Concept

OS: special layer of software that provides application software access to hardware resources include programs: kernel, drivers, shell.
(OS provides abstractions to apps, Manage resources, achieves the above by implementing specific algorithms and techniques).

Process: A process is an execution instance of a program. More than one process can execute the same program code. contain: PC, runnin time, list o files, page table
(An instance of an executing program is a process consisting of an address space and one/more threads of control)

System call: it's a function call exposed by kernel. (mkdir(), rmdir(), chown(), chmod(), printf(), scanf(), malloc(), free(), fopen(), fclose())

stack
heap
code
static data

~~the~~ Threads: Single unique execution context: fully describes program state: PC, Registers, Execution Flags, Stack.

Address Space: the set of stack accessible addresses + state associated with them. (Programs execute in an address space that is distinct from the memory space of the physical machine)

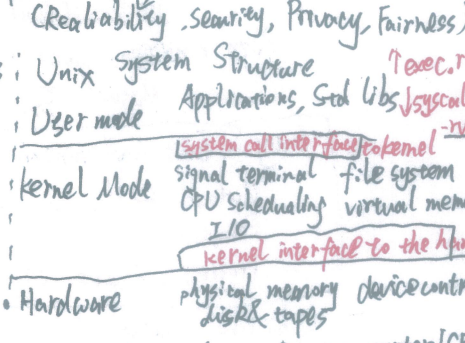
Dual mode: kerner mode (supervisor, protected) User mode: Normal programs executed.

Single and Multithreaded Process

PID_MAX
~~=~~ 32768
0x8000

(Reliability, Security, Privacy, Fairness)

Unix System Structure



↑exec, rtn, rfi
↓syscall, interrupt, exception, exit

von Neumann architecture

3 types of Mode transfer:
Syscall, Interrupt, Trap or exception
UNPROGRAMMED CONTROL TRANSFER

User space: Global variables, Local variables, Dynamically-allocated memory (mode bit = 1)
~~Code~~ Code + Constants.
Kernel space: PCB, Kernel calls with system call (mode bit = 0)

fork(): Clone: Program counter [CPU Register], Program code [File & Memory], Memory, Opened Files [kernel's internal]
Distinct: return value(0), pid, parent process, running time(0), File locks (Unchanged vs None)

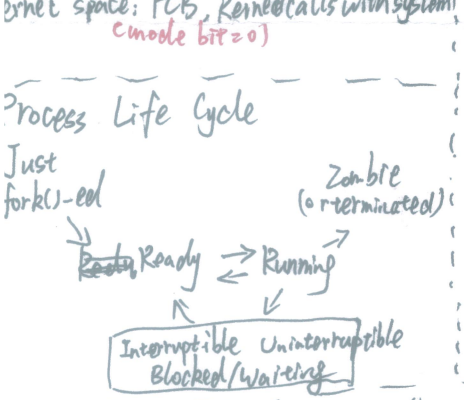
fork() + exec*() + wait() = system(): Real time >/< user time + sys time
(当前进程实际 cpu 耗时 = user time + sys time
单线程: real > cpu 耗时; 多线程 <. (Accessing Hardware)

fork() user space: copy kernel space: new pid, running time, children list, pointer to parent, return value.
Array of open files: 0 1 2 3 ...
stdin stdout stderr Files opened

exec*(): user space: old code → new code kernel space: Clear local variables and dynamically-allocated memory; Reset Global variable and code + constant.

exit(): The kernel free all the allocated memory. The list of opened files are all closed.
The kernel frees everything on the user-space memory about the process.
PID is still in kernel's process table. It's needed to allow the process that started the process to read its exit status. SIGCHLD signal handling routine
Received signal: Act and remove signal; Destory process in kernel space (process table, task-list)
Deregister signal handling routine, return PID of child.
Parent exit → children re-parent to init (pid=1)

Process Life Cycle



Context switch: Direct cost: Save and restore registers; switch address space
Indirect cost: cache, buffer cache & TLB Miss (translation lookaside buffer)

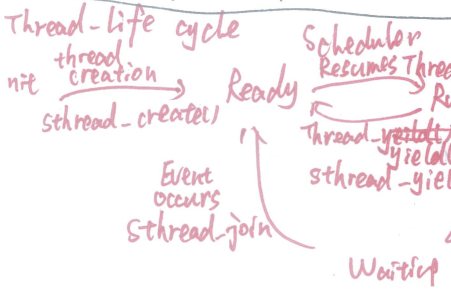
Preemptive scheduling: pros: Good for interactiveness cons: Bad for time in finishing tasks. online/offline
Algorithm Evaluation: # of context switches, Individual & average turnaround time/waiting time
AL: Shortest-Job-First (SJF), Round-robin (RR), Priority scheduling with multiple queues (Linux kernel 2.6 has priority 0-99)
Quantum
respon siveness great
↳ ordinary processes (pri=0) follow modified RR depend on nice value (-20 to 19, default = 0), remaining quantum, ...

Thread State: State shared by all threads in process/address space (Content of memory, I/O State)
State "private" to each thread (kept in TCB, CPU Registers, Execution Stack)
Parameters, temporary variables, PC

Threads encapsulate concurrency: "Active" component
Address space encapsulate protection: "Passive" part component

Shared State: Heap, Global variables, Code, File descriptor
Per-thread State: Stack + TCB (Stack info, Saved Registers, Thread Metadata)

Thread-life cycle



	Switch Overhead CPU State	Mem State	IO State	Process creation	Protection CPU	Mem	IO	Sharing overhead
Processes (1th, 1core)	low	high	high	high	yes	yes	yes	high
Threads (1core)	low (only CPU state)			low	yes		no	low
Multi-core	low (only CPU state)			low	yes		NO	low
Hyper-threading (2t/core)	Switch overhead between hardware-threads: very low.							