information and Computing Service)
  - UNICS was modified to support multiple users and eventually became Unix (x for multiple users)
- 1972: Unix was rewritten in C
  - Unix became portable to other hardware

History..

- 1975 –US govt bans AT&T from making any software
- AT& T gave away free software (Unix) to almost anyone who requested for it -Universities, Govt agencies and Corporations
- A number of mainframes were now running on Unix

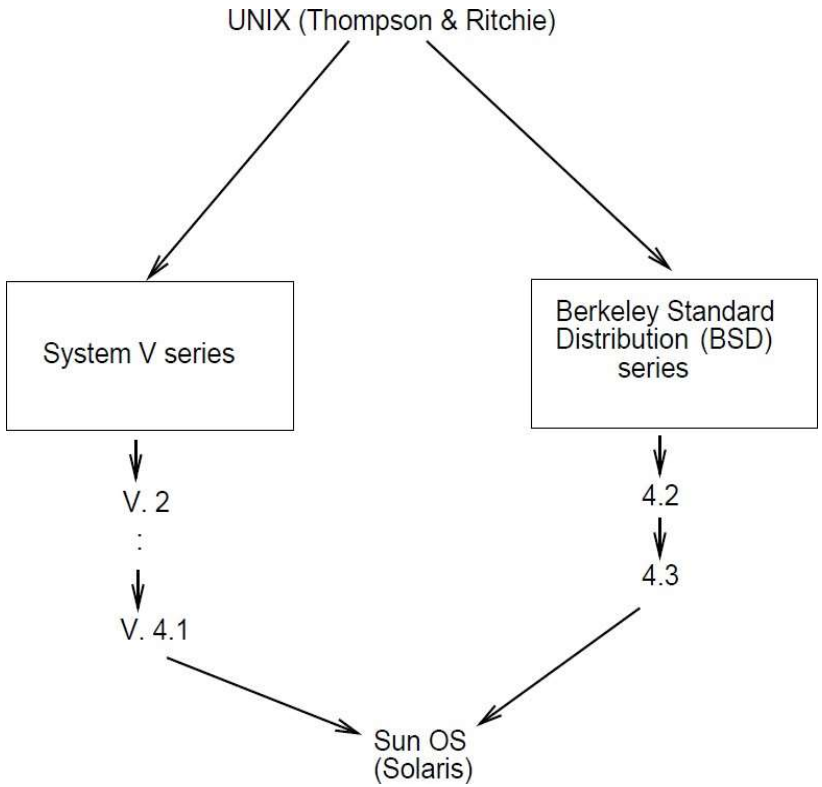
- First public version in 1975 is referred to as system 5 Open Source BSD, Linux

- Closed Source (Solaris, AIX etc) Proprietary HPUX

- Mixed Source (Mac OS / Darwin)
  IOS, ANDROID

Classic versions

- Two (classic) popular versions of Unix :
  - BSD (Berkeley Standard Distribution) Unix: introduction of Socket (Academics-Oriented)
  - System V: from Bell Laboratories (Industry-Oriented)



Other implementations of Unix

- **FreeBSD:** post BSD line after UCB decided to end work on BSD versions. Both binary and sources of the FreeBSD are freely available
- **Linux:** Freely available under GNU public license. Created in 1991, by **Linus Torvalds** to replace Minix
  - Became extremely popular(E.g., Debian, Ubuntu), in particular
    - **it is widely used on servers and mainframe computers**
    - **Android is built on top of the Linux kernel**

Where is Linux used?

- **Webservers**

  Linux is used to power **96.3%** of the world's top 1 million web servers. Windows (1.9%), and FreeBSD 1.8%) are the other players: https://www.enterpriseappstoday.com/stats/linux-statistics.html
- Single Board Computers (**Raspberry Pie**)
- **Android** uses a modified version of the Linux Kernel
- **macOS** is based on a BSD Unix kernel known as Darwin which is open-source.
- **iOS** is a variant of Darwin, derived from BSD, a UNIX-like kernel
- **Supercomputers**

  https://en.wikipedia.org/wiki/TOP500

  List of the fastest supercomputers in the world.

Other implementations of Unix..

- **Mac OS X:** a set of Unix-based operating systems with a graphical interface.
  - The core operating system is called **Darwin**.
  - Based on the FreeBSD OS.
- **IOS,** the mobile OS for iPhone/iPod/iPad and Apple TV, is based on Darwin with many similarities to Mac OS X.
- **Yosemite** is the later version of Mac OS X (10.10), released in Oct. 2014.
- **Android :** Uses Linux Kernel

Other implementations of Unix..

- **AIX**: (Advanced Interactive eXecutive), IBM's own version of Unix
- **HP-UX**: Hewlett-Packard Unix.

Shells

- A ***shell*** is a command-line interpreter that reads user input and executes commands.
- The user input to a shell is normally from the **terminal** (an interactive shell) or sometimes from a file (called a ***shell script***).

Name	Path	FreeBSD 8.0	Linux 3.2.0	Mac OS X 10.6.8	Solaris 10
Bourne shell	`/bin/sh`	•	•	copy of `bash`	•
Bourne-again shell	`/bin/bash`	optional	•	•	•
C shell	`/bin/csh`	link to `tcsh`	optional	link to `tcsh`	•
Korn shell	`/bin/ksh`	optional	optional	•	•
TENEX C shell	`/bin/tcsh`	•	optional	•	•

Figure 1.2 Common shells used on UNIX systems

//Current Shell

$ echo $SHELL

//Various shells supported by the system

$ cd /

$ cd bin

$ ls *sh -1

//Get the Version of Linux

$ uname –a

FILE SYSTEM

File System

- The UNIX file system is a hierarchical arrangement of **directories and files.**
- Everything starts in the directory called ***root***, whose name is the single character /
- A ***directory*** is a file that contains directory entries. Logically, we can think of each directory entry as containing a filename along with a structure of information describing the attributes of files and other directories.
- The attributes of a file are such things as the type of file (regular file, directory), the size of the file, the owner of the file, permissions for the file (whether other users may access this file), and when the file was last modified.
- The `stat` function returns information containing all the attributes of a file.
- $ cd / $ cd home $ cd pranga
- Home directory is denoted by the ~ symbol
- **$ stat sample.c**

Filename

- The only two characters that **cannot** appear in a filename are the slash character (/) and the null character.
- Characters are **CASE-SENSITIVE**
- The slash separates the filenames that form a pathname and the null character terminates a pathname.
- For portability, POSIX.1**(Portable Operating System Interface-IEEE standard)** recommends restricting filenames to consist of the following characters: letters (a-z, A-Z), numbers (0-9), period (.), dash (-), and underscore (_).
- Two filenames are automatically created whenever a new directory is created: .(called *dot*) and .. (called *dot-dot*).
- **$ cd ..**
- **$ cd .**
- Dot refers to the current directory, and dot-dot refers to the parent directory. In the root directory, dot-dot is the same as dot.

Pathname

- A sequence of one or more filenames, separated by slashes and optionally starting with a slash, forms a *pathname*.
- A pathname that begins with a slash is called an ***absolute pathname***; otherwise, it's called a ***relative pathname***. Relative pathnames refer to files relative to the current directory.
- The name for the root of the file system (/) is a special-case absolute pathname that has no filename component.

Absolute and Relative Pathnames and Other file operations

$ cd Music

$ cd /home/pranga/Music

//Moving files from one folder to another
// cp, mv and rm

$ cp hello.c Music // Keeps the original file after copying
$ mv hello.c Music //Removes/deletes the original file after moving
$ cd Music
$ rm hello.c //Permanently deletes the original file
//Rename hello.c to helloworld.c
$ mv hello.c helloworld.c

Working Directory

- Every process has a *working directory*, sometimes called the *current working directory*.

  `pwd` //**Print Working Directory**

- This is the directory from which all relative pathnames are interpreted.

Home Directory

- When we log in, the working directory is set to our *home directory*.
- Our home directory is obtained from our entry in the password file

```
cd
cd ~
```

INPUT AND OUTPUT

Input and Output

- **Standard Input, Standard Output, and Standard Error**
- By convention, all shells open three descriptors whenever a new program is run: standard input, standard output, and standard error.
- If nothing special is done, as in the simple command `ls` then all three are connected to the terminal.
- Most shells provide a way to redirect any or allof these three descriptors to any file.
- For example, `ls > listing.txt` executes the ls command with its standard output redirected to the file named listing.txt
- (use < for input redirection, >> for concatenation of output)

PIPES

Pipes

- **Pipes** : a mechanism that allows the user to specify that the output of one program is to be used as the input of another program.
  - several programs can be connected in this fashion to make a pipeline of data owing from the first process through the last one.

Example :

```
Ls -1 | grep .c | wc -w
```

PROCESSES AND PROCESS ID

Processes and Process ID

- An executing instance of a program is called a ***process***. (Some operating systems use the term *task* to refer to a program that is being executed).
- The UNIX System guarantees that every process has a unique numeric identifier called the ***process ID***. The process ID is always a non-negative integer.
- Try ps(Process Status)
- Every process has a **process id**, **parent process** id and **group id**.

Process ID, Parent ID, Group ID

```
#include <stdio.h>
#include <unistd.h>


// processids.c
//process id, parent id, process group id
        main(void)
        {

                printf("\n The current process id is %d \n",getpid());

                printf("\n The parent id of the current process is %d \n",getppid());

                printf("\n The group id of the current process is %d \n",getgid());

        }
```

Three Important Process- Related Operations

- Fork() //Used to create a new child process
- Exec() //Used to replace a child process with a new process
- Waitpid() // Used to wait for a process to complete execution

User ID

- The *user ID* from our entry in the password file is a numeric value that identifies us to the system.
- This user ID is assigned by the **system administrator** when our login name is assigned, and we cannot change it.
- The user ID is unique for every user.
- The kernel uses the user ID to check whether we have the appropriate permissions to perform certain operations.
- We call the user whose user ID is 0 as the ***superuser*.** The entry in the password file normally has a login name of root, and we refer to the special privileges of this user as **superuser privileges**.

A simple C program that prints the userid using the getuid() system call

```
// userid.c
//uses getuid() system call

#include <stdio.h>
#include <unistd.h>

        main(void)
        {
                printf("\nThe current userid is %d\n",getuid());
        }
```

SIGNALS

Signals

- Signals (Software Signals) are used to notify a process of the occurrence of some condition.
- For example, the following generate signals :
- A division by zero : the signal **SIGFPE** is sent to the responsible process that has three choices. Ignore the signal, terminate the process or, call a function to handle the situation.
- The Control-C key : when pressed, it generates a signal that causes the process receiving it to interrupt.
- Calling the function *kill* : a process can send a signal to another process causing its death. This is an example where Unix checks our permissions before allowing the signal to be sent.
  - A number of signals can be used along with the ***kill*** command: $ kill –l

Kill Processes – Example

Run Multiple Processes and Kill Processes

$ ps –u
$ kill SIGKILL *Processid*

$Kill –l *//List of all signals*

SYSTEM CALLS AND LIBRARY FUNCTIONS

System calls and library functions

- "In computing, a system call is the programmatic way in which a computer program **requests a service from the kernel** of the operating system on which it is executed" [1]
- Each system call in Unix has an **interface function**, in the C standard library, with the same name that the user process invokes.
- The interface function then invokes the appropriate kernel service, using whatever technique is required on the system.
- An interface function for a system call cannot be re-written/overridden by the user
- For our purpose, a system call will be viewed as a **regular C function.**

System Calls- Common Examples

Open()
Read()
Write()
Close()
Fork()
Exec()
Exit()
Getpid()
Getgid()
Getuid()
--
--
--

A simple C program that prints the current process id using the getpid() system call

```
// pid.c
//uses getpid() system call

#include <stdio.h>
#include <unistd.h>

        main(void)
        {
                printf("\nThe current process id is%d\n",getpid());
        }
```

System calls and library functions

- The functions which are a part of standard C library are known as Library functions: strcmp(), strlen() etc are library functions.
- Note that a user process can invoke either a system call or a library function.
- A library function might invoke a system call.

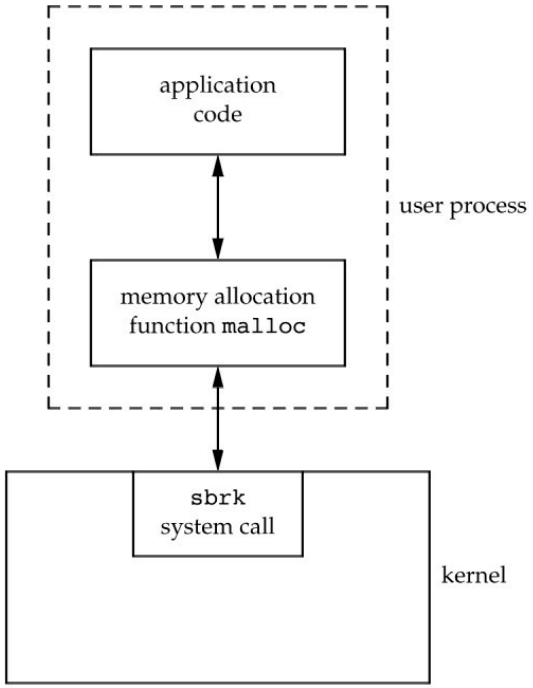


Figure 1.11 Separation of `malloc` function and `sbrk` system call

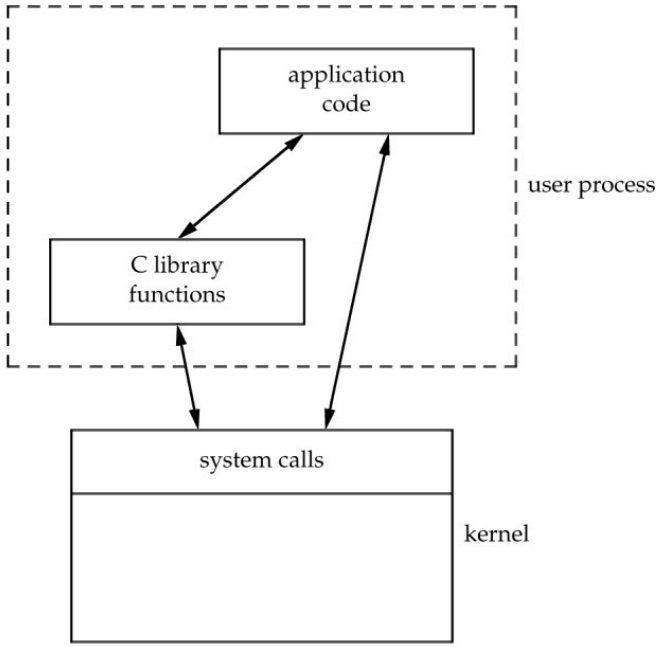


Figure 1.12 Difference between C library functions and system calls

Summary

- Operating system: Software to manage computer resources, in particular,
  - it runs a program for a user
  - it allows communication with devices and processes
- A program is a le containing instructions
- A process is a program being executed
- Unix is a multiuser operating system
- Most of Unix is written in the C language
- Unix has a simple philosophy: a program should do one thing and do it well.
- Entry points in Unix are called system calls. They allow the user to get services from the kernel.

LinkedIn Learning (Lab 1)
Due: May/16/2023

Unix Essential Training:

https://www.linkedin.com/learning-login/share?account=2212217&forceAccount=false&redirect=https%3A%2F%2Fwww.linkedin.com%2Flearning%2Funix-essential-training%3Ftrk%3Dshare_ent_url%26shareId%3DnHIsKhovToqnR9nuQigrsQ%253D%253D

APPENDIX

Linux

6.3.1 (Apr/30/2023 –SR

6.4-rc1 (May/07/2023 –PR)

version 11 (Debian)
The current stable distribution of Debian is version 11, codenamed bullseye. It was initially released as version 11.0 on August 14th, 2021 and its latest update, version 11.7, was released on April 29th, 2023. {debian.org)