Lecture 1 – Processes and Threads

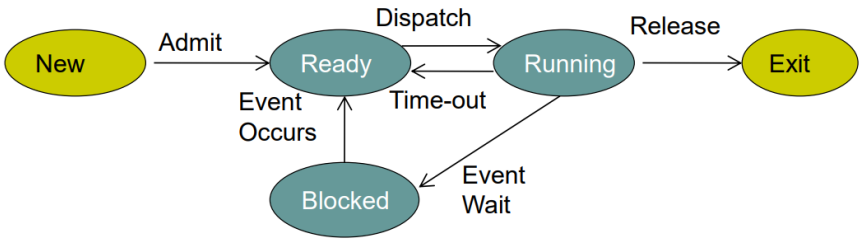
Coherency – cache management

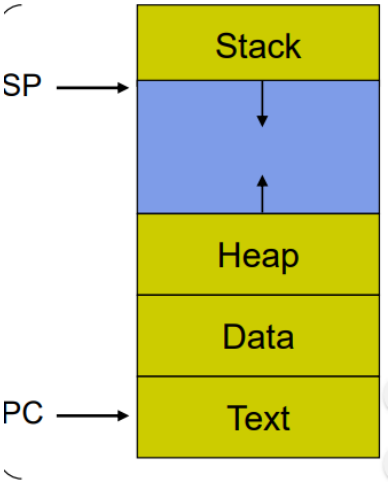
Interrupts – allows device controllers to signal the CPU that some event has occurred. Signal error (system call)

OS is event-driven program.

Process – unit of work

Data structure PCB contains all info about process. PCB(Process Control Block) includes : process state, program counter, CPU registers, CPU scheduling, memory management, IO info




State queues – collection of queues that represent state of all processes in system. One queue for each state ex – ready, waiting

Context switch – switch CPU to another process, saving state of the old process and loading saved state for new process

Process are containers in which threads are executed. One process can have several threads, while a thread can only have one process. Processes are static, threads are dynamic.

If there is no data sharing then processes are independent, else the process is cooperating and must synchronize their actions with others.

Lecture 2 – System Calls, Synchronization

Race condition – outcome depends on the order in which accesses take place

Local variable not shared, Global variable, static/dynamic/head objects shared.

Critical Section Requirements – 1) Mutual Exclusion – if one thread in CS, no other thread is

2) Progress – if no thread in CS, other thread should be able to enter in definite time

3) Bounded waiting (starvation) – if some thread in CS, there is a limit other threads can enter CS before the waiting thread is granted access

4) Performance – overhead of entering and exiting CS is small

# Peterson's Solution

```
#define FALSE  0
#define TRUE   1
#define N      2                        /* number of processes */

int turn;                               /* whose turn is it? */
int interested[N];                      /* all values initially 0 (FALSE) */

void enter_region(int process);         /* process is 0 or 1 */
{
      int other;                        /* number of the other process */

      other = 1 – process;              /* the opposite of process */
      interested[process] = TRUE;       /* show that you are interested */
      turn = process;                   /* set flag */
      while (turn == process && interested[other] == TRUE) /* null statement */ ;
}

void leave_region(int process)          /* process: who is leaving */
{
      interested[process] = FALSE;      /* indicate departure from critical region */
}
```

## Peterson's solution for achieving mutual exclusion.

More solutions for CS – 1) Locks – primitive, minimal semantics

2) Semaphores – easy to understand, hard to code

3) Monitors – high-level, ideally language supported

4) Messages – model for communication and synchronization

To build these higher level abstractions we need help from atomic instructions.

**LOCKS**

Atomic instructions: Test-and-Set

```
boolean test_and_set(boolean *lock)
{
      boolean old = *lock;
      *lock = True;
      return old;
}
```

```
boolean lock;

void acquire(boolean *lock) {
      while(test_and_set(lock));
}

void release(boolean *lock) {
      *lock = false;
}
```

This is a spinlock. Uses busy waiting – thread continually executes while loop in acquire(), consumes CPU cycles

Lecture 3 – Synchronization and Scheduling

**Semaphores**

- contains atomic operation wait (also called P or decrement): decrement variable and block until semaphore is free
- Atomic operation signal (also called V or increment): increment variable and unblock waiting thread if there is any
    - Mutex Semaphore – single access to resource, guarantees mutual exclusion to CS
    - Counting Semaphore – multiple threads can pass, max number of threads determined bu semaphore's initial count

## Reader's and writers operation:

```
//number of readers
int readcount = 0;
//mutual exclusion to readcount
Semaphore mutex = 1;
//exclusive writer or reading
Semaphore w_or_r = 1;

Writer {
   wait(w_or_r);  //lock out others
   Write;
   signal(w_or_r); //up for grabs
}
```

```
Reader {
   wait(mutex); //lock readcount
   // one more reader
   readcount += 1;
   // is this the first reader?
   if(readcount == 1)
        //synch w/ writers
        wait(w_or_r);
   //unlock readcount
   signal(mutex);
   Read;
   wait(mutex); //lock readcount
   readcount -= 1;
   if(readcount == 0)
        signal(w_or_r);
   signal(mutex);
}
```

**Monitors** – only one process at a time can be active in the monitor. Local data accessed only by monitor's procedures. Process enters monitor by invoking 1 of its procedures. Other processes that attempt to enter monitor are blocked.

- Hoare monitor – signal() immediately switches from caller to waiting thread, another queue for signaler if it wasn't done using the monitor
- Brinch Hansen – signaler must exit monitor immediately. I.e signal() is always last statement in monitor procedure
- Mesa monitor – signal() places waiter on the ready queue, but signaler continues inside monitor

Scheduling Goals

- All systems: Fairness – each process receives fair share of CPU
- Batch Systems: Throughput – maximize jobs completed per hour; Turnaround time – minimize time between submission and completion
- Interactive System: Response time – minimize time between receiving request and starting to produce output
- Real-time systems: Meet deadlines

Types of Scheduling

- Non-preemptive scheduling: once CPU has been allocated a process, it keeps the CPU until it is terminated; suitable for batch scheduling
- Preemptive scheduling: CPUS can be taken from running process and allocated to another; needed in integrative or real-time systems

Lecture 4 – Scheduling

## Dining Philosophers Problem (4)

```
. . .

void put_forks(i)                    /* i: philosopher number, from 0 to N–1 */
{
        down(&mutex);                /* enter critical region */
        state[i] = THINKING;         /* philosopher has finished eating */
        test(LEFT);                  /* see if left neighbor can now eat */
        test(RIGHT);                 /* see if right neighbor can now eat */
        up(&mutex);                  /* exit critical region */
}

void test(i) /* i: philosopher number, from 0 to N–1 */
{
        if (state[i] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING) {
                state[i] = EATING;
                up(&s[i]);
        }
}
```

A solution to the dining philosophers problem.

Scheduling Algorithms

- First Come First Sever (FCFS) – FIFO. Average waiting time quite long
- Shortest Job First (SJF) – Choose the process with the shortest processing time. Probably optimal average wait time

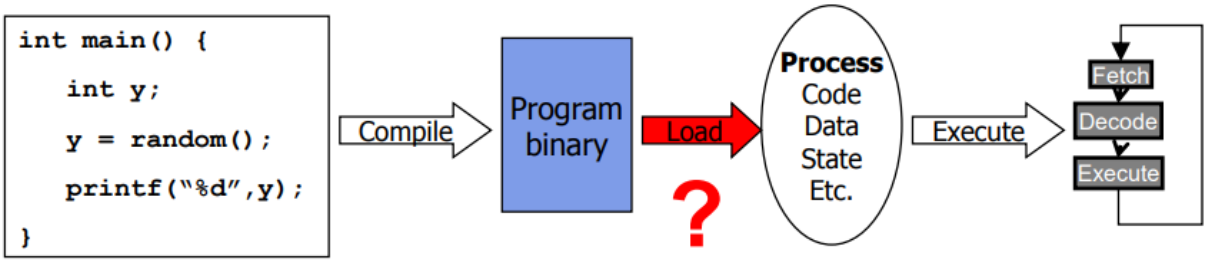
- Round Robin (RR) – Circular. Each process is allowed to run for time quantum q before being pre-empted and put back on queue
- Priority Scheduling – Highest priority job is selected from ready queue

Real systems implement a combination of multi-level queue scheduling, which is Round Robin (RR) with priorities.

## Lecture 5 – Memory Management

### Requirements

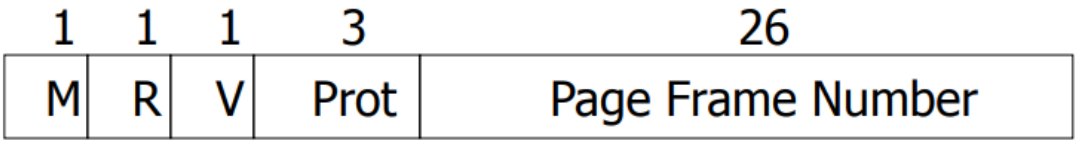
- Relocation – Programmers do not know what physical memory will be available when their programs are run; some type of address translation
- Logical Organization – Machine accesses memory addresses as one-dimensional array of bytes
- Protection – processes memory protected from unwanted access by other processes
- Sharing - Sometimes processes need to be able to access the same memory
- Physical Organization – Memory and Disk form two-level hierarchy, flow of information between levels must be managed




### Memory Management Techniques

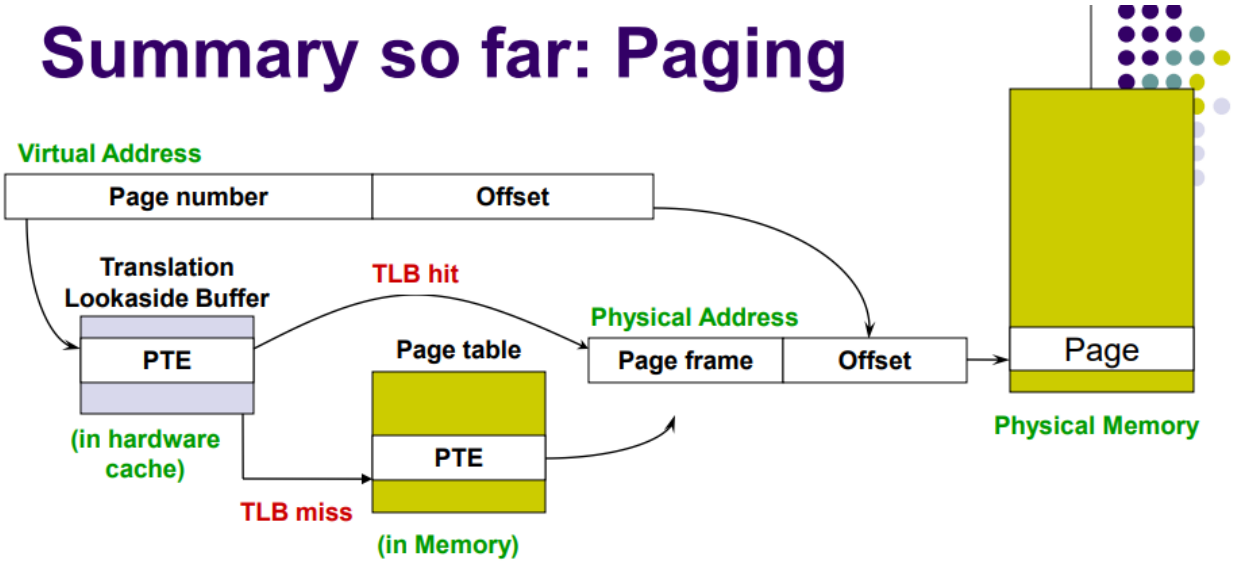
- Fixed Partitioning: Divide memory into regions with fixed boundaries
    - Internal Fragmentation: memory is wasted if process is smaller than the partition
    - Overlays: programmer must deal with programs that are larger than the partition
    - Assign process to the smallest available partition
- Dynamic Partitioning: Partitions vary in length and number over time
    - External fragmentation: some blocks may be too small for any processes
    - Compaction: may mover processes around to create larger chunks of free space
- Paging: partition memory into equal, fixed chunks, called page frames or frames; divide processes' memory into chunks of the same size called pages
    - Paging addresses the issue of contiguous blocks of physical memory
    - External fragmentation is eliminated; internal fragmentation is at most a part of one page per process

# Page Table Entries (PTE)




- Page table entries (PTEs) control mapping
  - Modify bit (M) says whether or not page has been written
    - Set when a write to a page occurs
  - Reference bit (R) says whether page has been accessed
    - Set when a read or write to the page occurs
  - Valid bit (V) says whether PTE can be used
    - Checked on each use of virtual address
  - Protection bits specify what operations are allowed on page
    - Read/write/execute
  - Page frame number (PFN) determines physical page
  - Not all bits are provided by all architectures

# Summary so far: Paging




Tracking Memory Allocation

- Bitmaps: 1 bit per unit. 0 is free, 1 is allocated
- Free lists: maintain list of allocated and free segments
- Implicit list: each block has header that records size and status

Placement Algorithms

- First-fit: choose first block that is large enough; search can start at beginning, or where previous search ended (called next-fit)
    - May leave small fragments near start of memory that will be searched repeatedly
- Best-fit: choose the block that is closest in size to the request
    - Left over fragments tend to be small
- Worst-fit: choose the largest block
    - Not as good as best-fit or first-fit
- Quick-fit: keep multiple free lists for common block sizes
    - Great for fast allocation

Lecture 6 – Virtual Memory & Page replacement

TLB's hide the cost for frequently-used pages.

Policy Decisions

- Fetch Policy – when to fetch a page
    - Demand Paging – when the process needs the page
    - Prepaging – predict future page use at time of current fault
- Placement Policy – where to put the page
- Replacement Policy – what page to evict to make room
    - Belady's algo – optimal page replacement because lowest fault rate
    - Random replacement – lower bound
    - First-in-first-out (FIFO)
        - Fault rate might increase when algorithm given more memory
    - Least-recently-used (LRU)
        - Time stamp every reference, evict page with oldest time stamp
    - Least-frequently-used
    - Most-frequently-used
    - Not-recently-used(NRU)
        - Class 1: Not referenced, not modified
        - Class 2: Note referenced, modified
        - Class 3: Referenced, not modified
        - Class 4: Referenced, modified
        - Remove page at random from lowest-numbered class that's not empty
    - Second-chance(clock): evict the oldest page that has not been used
        - Sweep through pages; ref bit off then it has been recently used which means evict page; ref bit is on then turn it off and go to next page
    - Page Fault Frequency (PFF): variable space algorithm that uses ad-hoc approach

- If the fault rate is above high threshold, give it more memory
- If the fault rate is below low threshold, take away memory

Fixed vs Variable Space

- Fixed space algorithms
  - Limit of pages it can use
  - Local replacement
- Variable space algorithms
  - Process' set of pages grows and shrinks dynamically
  - Global replacement: one process can ruin it for the rest
  - Local replacement: replacement and set size are separate

Working Set Model – page is in working set only if it was referenced in the last x references

Lecture 7 – File System

File access methods

- Sequential access – read bytes one at a time, in order ex -read/write next
- Direct access – random access given block/byte number ex- read/write n

When file is first actively used, store its attribute info in a system-wide open-file table; the index into this table is used on subsequent operations, therefore no searching

There are 2 levels of internal tables – per process table of all files that each process has open

- per process table points to an entry in the system-wide open-file table

Possible Directory Implementations – single level or acyclic-graph. Acyclic graph directories allows for shared directories. A file or subdirectory may be in 2 different directories.

Symbolic link – Directory entry contains true path to the file

Hard link – Second directory entry identical to the first

Two key issues when sharing files

- Semantics of concurrent access
- Protection – given action by given user is allowed or not

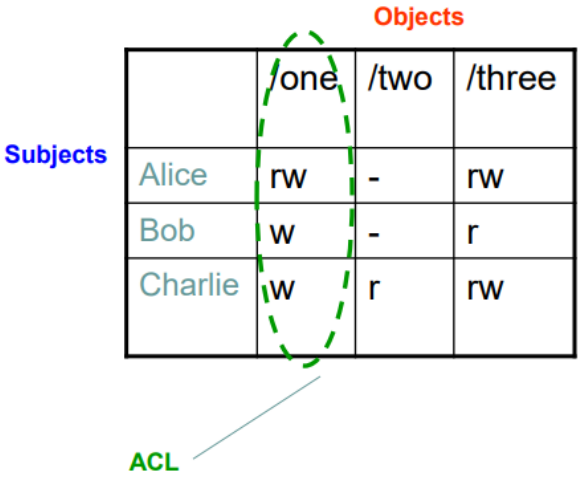
# Representing Protection

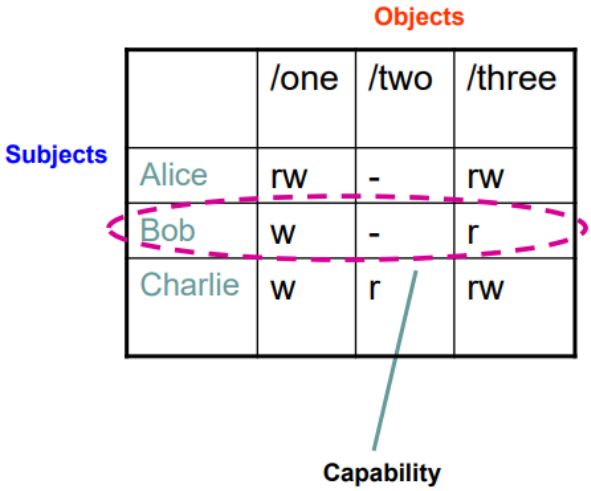
## Access Control Lists (ACL)

- For each object, maintain a list of subjects and their permitted actions

## Capabilities

- For each subject, maintain a list of objects and their permitted actions




Disk Layout Strategies

- Contiguous Allocation – one after another
- Linked or chained structure – pointer from one to another
- Indexed structure – each node has 15 block pointers. First 12 are direct block pointers. Then single, double, triple indirect.

Caching Writes

- Write behind
  - Maintain queue of uncommitted blocks; periodically flush queue to disk
- Battery backed-up RAM
  - Maintain queue in NVRAM
- Log-structured file system
  - Always write contiguously at end of previous write

Read ahead – File system predicts that the process will request next block; requests it from disk while process is computing in previous block; when process requests block it will be in cache

MMV (Memory Management Unit) – hardware device that performs memory access checks

Lecture 8 – Disk I/O

Disk request performance depends on 3 steps

- Seek: moving the disk arm to the correct cylinder. Very slowly improving, average 5-6 ms.
- Rotation: waiting for the sector to rotate under the head. Increasing but slowly, average 3 ms.
- Transfer: transferring data from surface into disk controller electronics, sending it back to host. Increasing quickly

Redundant Arrays of Inexpensive Disks (RAID): spreads the work across several disks and therefore achieves better I/O performance through parallelism.

Modern disks have logical block addressing. Logical array of blocks [0.. N]

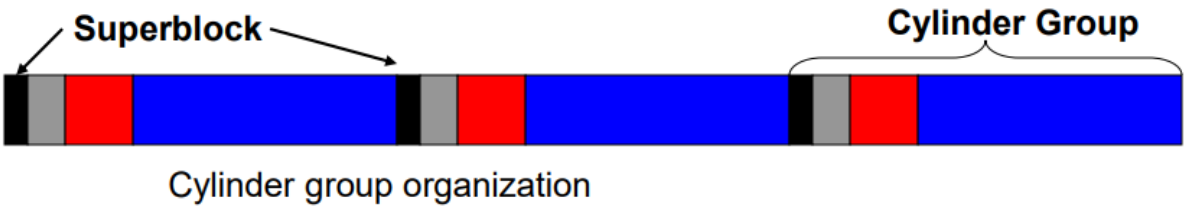
Disk Scheduling

- FCFS
- SSTF (shortest seek time first) – minimize arm movement, maximize request rate; favours middle blocks
- SCAN (elevator) – service requests in one direction and then reverses
- C-SCAN – like SCAN, but can go in one direction
- LOOK / C-LOOK – like SCAN/C-SCAN but only go as far as last request in each direction

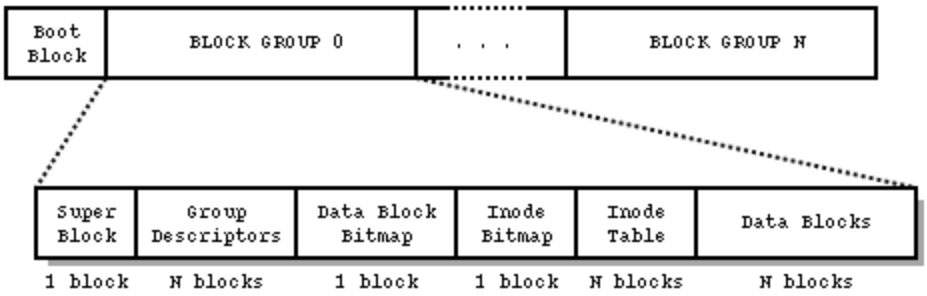
Original Unix File System


1. Deleted files randomly placed.
2. Inodes allocated far from blocks.

Fast File System (FFS)




Ext2 Filesystem




Superblock – number of blocks on disk, size of blocks, number of free blocks

Group descriptors – block bitmap, inode bitmap, inode table

Inode table – contains everything OS needs to know about file

New Technology File System (NTFS) Filesystem

- Used in windows.
- Each volume has a Master File Table (MFT)

Lecture 9 – Deadlocks

Types of Resources

- Reusable: can be used by one process at a time, released and used by another process. Ex – locks, monitors
- Consumable: Dynamically created and destroyed. Can be allocated once. Ex – signals, messages

Causes of Deadlock

- Resources are finite
- Processes wait if a resource they need in unavailable
- Resources may be held by other waiting processes

Conditions for Deadlock : First 3 necessary condition, last is sufficient

- Mutual Exclusion: only one process may use a resource at a time
- Hold and wait: process may hold allocated resources while waiting assignments of others
- No pre-emption: no resource can be forcibly removed from a process holding it
- Circular wait: closed chain of processes exists, such that each process holds at least one resource needed by the next process in the chain

Solutions

- Prevention
    - Does not break mutual exclusion
    - Break “hold and wait” – processes must request all resource at once
    - Break no-premption: forcibly remove a resource from one process and assign to another

    - Break circular wait – assign a liner ordering to resource types and require that a process holding a resource of one type can only request resources that follow in the ordering
- Avoidance
    - Do not start a process if its maximum resource requirements, together with the maximum needs of all processes already running, exceed the total system resources. Pessimistic, assumes all processes will need all their resources at the same time
    - Do not grant an individual resource request if it might lead to deadlock
    - Banker's Algorithm: Deadlock Avoidance. Processes must declare all their needs
        - Update state assuming request granted; check if new state safe; if so continue else restore old state and block process until it is safe to grant the request
- Detection and Recovery
    - Find circular waits to detect
    - Recovery
        - Drastic – kill all deadlocked processes
        - Painful – back up and restart deadlocked processes
        - Better – selectively kill deadlocked processes until the cycle is broken
        - Tricky – selectively pre-empt resources until cycle is broken
- Do Nothing

Safe states – if there is at least one sequence of process executions that does not lead to deadlock

Conflict-serializable: if a serial schedule can be obtained by swapping non-conflicting operations than the original schedule is conflict serializable.

Livelock: when a set of processes continually retry some failed operation and prevent other processes in the set from making any progress.

## Lecture 10 – Security

### Four requirements of security

- Confidentiality: prevent unauthorized release of info
    - Interception or eavesdropping – attacker gains knowledge they should not have access to
- Integrity: preventing unauthorized modification of info
    - Modification – attacker alters existing files, programs, network packets etc
- Availability: ensuring access to legitimate users
    - Theft of service – attacker installs daemon
- Authenticity: verifying the identity of the user
    - Fabrication – attacker creates counterfeit objects which appear to come from a trusted source

### Intrusion Detection

- Signature-based detection: examine network or system for pattern matching known attacks
- Anomaly detection – identify patters of "normal" behaviour over time and detect variations of those patterns

Malicious Software

- Trap doors: program contains secret entry point that allows attacker to bypass security
- Logic bombs: destructive code in legitimate program triggered by some event
- Trojan horses: apparently useful program that tricks users into running it
- Viruses: can infect other programs by copying itself into them
- Worms: program spreads via network connects

Stack & Buffer Overflow – Overflow some stack-allocated input buffer past the space allocated for the input; overwrite the return address with the address of the exploit code

Trusted Computing Base(TCB) – set of components (hardware, software, people) that you trust your secrets with

Network Attacks

- Passive: eavesdropping or monitoring communications
    - Secret/Symmetric key cryptography
    - Public/Asymmetric key cryptography – much slower, one key to decrypt and one to encrypt
    - Traffic analysis – infer information based on sender/recipient of messages, size of messages, frequency of communication etc. Solution obfuscate communication pattern
- Active: modification or tampering with communication stream
    - Replay – capture a legitimate message and re-send it later to produce unauthorized effect. Defence: "nonce"- messages include an item so they cant be reused
    - Modification of messages – alter contents of message to change effect. Defence: message digest to detect tampering
    - Masquerade – attacker pretends to be another entity or user

Digital signatures: provide a way to verify origin and integrity of message

Denial of service – attack on availability

Principle of Least privilege – figure out exactly which capabilities a program needs to run and grant it only those

Bruce Schneier: 3 waves of security attacks

- 1st wave: physical attacks on wires and hardware
- 2nd wave: syntactic attacks on crypto protocols and systems. Eg- buffer overflows, DDos
- 3rd wave: humans and computers trust information that they shouldn't

Secure Socket Layer(SSL). Typically seen web services(https://). Communication begin with handshake protocol between client and server to establish identity and set up session keys used to encrypt remainder of the transmissions.