Operating Systems and Systems Programming I (*CPS1012*) Short Notes

Aiden Cachia

Contents

1 Intoduction

What is an Operating System?

An Operating System is a program that acts as an intermediary between a user of a computer and the computer hardware.

Operating System Goals

- Execute user programs and make solving user problems easier.
- Make the computer system convenient to use.
- Use the computer hardware in an efficient manner.

Computer System Structure

- Hardware.
- Operating System.
- Application programs.
- Users.

1.1 Operating System Services

- User Interface (CLI, GUI, Touch-screen, batch)
- Program Execution
- I/O Operations
- File-system manipulation
- Communications (On Device, Network)
- Error Detection
- Resource Allocation
- Protection and Security

Command Line Interface

Simple Interface where command is fetched from user, it is exectued and the output is displayed.

Graphical User Interface

More user friendly design, making use of;

- Mosue, Keyboard and monitor
- Icons representing files, programs, actions etc ...
- Menus

1.2 System Calls

Ways to interact with the Operating System. They are typically written in a high level language (C or C++) and are used to request a service from the Operating System.

Implementation of System Calls

- A table of all is usualy stored in the System-call interface.
- The System-call invokes the system-call in OS kernel and returns the status of the system-call and any return values.

System Call Parameters

- Registers
- Block/table
- Stack

Types of System Calls

Process Control

- Create process, terminate process

- End, abort
- Load, execute
- Get process attributes, set process attributes
- Wait for time
- Wait for event, signal event
- Allocate and free memory
- Dump memory
- Debugging
- Locking

File Management

- Create file, delete file
- Open, close
- Read, write, reposition
- Get file attributes, set file attributes

Device Management

- Request device, release device
- Read, write, reposition
- Get device attributes, set device attributes
- Logically attach or detach devices

Information Maintenance

- Get or set
  - time or date
  - system data
  - process, file or device attributes

Communications

- Create, delete communication connection

- Send, receive messages
- Shared memory
- Transfer status information
- Attach or detach remote devices

1.3 System Services

Helps in the development of programs. Some are just system-call interfaces, others more complex.

Types of System Services

- File management

  Create, delete, copy, rename, print, dump, lis and generaslly manipulate files and directories.

- Status information
  - Basic information (date, time, amount of available memory, disk space, number of users).
  - Detailed logging and Debugging information.
  - Registry to store and retrieve configuration information.
  - File modification
  - Programming language support
  - Program loading and execution
  - Communications
- Programming language support
- Program loading and execution
- Communications
- Background services
  - Launch at boot
  - Provides facilities like disk checking, process scheduling, error logging, printing.
  - Run in user context not kernel context
  - Known as *services*, *subsystems* or *daemons*
- Application programs
  - Unrelated to system
  - Run by users
  - Not part of OS
  - Launched by CLI, Mouse click, finger poke

Linkers and Loaders

A Linker is used to link multiple object files (compiled from source code) into a single executable file.

A Loader is used to load the executable file into memory and start the execution of the program.

Relocation is the process of adjusting the addresses in the object file to match the addresses in memory.

Dynamic linked libraries (DLLs) are libraries that are linked at runtime, rather than at compile time, so they could be used by multiple programs that would require the same version of the library.

Why Applications are Operating System Specific

System calls and file formats are different from one operating system to another, among other things. Because of this applications are not portable between operating systems without modification.

Applications can be made portable:

- By using an interpreted language such as Python or Ruby;
- By using a language which include a VM such as Java;
- By using compiling a program which a User standard language (such as C), then recompiling it in target OSes.

An Application Binary Interface (ABI) is a specification defining requirements for an application on a specific hardware platform.

1.4 Operating System Design

There is no correct way to design an Operating System, but there are some common design principles.

An OS should have User goals (ease of use, convience, etc ...) and System goals (efficiency, security, etc ...).

Policy and Mechanism

Policy is what will be done, *Mechanism* is how it will be done.

They should be seperate, as it would allow for maximum flexibility if policy decisioons are to be chanced later.

Implementation

Early OSes were written in Assembly, but now they only have the base of the OS writen in Assembly, and the rest written in high level languages such as C or C++.

Emulation allows for OS to ron on non-native hardware.

The Kernel

The Kernel is the core of the OS, which is used to manage system resources and provides essential services for all other parts of the OS and user-level programs.

Handles tasks such as:

- Process management;
- Memory management;
- Scheduling;
- Interfacing with hardware devices.

Operating System Structure

- Simple Structure - MS-DOS
- More complex - UNIX
- Layered - an abstraction
- Microkernel - Mach

Monolithic - OG Unix

Structure

- System Programs
- The kernel, which contains everything between the system-call interface and the hardware, providing File system, CPU scheduling, memory management, etc …

Layered

Like an Onion, the Core being Hardware (Layer 0), and the outer being the User Interface (Layer N). Each layer only has access to the layers below it.

Microkernel

A microkernel is the near-minimum amount of software that can provide the mechanisms needed to implement an OS.

Modules

Most modern OSes maks use of Loadable Kernel Modules (LKMs) which can be loaded and unloaded at runtime, Each:

- uses Object-oriented approach;
- core component is seperate;
- talks to the other over known interfaces;
- is loadable as needed within the kernel.

Hybrid Systems

Most modern OSes are hybrids of the above systems inorder to address performance, security and usability needs.

Dual-mode Operation

Dual-mode operation allows OS to protect itself and other system components.

Mode bit is used to distinguish between user and kernel mode, which is provided by hardware.

2 Processes

A *process* is a program in execution.

Its execution must be sequential. No parallel executution of a single process.

A program becomes a process once it is **active** and **loaded into memory**