# OS

* Purpose: Hardware Abstraction; Resource Management (Time & Space Multiplexing)

## User Mode VS Kernel Mode

* *User Mode*: cannot issue privileged instructions; can access parts of memory allowed by kernel mode code
* *Kernel Mode*: can issue all instructions; access all memory
* *Privileged*: instructions/memory locations that can interfere with other processes

## I/O Devices

* Device → Controller → Driver
* Accessed via system calls (I/O calls are privileged instructions)
* Can be done using:
* Busy Waiting: Driver sits in tight loop; poll device (see if done/busy)
* Interrupts: Device sends interrupt to driver when done
* Direct Memory Access (DMA): chip controls flow of bits between memory and controller without CPU intervention; chip causes interrupt when done

## System Calls

* *System Call*: allows user mode program/process to request a service from OS kernel
* switches user→kernel mode; significant performance overhead
* Process, File, Directory Management; kill process, get time

1. Put the system call number in a place where OS expects it (register)
2. Execute a TRAP instruction to switch from user mode to kernel mode and start execution at a fixed address within the kernel
3. The kernel code that starts following the TRAP examines the system-call number and dispatches to the correct system-call handle
4. System call handler runs
5. Once it has completed its work, control may be returned to the user program
6. The system call may block the caller, then the OS will look around to see if some other process can be run next

# Inter-Process Communication

## Process

* *Process*: abstraction of a running program; encapsulates all resources/information needed to run a program

|  | 1. Process blocks for input 2. Scheduler picks another process 3. Scheduler picks this process 4. Input becomes available | *Running*: using CPU  *Ready*: runnable, but temporarily stopped  *Blocked*: unable to run until external event happens |
| --- | --- | --- |

## Process Table

* linked list of PCBs (Process Control Blocks)
* *PCB*: contains state required to resume a process
* lCurrent process runs on RAM
* Not-currently-running process: info stored in process table
* address space, process table entry (references PCB), process state

|  |
| --- |

## Threads

* *Thread*: sequential execution stream within the process; entities scheduled for execution by CPU
* PRO: parallel; shared address space; reduced overhead (easier create, destroy, switch, communicate); 🠕 I/O
* CON: not help performance when CPU bound (more threads than CPU cores)

| Per Process Items (shared among threads) | Per Thread |
| --- | --- |
| * Address space * Global Variables * Open files * Child processes * Pending alarms * Signals and signal handlers * Accounting information | * Program counter * Registers * Stack * State   Threads can read, write or delete other thread’s stacks |

## Multiprogramming

* allows multiple programs to share CPU and run in (pseudo)parallel
* but CPU can only run one program at a time – OS gives illusion of parallelism by switching between processes quickly
* processes are sequential but execution is interleaved
* programs cannot rely on timing assumptions

## Race Conditions

* when multiple processes/threads read/write from common storage

|  | Avoiding Race Conditions   * *Mutual Exclusion*: ensure that 1 process exists in the critical region at any given time * *Critical Region (CR)*: part of program where shared resource/data is accessed * Conditions to achieve good solution  1. No two processes may be simultaneously inside their CRs 2. No assumptions can be made about speed and number of CPUs 3. No process running outside its CR can block other processes 4. No process should have to wait forever to enter CR |
| --- | --- |

## Deadlocks

4 Conditions for deadlock

1. Mutual Exclusion: each resource can only be assigned to 1 process at a time
2. Hold-and-Wait: process currently holding resources can request new resources
3. No-Preemption: resources previously granted cannot be forcibly taken away
4. Circular Wait: circular list of 2 or more processes; each waiting for resource held by next member of chain

Dealing with Deadlocks

1. Ignore the problem
2. Detection (Resource allocation graph and graph algorithms) and recovery
3. Avoid by careful resource allocation
4. Prevention

# Process Scheduling

* *Scheduling Algorithm*: determines which process to run next
* General Scheduling Goals
* *Fairness*: giving each process a fair share of the CPU
* *Policy Enforcement*: seeing that stated policy is carried out
* *Balance*: keeping all parts of the system busy
* When to schedule: process created/exits/blocks, I/O interrupt, clock interrupt

## Preemptive VS Non-preemptive Scheduling

* *Preemptive*: pick a process and let it run for a quantum (maximum fixed amount of time); clock interrupt at the end of time interval to give control to CPU back to scheduler
* *Non-preemptive*: pick process and let it run for as long as it wants until blocks/voluntarily releases to CPU
* Context Switching: happens every time OS switches process; expensive

## Scheduling Environments

| Batch | Interactive | Real Time |
| --- | --- | --- |
| -Don’t require user interaction  -A set of (potentially periodic) jobs that need to run | -Interactive users expect fast response times/responsiveness from the system | -Need to meet deadlines  -Not really focused in this subject |
| Can be Preemptive or Non-Preemptive | Preemptive  -by interrupting the process after a fixed time, it ensures that every process will produce a fast response  -rather than having to wait, a very long time for one process to finish before moving onto the next one, which may lead to the user having to wait and thus not achieving fast interactivity | Can be Preemptive or Non-Preemptive  -possibly schedule the process with fastest deadline first |
| -Throughput: maximise jobs per hour  -Turnaround time: minimise time between submission and termination  -CPU utilisation: keep CPU busy all the time | -Response Time: respond to requests quickly  -Proportionality: meet users’ expectations | -Meeting deadlines: avoid losing data  -Predictability: avoid quality degradation in multimedia systems |
| -[Non-Pre] First-Come First Served (FCFS)  -[Non-Pre] Shortest Job First (SJF)  -[Pre] Shortest Remaining Time Next (SRT) | -[Pre] Round Robin | -[Pre] Priority Scheduling |

# Memory Management

* PRO: support multiprogramming; security (isolation between process and OS memory); enable running processes with large memory requirements
* *Memory Manager*: manages (part of) memory hierarchy; keep track of which parts of memory are in use; allocate when needed – deallocate when done
* *Swapping*: bringing in each process in its entirety, running it for a while

# Cryptography

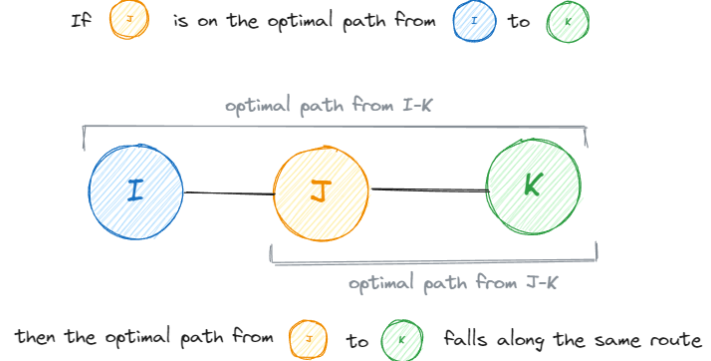
|  | *Cryptography*: encrypt plaintext to ciphertext such that only authorised individuals can decrypt it  relies on keys being secret (not algorithm)  based on mathematically hard problems (RSA: factoring product of 2 large primes; EIGamal: solving discrete logs; AES: substitution-permutation network) |
| --- | --- |

## CIA

* *Confidentiality*: only sender and intended receiver should understand contents of transmitted message
* *Authentication*: establish identities of one or both endpoints
* *Integrity*: ensure that messages are not altered

## Symmetric Cryptography

* Same key used for encryption and decryption
* *Advanced Encryption Standard (AES)*
* Break data into blocks and encrypt each block
* Modes of operation: ECB, CBC – determines how each block is treated/linked
* *Electronic Codebook (ECB)*

**

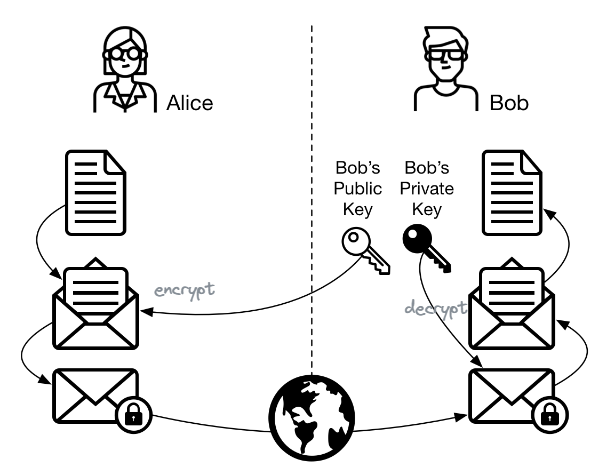
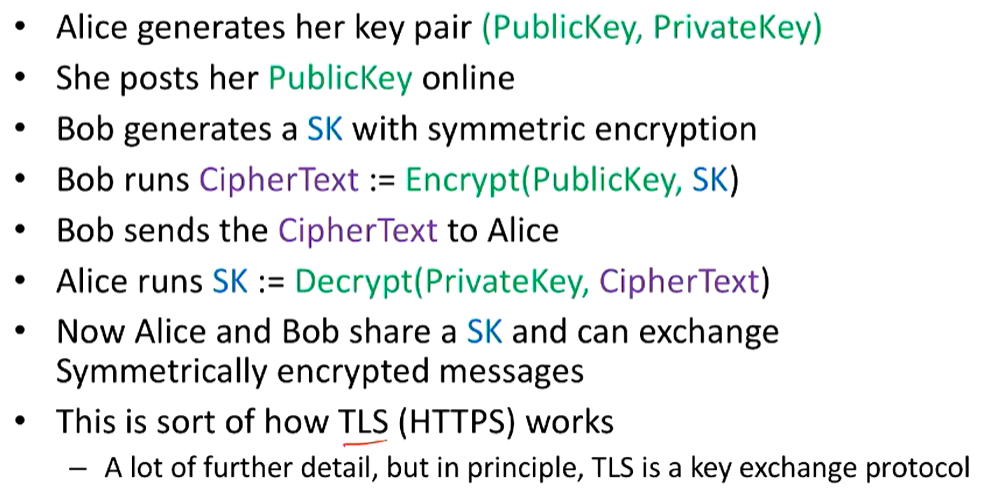
|  | 1. plaintext divided into fixed-sized blocks 2. blocks individually and independently encrypted using encryption key 3. each block decrypted individually  * PRO: simple & parallelizable; no race conditions * CON: deterministic (identical blocks produce identical ciphertext); vulnerable to plaintext attacks; limited diffusion (changes to 1 block only affect that block) * should never be used |
| --- | --- |

* *Cipher Block Chaining (CBC)*

|  | 1. plaintext divided into fixed-sized blocks 2. create random Initialisation Vector (IV) 3. first block XOR-ed with IV to create ciphertext 4. subsequent ciphertexts created by XOR-ing with previous cipher block  * PRO: fixed problems with ECB; decryption can be parallel; loss/corruption of block/IV affects at most 2 blocks * CON: encryption must be done sequentially |
| --- | --- |

## Symmetric Cryptography

* Two different keys for encryption and decryption
* CONS: slower, not suitable for large amounts of data/multiple blocks



* Digital Signatures

|  | * private key for encryption; public key for authentication/verification * provides non-repudiation (cannot deny that Bob didn’t sign the document) * integrity (no tampering) * authentication/verification |
| --- | --- |

* Hash Digest

|  | * For large documents instead of digital signatures * Hash Function * lossy compression function * should look random * collision resistant * resulting has has constant length      * diff with Encryption: does not need key; one-way |
| --- | --- |

* Message Authentication Code (MAC)

|  | * Provides Integrity * Sender use shared secret authentication key (s) and message to be verified (m) to generate a tap (t) using a MAC function (hash function) * Append tap with message to be sent to receiver * Receiver regenerates the tap (t’) using s and m and compares to tap sent |
| --- | --- |

* Authenticated Encryption – Integrity + Confidentiality

|  |  |
| --- | --- |

* Diffie Hellman Key Exchange – Sharing Secret Keys

|  |  |
| --- | --- |

* Public Key Infrastructure (Certificates) – Authentication
* Certificate: bind public key to identity of an entity (person, organisation, website etc.); signed by trusted entities (Certificate Authorities: explicitly trusted; predefined in OS; root cert is self-signed)

|  |  |
| --- | --- |

# Network Protocols and Service Models

|  | * Each layer * offers services to layers above it * uses service from layers below it * Inter-layer exchanges are conducted according to a protocol * *Service*: set of primitives/operations that a layer provides * *Protocol*: rules that govern the format and meaning of packets exchanges by peers within a layer |
| --- | --- |
|  | TCP/IP is what happens on the internet; OSI is idealised model |

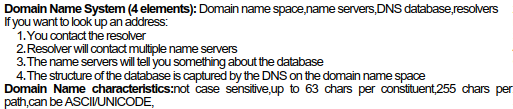
# HTTP

|  |  |
| --- | --- |

* Non-Persistent: requires 2 response times (initiate TCP; initial HTTP request) per object + file transmission time; OS overhead for each TCP connection
* Persistent: server leaves connection open after sending response
* Idempotent: multiple identical requests have same effect
* Safe: only for information retrieval, should not change state

| Codes&Headers | Request & Response |
| --- | --- |
|  |  |

# DNS

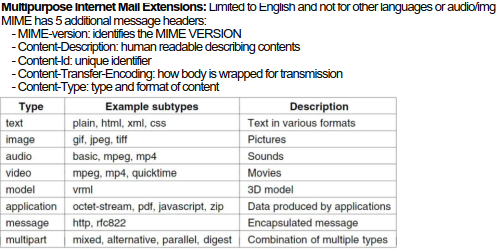
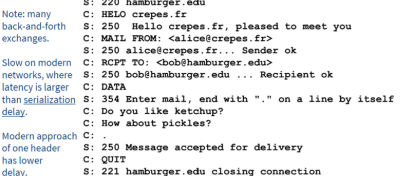


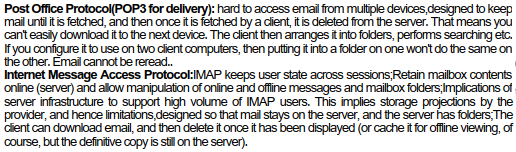
| Resource Records | Name Servers |
| --- | --- |
|  |  |

* Root Name Server: 13 globally; authoritative cluster for enquiries
* Top-level Domain Servers: (com, edu, gov, int, mil, net, org, …)
* Authoritative Servers: maintained by organisation/service provider; takes domain and subdomain→returns correct IP address
* Local DNS Server: each ISP/company/university; returns cached value if exists; acts as proxy, forwards requests up hierarchy

# SMTP

|  | * Direct, persistent TCP connection * 7-bit ASCII * all message’s objects is in 1 message * very chatty protocols * multiple transactions * unlike HTTPS (big pushes) |
| --- | --- |





# TCP

* Transport layer: tidy up end-to-end (lost/duplicated/corrupted packets)
* stream-oriented (segmenting into IP datagrams), reliable (dropped/duplicated) and in-order protocol
* *full duplex*: data in both directions, simultaneously
* *end-to-end*: exact pairs of senders and receivers
* *byte streams*: not message streams; message boundaries not preserved (no way of knowing sender has ended message)
* *buffer capable*: TCP entity can choose to buffer prior to sending or not; buffering reduces overhead (🠗headers) but 🠕delay
* *segments*: 20-60 byte header + zero or more data bytes (TCP decides size of segments)
* 5-tuple address: sender IP, receiver IP, sender port#, receiver port#, protocol

| Primitives | Sockets |
| --- | --- |
|  |  |
| Header | |
|  |  |

| 3 way handshake | SEQ and ACK incrementing | 4 way termination |
| --- | --- | --- |
|  |  |  |

Sliding Window

|  |  |
| --- | --- |

Deadlock

* Full Window: sender window full and doesn’t receive ACKs; receiver window full, doesn’t receive packets
* Lost ACK: receiver waiting for a packet that will never be retransmitted

| Solution 1: Send a Window Update | Solution 2: Zero Window Probe |
| --- | --- |
| data sent to application→window moves→window update |  |

|  |  |
| --- | --- |

|  |  |
| --- | --- |
|  |  |