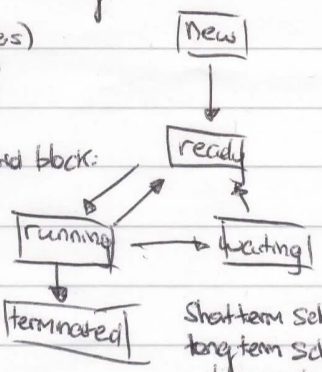
System calls low-level function for common operations
Kernel Space where Kernel executes and gives Services
user Space where user processes run

Program passive entity stored on disk as executable
Process active, loaded into memory:
- program code
- current activity (PC/registers)
- Stack (temp variables)
- data (global variables)
- heap

OS tracks process with process control block:
- program counter
- process number
- process state
- registers
- memory info
- I/O status



Simple Structure — System Progs → OS Drivers — Application Progs. — BIOS Drivers — Difficult to maintain, Little overhead Sys calls

Layered Approach
Bottom layer hardware, top layer user interface, clear interfaces between layers, easy to construct and debug, hard to define layers, inefficient with lots of system calls

Microkernel
Non-essential programs in user mode, essential in kernel mode, extending OS is easy, increased system call overhead

Modular
Extends upon microkernels, kernel modules loaded when needed, do not need to rebuild to add functionality

I/O bound does I/O more, CPU bound does computation more

Short term Scheduler — Chooses what to execute next
long term Scheduler — Chooses what to put in ready queue
dispatcher — gives control of CPU to process chosen by ST Scheduler

Job Queue — Set of all processes
Ready queue — processes ready for execution
Device queue — waiting for I/O devices

Context switch when CPU changes to another process
1 Save State of current process in process control block
2 load saved State of new process

independent process cannot affect other process
but cooperating process can
good for | information sharing | computational Speedup | modularity | convenience

Shared Memory
processes share a region of memory
only need system calls for setup
have to avoid conflicts/race conds.

Message Passing
processes send message to each other
send(m) and receive(m)
must establish a connection
direct communication (name each other)
indirect communication (mailboxes)
blocking (Sync.), nonblocking (aSync.)

data parallelism — Subsets of same data operated on in different cores
task parallelism — different tasks performed on different cores
Amdahl's law — Speedup with more cores $\leq \frac{1}{S + \frac{(1-S)}{N}}$
implicit threading — when compilers and runtime libraries control threading

thread is
basic unit of CPU execution
contains thread id, program counter, register set, stack
threads are cheaper to create, good for:
responsiveness
resource sharing
economy
scalability

threadpool is where they await for work

four instances that trigger scheduler
- running → waiting
- running → ready
- waiting → ready
- terminates

nonpreemptive — no choice
preemptive — choices

CPU utilisation — keep the CPU as busy as possible
throughput — number of processes that complete their execution per time unit
turnaround time — amount of time to execute a particular process
waiting time — amount of time a process has been in waiting queue
response time — time from request to first response

race condition when two processes write to same variable
but only one value is preserved

Starvation very low priority processes never get executed
aging gradually increase priority of waiting processes

Critical Section Problem
1. only one can be in it's critical section at any time
2. if none in critical section and some waiting, one go in
3. can only wait a limited number of times before going in

Preemptive kernel allows a process to be preempted while running in kernel mode
Nonpreemptive kernel does not allow preempting

Mutex Lock
Process must acquire lock before entering Critical Section and release when done
busy-waiting

Semaphore
Semephore is an integer
Process puts itself in wait state until can run
control access to a finite number of a given resource

(Symmetric multiprocessing
one processor does scheduling while others run code
Symmetric multiprocessing
each manages own scheduling

Deadlock occurs when two or more processes wait indefinitely for an event that can only be caused by the waiting processes

Deadlock occurs if four conditions hold:
1. if only one process can use resource at a time
2. a process holding a resource is waiting for a resource held by another
3. a resource can only be released after a process finishes its tasks
4. $\{P_0, P_1, \ldots, P_n\}$, P_0 waiting for P_1 res, P_1 for P_2 res, ..., P_n for P_0 res

Safe State if no deadlocks
unSafe State if Possible deadlocks

Priority inversion, the lower priority process is Secretly a higher priority than that assigned to it

external fragmentation if total memory space exists to satisfy request but not contiguous / internal fragmentation if allocated memory is slightly larger than requested memory