Operating Systems and Computer Networking — Terminology Sheet

CS2ON — Introduction to Operating Systems and System Calls (Lecture 1)

1. Operating System Fundamentals

Term	Definition	Source
Operating System	System software that acts as an intermediary between computer users and hardware. It manages hardware resources (CPU, memory, I/O devices) and provides common services for application programs, enabling convenient and efficient program execution.	Both
Computer Hardware	The physical components of a computer system — the central processing unit (CPU), main memory (RAM), and input/output (I/O) devices — that the operating system manages on behalf of user programs.	Both
Application Software	Programs that perform specific tasks for users (e.g., word processors, web browsers, compilers). Application software relies on the operating system for resource allocation, file management, and hardware access.	Both
System Software	Software that provides the platform for running application programs, including the operating system itself, device drivers, and system utilities. Distinct from application software in that it manages the computer rather than performing user tasks.	Both
Resource Manager	One of the two fundamental roles of an operating system: allocating CPU time, memory, and I/O bandwidth among competing processes fairly and efficiently.	PDF
Abstraction Provider	The second fundamental role of an operating system: hiding the complexities of hardware from application developers by presenting a clean, uniform interface, enabling portable, high-level code.	PDF
Shell	The outer layer of an operating system that provides the user interface (either a command-line interface or a graphical user interface). It accepts user commands, translates them into system calls, and relays results back to the user. Analogous to a receptionist.	Both
Kernel	The core component of an operating system that runs in privileged (kernel) mode with unrestricted hardware access. Responsible for process management, memory management, device driver coordination, and security enforcement. Analogous to an office manager.	Both
Command-Line Interface (CLI)	A text-based user interface in which users interact with the operating system by typing textual commands. Examples include Bash (Linux/macOS) and PowerShell (Windows).	Both
Graphical User Interface (GUI)	A visual user interface that allows users to interact with the operating system through windows, icons, menus, and pointers. Examples include Windows	Both

Desktop, GNOME, and macOS Aqua.

2. Operating System Services

Term	Definition	Source
Program Execution	An OS service that loads programs into memory, allocates CPU time, and manages their execution from start to termination.	Both
I/O Operations	An OS service that provides a uniform interface for applications to read from and write to I/O devices (disks, printers, network adapters) without accessing hardware directly.	Both
File System Manipulation	An OS service that manages the creation, deletion, reading, writing, and organisation of files and directories on storage media.	Both
Disk Management	An OS service that handles disk scheduling, space allocation, and defragmentation to optimise storage performance and reliability.	Both
Memory Management	An OS service that tracks which parts of memory are in use, allocates and deallocates memory for processes, and implements virtual memory through paging and segmentation.	Both
Protection and Security	An OS service that enforces access controls, authenticates users, and isolates processes from one another to prevent unauthorised access and ensure system integrity.	Both

3. Operating System Structures

Term	Definition	Source
Simple Structure	An early OS architecture (e.g., MS-DOS) with little or no separation between user-mode and kernel-mode code. Applications could access hardware directly, offering no protection, multitasking, or memory isolation.	Both
Monolithic Kernel	An OS architecture in which all system services — process management, memory management, file systems, device drivers, and networking — run in a single address space in kernel mode. Maximises performance through direct function calls but increases crash risk. Examples: UNIX, Linux, Windows NT.	Both
Layered Structure	An OS architecture that organises the system into a hierarchy of layers, each built on top of the one below. The lowest layer is the hardware; the highest is the user interface. Provides clean separation of concerns but can introduce performance overhead. Examples: THE (Dijkstra), OS/2.	Both
Microkernel	An OS architecture that keeps only the most essential services (IPC, basic scheduling, low-level address space management) in the kernel, running all other services as user-space processes. Offers high reliability and security at the cost of IPC overhead. Examples: Mach, MINIX, QNX.	Both
Modular Structure	An OS architecture that provides a core kernel with interfaces for dynamically loading additional functionality as kernel modules. Combines monolithic performance with modular flexibility. Examples: modern Linux,	Both

	Solaris.	
Loadable Kernel Module	A compiled piece of code (e.g., a device driver or file system) that can be inserted into or removed from a running kernel without requiring a reboot. Central to the modular OS structure.	Both
Inter-Process Communication (IPC)	The mechanisms by which separate processes exchange data or signals. In microkernel architectures, all non-essential services communicate with the kernel and each other via IPC, typically through message passing.	PDF

4. User Mode and Kernel Mode

Term	Definition	Source
Dual-Mode Operation	A CPU design that divides execution into user mode (restricted privileges) and kernel mode (full hardware access). A hardware mode bit tracks the current mode. This separation protects the OS from errant or malicious application code.	Both
User Mode	A restricted CPU execution mode in which application processes run. User-mode code cannot execute privileged instructions or access hardware directly. If it attempts a privileged operation, the CPU raises an exception. A crash in user mode affects only the offending process.	Both
Kernel Mode	A privileged CPU execution mode (also called supervisor mode) in which the operating system kernel runs. Code in kernel mode has unrestricted access to all hardware and memory. A crash in kernel mode can bring down the entire system (kernel panic / BSOD).	Both
Mode Bit	A hardware bit in the CPU that indicates the current execution mode: 0 for kernel mode, 1 for user mode. The system call instruction atomically switches this bit and transfers control to a predefined kernel entry point.	PDF
Privileged Instruction	A CPU instruction that can only be executed in kernel mode, such as directly accessing hardware registers, modifying page tables, or disabling interrupts. Attempting a privileged instruction in user mode raises a hardware exception.	PDF
Mode Switching	The transition between user mode and kernel mode, triggered by a system call, interrupt, or exception. The CPU atomically changes the mode bit and transfers control to the kernel entry point, then restores user mode upon completion.	Both
Kernel Panic	A fatal error condition in UNIX/Linux/macOS in which the kernel detects an internal inconsistency and halts the system. The Windows equivalent is the Blue Screen of Death (BSOD). Typically caused by bugs in kernel-mode code such as faulty device drivers.	PDF
Blue Screen of Death (BSOD)	The Windows equivalent of a kernel panic: a stop error screen displayed when the kernel encounters an unrecoverable fault. Requires a full system reboot.	PDF

5. Types of System Users

Term	Definition	Source
End User	A person who interacts with the computer through application programs and typically has no knowledge of the underlying operating system mechanisms. End users value simplicity, stability, and security.	Both
Application Programmer	A software developer who writes application programs using system APIs provided by the OS. Does not modify the OS itself but relies on it for file operations, process creation, memory allocation, and networking.	Both
System Programmer	A developer who works at the lowest level of the software stack, writing or modifying the OS kernel, device drivers, compilers, and system utilities. Requires deep understanding of hardware architecture and concurrency.	Both
POSIX	Portable Operating System Interface — a family of IEEE standards that define a portable API for UNIX-like operating systems. Enables application code to be compiled and run on any POSIX-compliant system (Linux, macOS, various UNIX variants) with minimal modification.	PDF

6. System Calls and Mechanisms

Term	Definition	Source
System Call	A programmatic request made by a user-space process to the operating system kernel for a privileged operation. The primary mechanism by which applications interact with the OS, crossing the boundary from user mode to kernel mode in a controlled manner.	Both
System Call Interface	The programming-language-level API (typically in C/C++) through which application programs invoke system calls. Abstracts the low-level details of trap instructions and register conventions.	Both
System Call Number	A unique integer assigned to each system call. When a system call is invoked, the number is placed in a CPU register so the kernel's dispatch table can look up and execute the corresponding kernel function.	PDF
System Call Table (Dispatch Table)	A kernel data structure that maps system call numbers to their corresponding kernel functions. The system call handler uses this table to route each request to the correct implementation.	PDF
Trap Instruction	A special CPU instruction (e.g., SYSCALL on x86-64, SVC on ARM) that atomically switches the processor from user mode to kernel mode and transfers control to the kernel's system call handler.	PDF
File Descriptor	A small non-negative integer returned by system calls such as open() that identifies an open file within a process. Used by subsequent read(), write(), and close() calls to refer to the file.	PDF
File Operations	A category of system calls for creating, opening, reading, writing, closing, and deleting files, as well as managing file attributes and permissions. Examples: open(), read(), write(), close(), unlink().	Both
Process Control	A category of system calls for creating, terminating, and waiting for processes, as well as loading and executing programs. Examples: fork(),	Both

	exec(), exit(), wait() (UNIX); CreateProcess(), ExitProcess() (Windows).	
Device Manipulation	A category of system calls for requesting and releasing devices and reading from or writing to them. Examples: ioctl(), read(), write() (UNIX); DeviceIoControl() (Windows).	Both
Information Maintenance	A category of system calls for getting and setting system data such as the current time, date, or process attributes. Examples: getpid(), time() (UNIX); GetCurrentProcessId(), GetSystemTime() (Windows).	Both
Communication (System Calls)	A category of system calls for creating communication connections and transferring data between processes. Examples: pipe(), socket(), send(), recv() (UNIX); CreatePipe(), CreateFileMapping() (Windows).	Both
Protection (System Calls)	A category of system calls for setting and getting file permissions and controlling access to resources. Examples: chmod(), chown() (UNIX); SetFileSecurity() (Windows).	Both
Interrupt	A hardware or software signal that causes the CPU to suspend its current activity and transfer control to an interrupt handler in the kernel. Interrupts are one mechanism (alongside system calls and exceptions) that triggers a switch from user mode to kernel mode.	PDF
Exception (Hardware)	An event generated by the CPU when it detects an error condition during instruction execution (e.g., division by zero, invalid memory access). Like interrupts, exceptions cause a switch to kernel mode so the OS can handle the fault.	PDF
Device Driver	Kernel-mode software that provides an interface between the operating system and a specific hardware device. Translates generic OS I/O requests into device-specific commands. A faulty driver is a common cause of kernel-mode crashes.	Both