OS

Overview

- Architecture & Structure
- Process Management
- Memory Management
- File Management
- Protection Mechanism

Definition:

History:

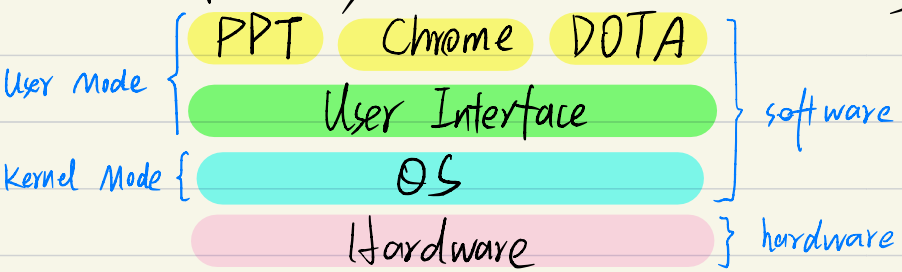
- No OS
- Mainframe (IBM 360)
- Time-sharing OS (Multics)
- Microcomputer
- Unix
- MSDOS, Apple, Windows

Motivations:

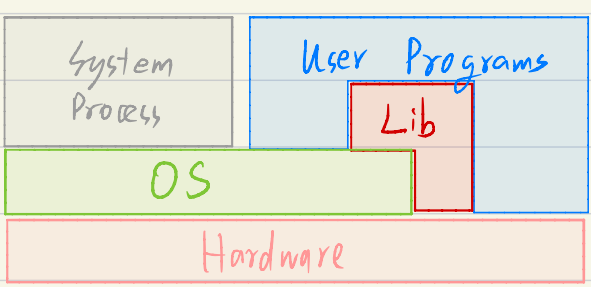
- abstraction: many different hardware config
- resource allocator: different jobs and users
- control program: avoid misuse, sharing, controler

Structures

Provide: flexibility, robustness, maintainability



OS Component



Kernel:

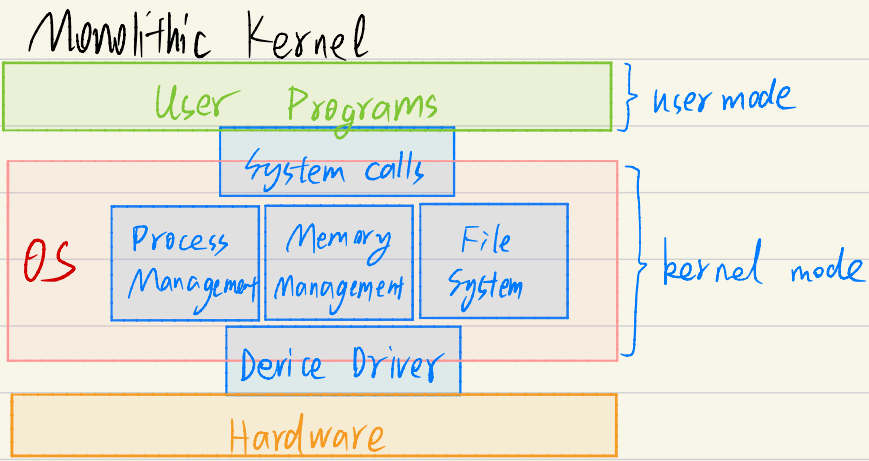
- deals with hardware issues
- provide system call interface
- special code for interrupt handler and device drivers

To write an OS:

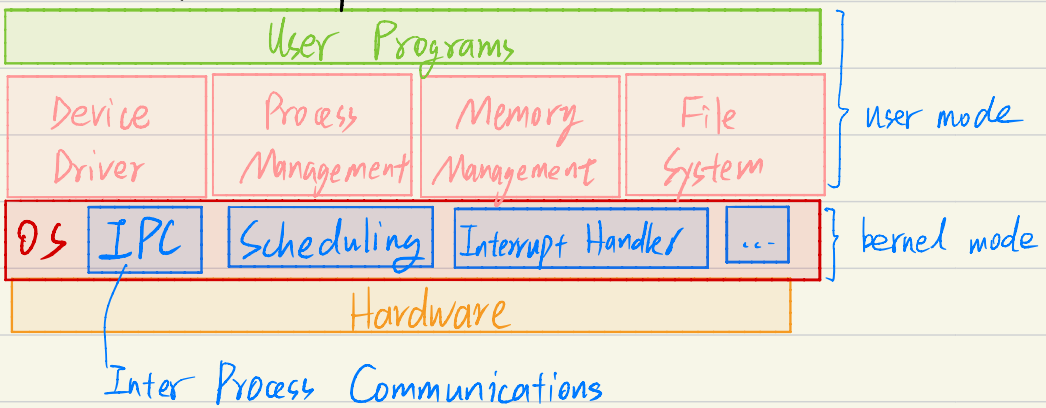
assembly / HLL: hardware dependent

Challenges: no libs; debugging; enormous; complexity

Two architecture



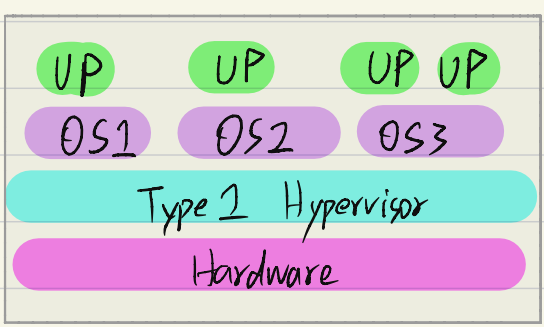
Microkernel Component

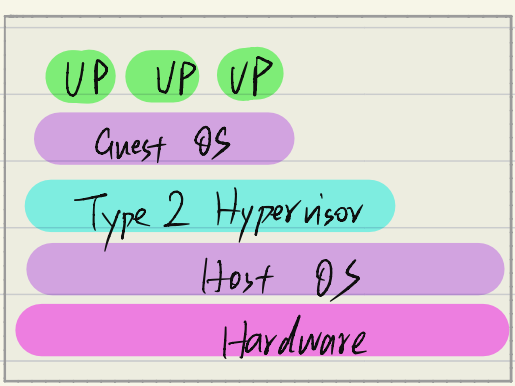


Other structure:

- Layered systems: hierarchy of layers
- Client-Server Model: variation of microkernel, have two types of process: client & server

Virtual Machine





Process Abstraction

Process: a running program 方便转换

A Process

- Memory Context
- Hardware Context
- OS context

Process Abstraction
Process Scheduling
Inter-Process Communication & Synchronization
Alternative to Process

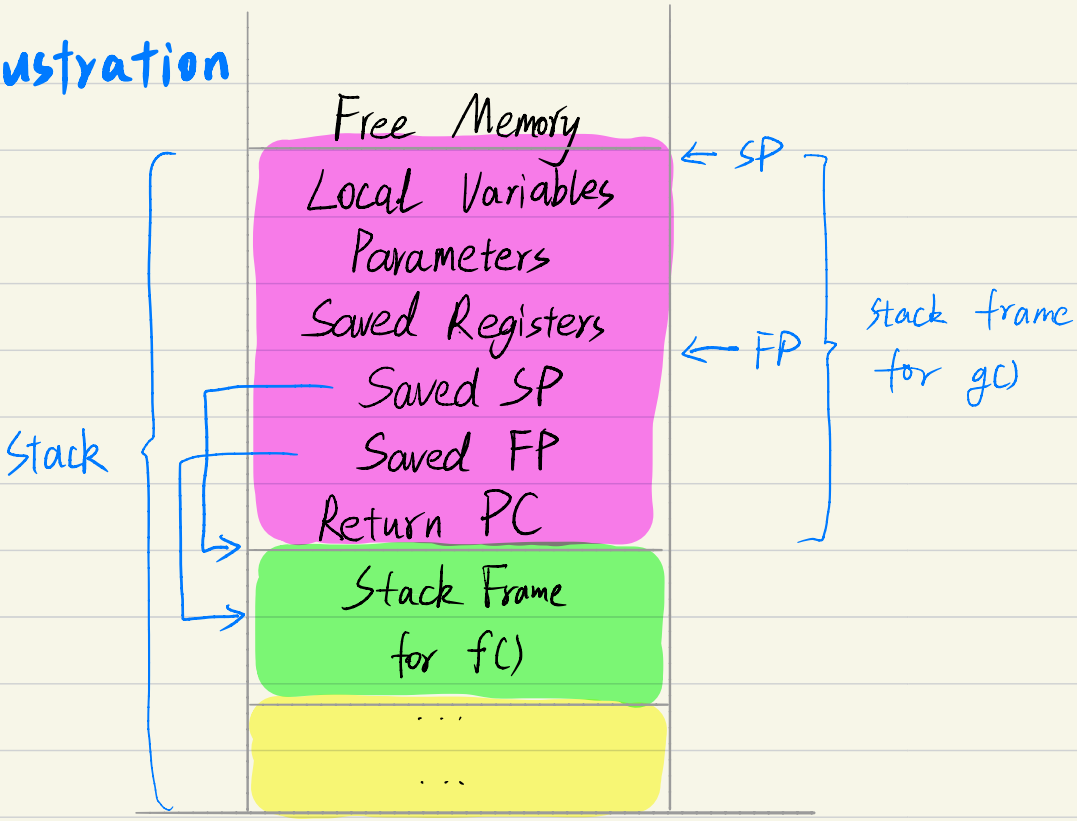
Function Call

stack memory region
stack frame: information of one functio:

- return address
- Arguments
- Storage for local variables
- others

stack pointer (SP)
frame pointer (FP)

Illustration

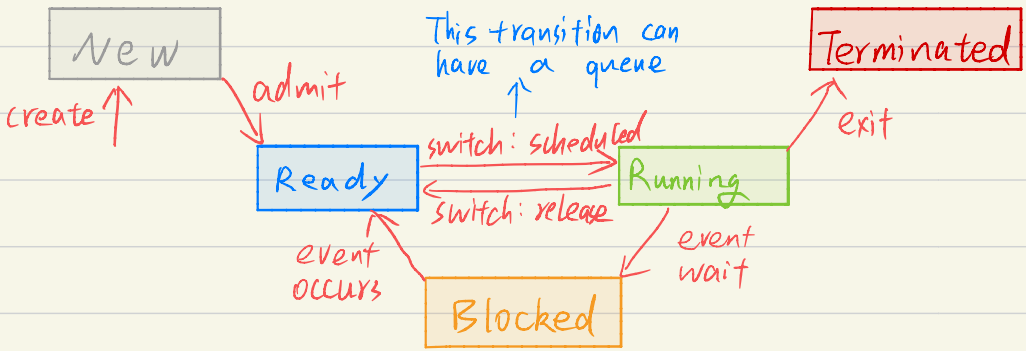


Dynamically Allocated Memory: heap memory

Process Id & Process State

PID

Generic 5-State Process Model



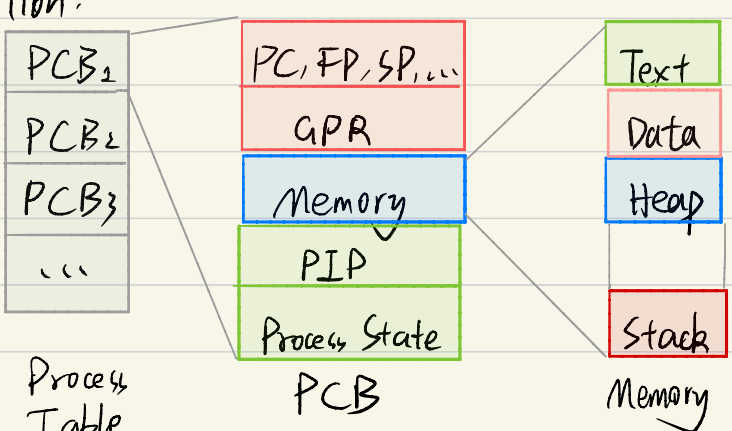
no more than 1 process can running on 1 CPU

Process Table & Process Control Block (PCB)

PCB is also Process Table Entry (PTE)

should have: Scalibility, Efficiency

Illustration:



System Calls

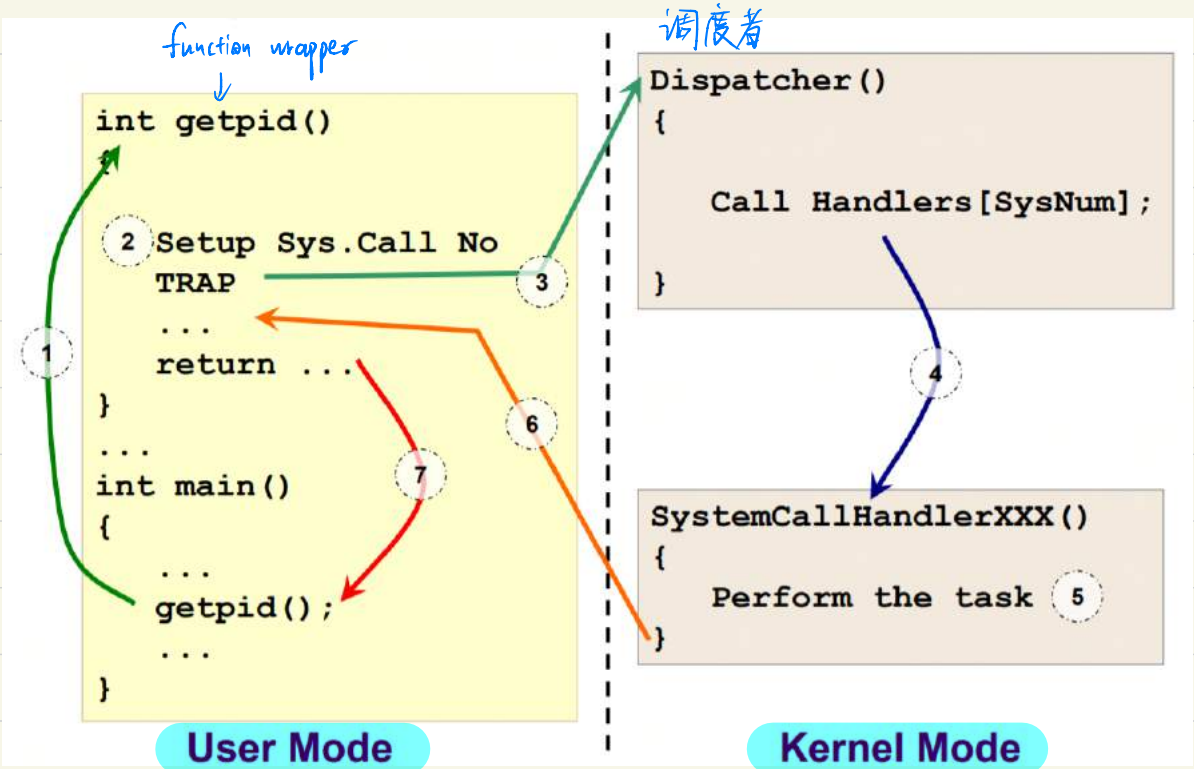
Unix : ~ 100

Win : ~ 1000

In C :
- function wrapper : same name & function
- function adapter : more user-friendly

getpid() printf()

Mechanism



Exception & Interrupt

Exception: Machine Level Instructions:

- Arithmetic Errors
- Memory Access Errors

which is **synchronous**

lead to exception handler

Interrupt: usually hardware related

- keyboard, mouse
- Timmer

which is **asynchronous**

lead to interrupt handler

Process Abstraction in Unix

Identification: PID

Information: - Process State: Running, Sleeping, Stopped, Zombie
- Parent PID
- Cumulative CPU time

`fork()` : return child PID (for parent process) OR 0 (for child process)

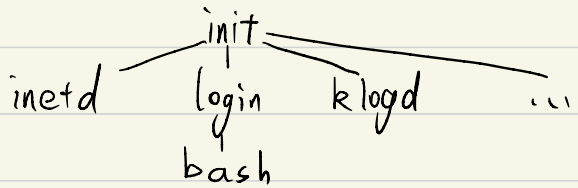
`execl()` : replace current process image with a new one

```
int execl(path*, arg[], NULL)
          ↑ new program   ↑ end of args
```

The Master Process

`init` process, usually with PID = 1

Simplified Process Tree Example:



Termination: `exit(int status)` (no return value)

by convention, status code == 0 ⇒ normal exit

!= 0 ⇒ problematic

return from main() implicitly calls exit()

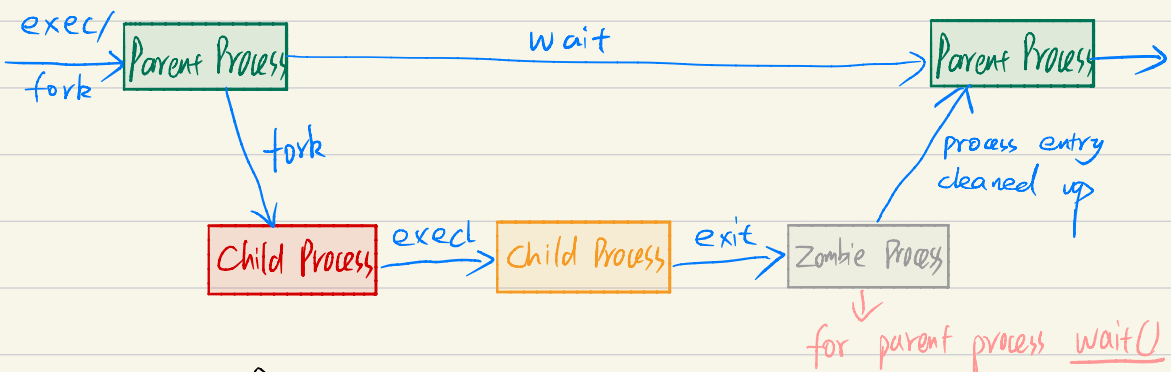
Parent process can `wait(int *status)` for child process to terminate

↑ could be NULL

Others:

`waitpid()` for a specific child process

`waitid()` for any child process to change status

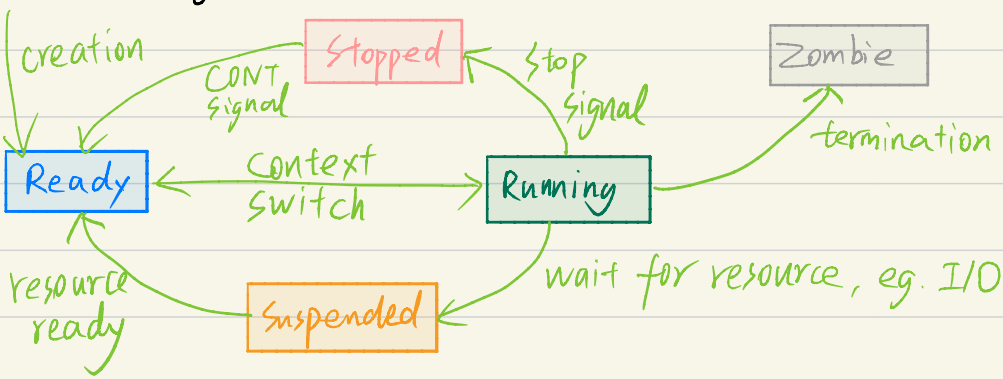


For Zombie Process:

- Parent exit first:
  init become parent,
  init wait() automatically

- Child exit first:
  become a zombie,
  wait for wait()

Process State Diagram in Unix



Implementation of fork()

1. create address space for child process
2. Allocate p' = new PID
3. Create kernel process data structure (e.g. PCB)
4. Copy kernel environment of parent process (e.g. priority)
5. Initialize child process context (PID, PPID, zero CPU time)
6. Copy memory region from parent (expensive)
7. Acquaires shared resources (open files, current dir, etc)
8. Initialize hardware context of child (reg, etc.)
9. Ready to run (add to scheduler queue)

Copy on Write is a strategy

clone() supersedes fork(), create new threads