Operating Systems

Functions and Services

Key functions of an operating system

- Process Management
- Main Memory Management
- File Management
- I/O System Management
- Secondary **Storage** Management **(disk)**
- Networking
- Protection System
- Command-Interpreter System

- A *process* is a program in **some "state" of** execution. A process needs certain resources, including CPU time, memory, files, and I/O devices, to accomplish its task.
- **A process is a unit of work in a system**
- The operating system is responsible for the following activities in connection with process management.
  - ✓ Process creation and deletion.
    - ❑ **UNIX: processes should have the ability to dynamically (in real time) spawn off or create other processes**
  - ✓ process suspension **(process is in I/O wait queue, or "swapped" out to disk, …) and resumption (move to ready queue or execution) – manage the *state* of the process.**
  - ✓ Provision of mechanisms for:
    - ❑ process synchronization **- concurrent processing is supported thus the need for synchronization of processes or threads**.
    - ❑ process communication
    - ❑ **Deadlock handling**

- Memory is a large array of words or bytes, each with its own address. It is a repository of quickly accessible data shared by the CPU and I/O devices.
- Main memory is a volatile storage device. It loses it contents in the case of system failure.
- The operating system is responsible for the following activities in connections with memory management:
  - ✓ Keep track of which parts of memory are currently being used and by whom.
  - ✓ Decide which processes to load when memory space becomes available **- long term or medium-term scheduler.**
  - ✓ **Mapping addresses in a process to absolute memory addresses - at load time or run time.**
  - ✓ Allocate and deallocate memory space as needed.
  - ✓ **Memory partitioning, allocation, paging (VM), address translation, defrag, …**
  - ✓ **Memory protection**

- **The OS abstracts the data stored on a physical device to a logical unit: <u>The File</u>**
- A file is a collection of related information defined by its creator. Commonly, files represent programs (both source and object forms) and data **- identifiable by name and location**.
- The operating system is responsible for the following activities in connections with file management:
  - ✓ File creation and deletion **- system calls or commands**.
  - ✓ Directory creation and deletion **- system calls or commands**.
  - ✓ Support of primitives for manipulating files and directories **in an efficient manner - system calls or commands**.
  - ✓ Mapping files onto secondary storage.
  - ✓ File backup on stable (nonvolatile) storage media.
  - ✓ **EX: File Allocation Table (FAT) for Windows/PC systems**

- **Hide the peculiarities of a specific HW device from the user - device drivers**
- The I/O system consists of:
  - ✓ A buffer-caching system
  - ✓ A general device-driver interface **– part of OS**
  - ✓ Drivers for specific hardware devices **– OS must provide for all devices.**
  - ✓ **Provide system call API for I/O - I/O is a privileged operation**

Secondary-Storage Management

- Since main memory (*primary storage*) is volatile and too small to accommodate all data and programs permanently, the computer system must provide *secondary storage* **(disk)** to back up main memory **– basis for virtual memory**.
- Most modern computer systems use disks as the principle on-line storage medium, for both programs and data.
- The operating system is responsible for the following activities in connection with disk management:
  - ✓ Free space management
  - ✓ Storage allocation
  - ✓ Disk scheduling **– minimize seeks (arm movement … very slow operation)**
  - ✓ **Disk as the media for mapping virtual memory space**
  - ✓ **Disk caching for performance**
  - ✓ **Disk utilities: defrag, recovery of lost clusters, etc.**
- **Accessing secondary storage is very frequent, thus disk performance can be a performance bottleneck for the entire system**
  - ✓ **Minimize head seeks (cylinder transitions)**

- **Low level support for pure connectivity - message passing, FTP, file sharing, …**
- **Higher level functional support: clustering, parallel processing, …**
- A *distributed* system is a collection **heterogeneous** processors that do not share memory or a clock. Each processor has its own local memory. **Connected by a network**.
- The processors in the system are connected through a communication network.
- Communication takes place using a *protocol.*
- A distributed system provides user access to various system resources.
- **Cooperative vs independent processing**.
- Access to a shared resource allows:
  - ✓ Computation speed-up
  - ✓ Increased data availability
  - ✓ Enhanced reliability

- **Keep processes from interfering with each other**
- *Protection* refers to a mechanism for controlling access by programs, processes, or users to both system and user resources.
- The protection mechanism must:
  - ✓ distinguish between authorized and unauthorized usage.
  - ✓ specify the controls to be imposed.
  - ✓ provide a means of enforcement.

- **Hardware assists in screening addresses for illegal references.**

Command-Interpreter System

- Many commands are given to the operating system by control statements which deal with:
  - ✓ process creation and management
  - ✓ I/O handling
  - ✓ secondary-storage management
  - ✓ main-memory management
  - ✓ file-system access
  - ✓ protection
  - ✓ Networking
  - ✓ **Commands may have counterparts for use in programming**
- **ASCII command line vs. graphic interface, Windows explorer, “add-ins” like Norton Utilities**

- The program that reads and interprets control statements is called variously:
  - ✓ command-line interpreter (Control card interpreter in the "old batch days")
  - ✓ shell (in UNIX)
  - ✓ **Command.com for commands in DOS**

  Its function is to get and execute the next command statement.

- **Example: DOS window, UNIX command line, Windows "run" window**

Operating System Services

- **OS as a service provider via system calls & commands (typically for the programmer).**
- **Program execution** – system capability to load a program into memory and to run it **- address mapping and translation a key issue**.
- **I/O operations** – since user programs cannot execute I/O operations directly, the operating system must provide some means to perform I/O **- system calls and API**.
- **File-system manipulation** – ***program*** capability to read, write, create, and delete files
- **Communications** – exchange of information between processes executing either on the same computer or on different systems tied together by a network. Implemented via *shared memory* or *message passing*.
- **Error detection** – ensure correct computing by detecting errors in the CPU and memory hardware, in I/O devices, or in user programs **– ex:parity errors, arithmetic "errors", out of memory, out of disk space, program not found, …**
- **Memory management**

Additional Operating System Services

Additional functions exist not for helping the user, but rather for ensuring efficient system operations.

- Resource allocation – allocating resources to multiple users or multiple jobs running at the same time **– avoiding Deadlock**.
- Accounting – keep track of and record which users use how much and what kinds of computer resources for account billing or for accumulating usage statistics.
- Protection – ensuring that all access to system resources is controlled, **ex: firewalls, passwords, file permissions, etc.**

- System calls provide the interface between a running program and the operating system**– like invoking a command from inside a program.**
  - ✓ Generally available as assembly-language instructions.
  - ✓ Languages defined to replace assembly language for systems programming allow system calls to be made directly (e.g., C, C++) - **C language: open, close, read, write, …**
- Three general methods are used to pass parameters between a running program and the operating system.
  - ✓ Pass parameters in *registers*.
  - ✓ Store the parameters in a table in memory, and the table address is passed as a parameter in a register **– if parms complicated.**
  - ✓ *Push* (store) the parameters onto the *stack* by the program, and *pop* off the stack by operating system **– usually in assembly language.**

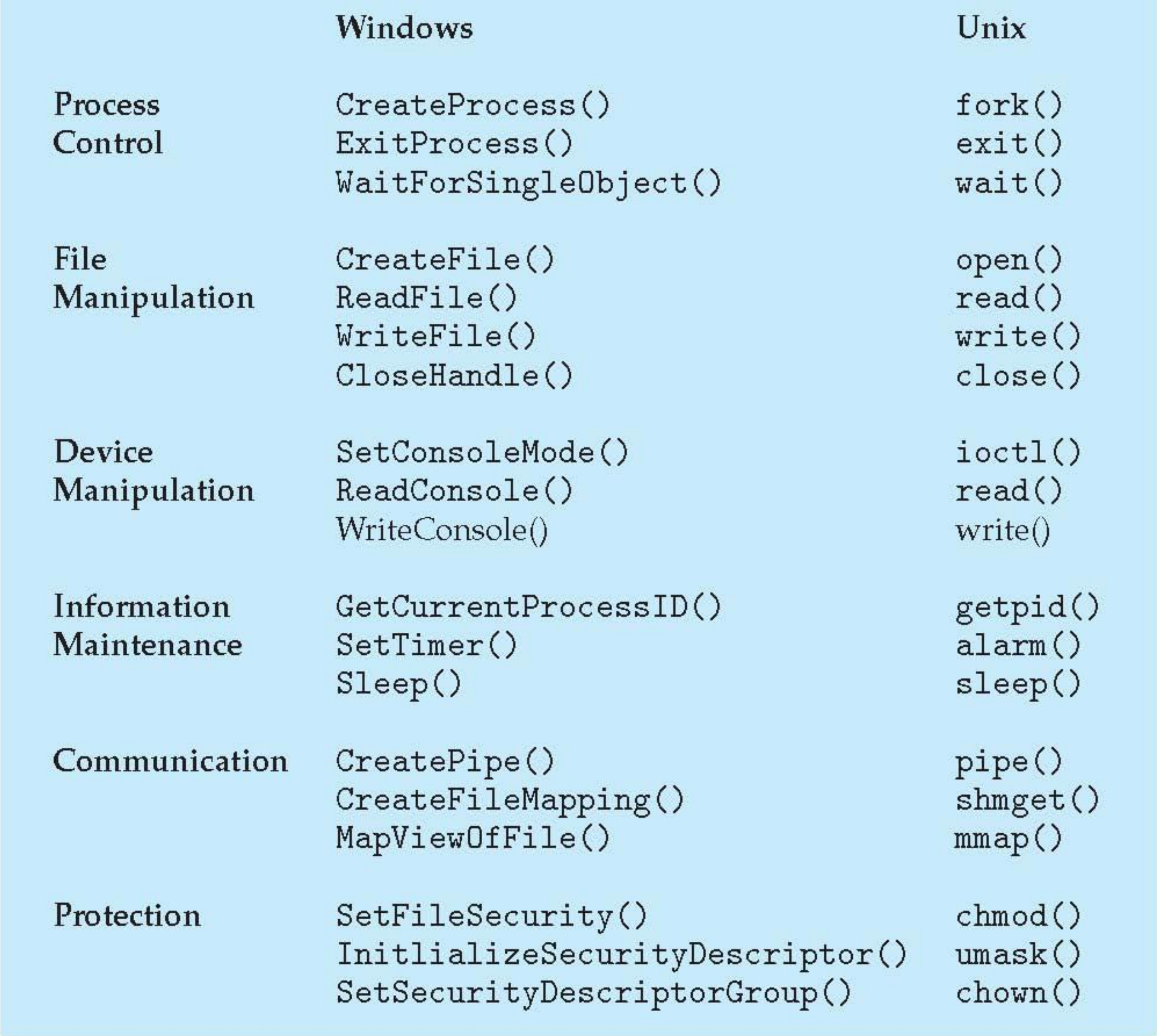
Types of System Calls

- Process control
  - ✓ create process, terminate process
  - ✓ end, abort
  - ✓ load, execute
  - ✓ get process attributes, set process attributes
  - ✓ wait for time
  - ✓ wait event, signal event
  - ✓ allocate and free memory
  - ✓ Dump memory if error
  - ✓ **Debugger** for determining **bugs, single step** execution
  - ✓ **Locks** for managing access to shared data between processes
- File management
  - ✓ create file, delete file
  - ✓ open, close file
  - ✓ read, write, reposition
  - ✓ get and 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 time or date, set time or date
  - ✓ get system data, set system data
  - ✓ get and set process, file, or device attributes

- Communications
  - ✓ create, delete communication connection
  - ✓ send, receive messages if **message passing model** to **host name** or **process name**
    - ❑ From **client** to **server**
  - ✓ **Shared-memory model** create and gain access to memory regions
  - ✓ transfer status information
  - ✓ attach and detach remote devices
- Protection
  - ✓ Control access to resources
  - ✓ Get and set permissions
  - ✓ Allow and deny user access

Examples of Windows and Unix System Calls



	Windows	Unix
Process Control	CreateProcess()	fork()
	ExitProcess()	exit()
	WaitForSingleObject()	wait()
File Manipulation	CreateFile()	open()
	ReadFile()	read()
	WriteFile()	write()
	CloseHandle()	close()
Device Manipulation	SetConsoleMode()	ioctl()
	ReadConsole()	read()
	WriteConsole()	write()
Information Maintenance	GetCurrentProcessID()	getpid()
	SetTimer()	alarm()
	Sleep()	sleep()
Communication	CreatePipe()	pipe()
	CreateFileMapping()	shmget()
	MapViewOfFile()	mmap()
Protection	SetFileSecurity()	chmod()
	InitlializeSecurityDescriptor()	umask()
	SetSecurityDescriptorGroup()	chown()

Bibliography

- Silberschatz, A, Galvin, P.B, and Gagne, G., Operating System Principles, 9e, John Wiley & Sons, 2013.
- Stallings W., Operating Systems-Internals and Design Principles, 7e, Pearson Education, 2014.
- Harvey M. Deital, “Operating System”, Third Edition, Pearson Education, 2013.
- Andrew S. Tanenbaum, “Modern Operating Systems”, Second Edition, Pearson Education, 2004.
- Gary Nutt, “Operating Systems”, Third Edition, Pearson Education, 2004.

Acknowledgements

- I have drawn materials from various sources such as mentioned in bibliography or freely available on Internet to prepare this presentation.
- I sincerely acknowledge all sources, their contributions and extend my courtesy to use their contribution and knowledge for educational purpose.

Thank You!!

?