OS

Overview

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

Definition:

History:

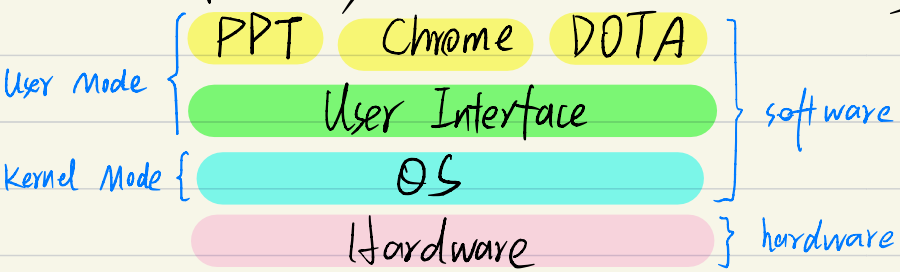
- No OS
- Mainframe (IBM 360)
- Time-sharing OS (Multics)
- Microcomputer
- Unix
- MSDOS, Apple, Windows

Motivations:

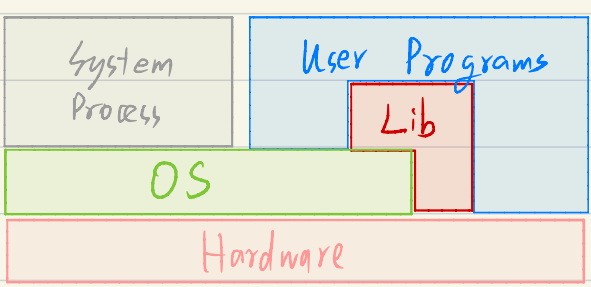
- abstraction: many different hardware config
- resource allocator: different jobs and users
- control program: avoid misuse, sharing, controler

Structures

Provide: flexibility, robustness, maintainability



OS Component



Kernel:

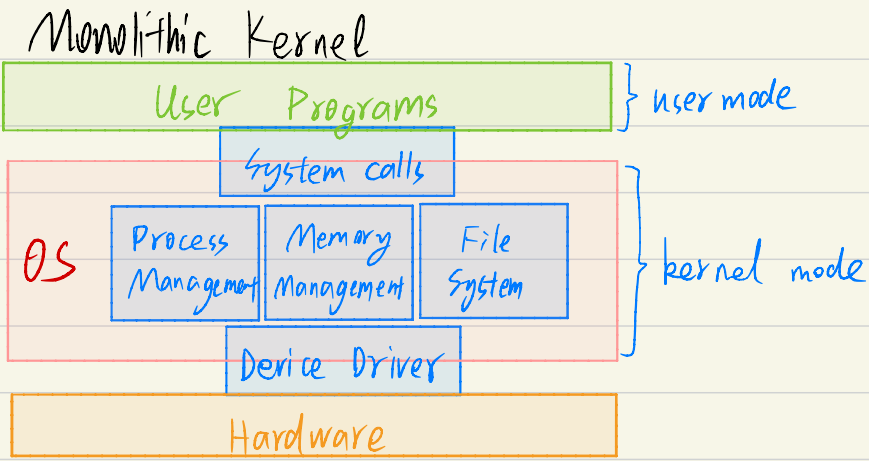
- deals with hardware issues
- provide system call interface
- Special code for interrupt handler and device drivers

To write an OS:

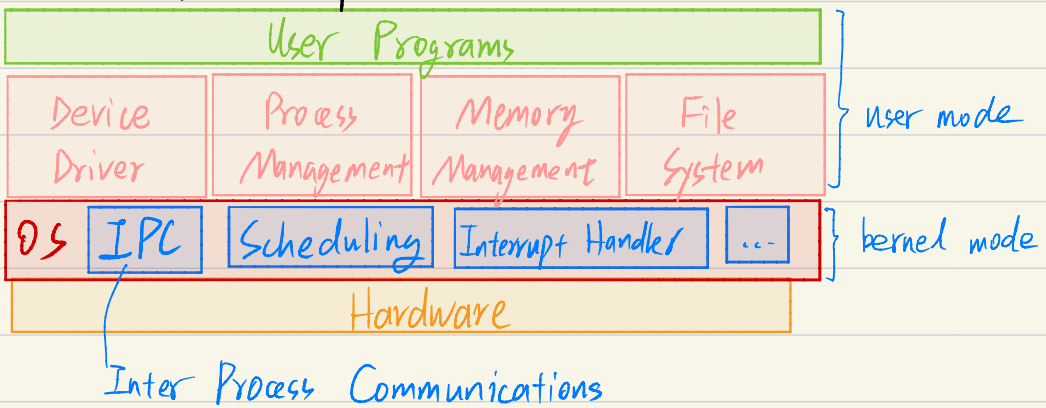
assembly / HLL: hardware dependent

Challenges: no libs; debugging; enormous; complexity

Two architecture



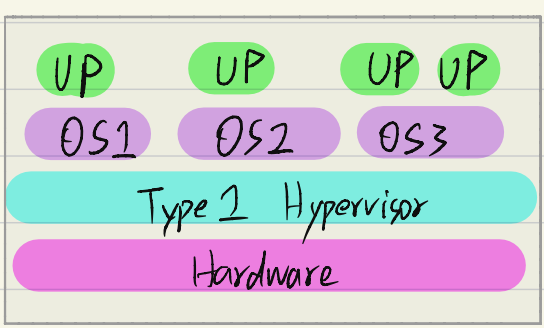
Microkernel Component

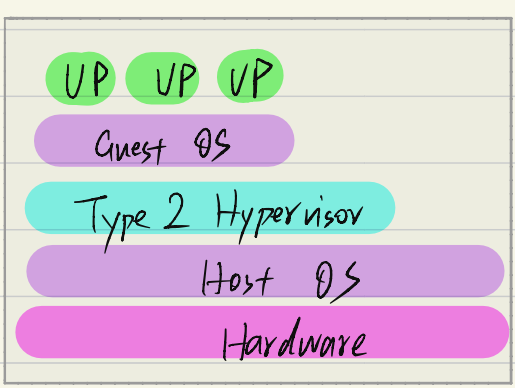


Other structure:

- 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

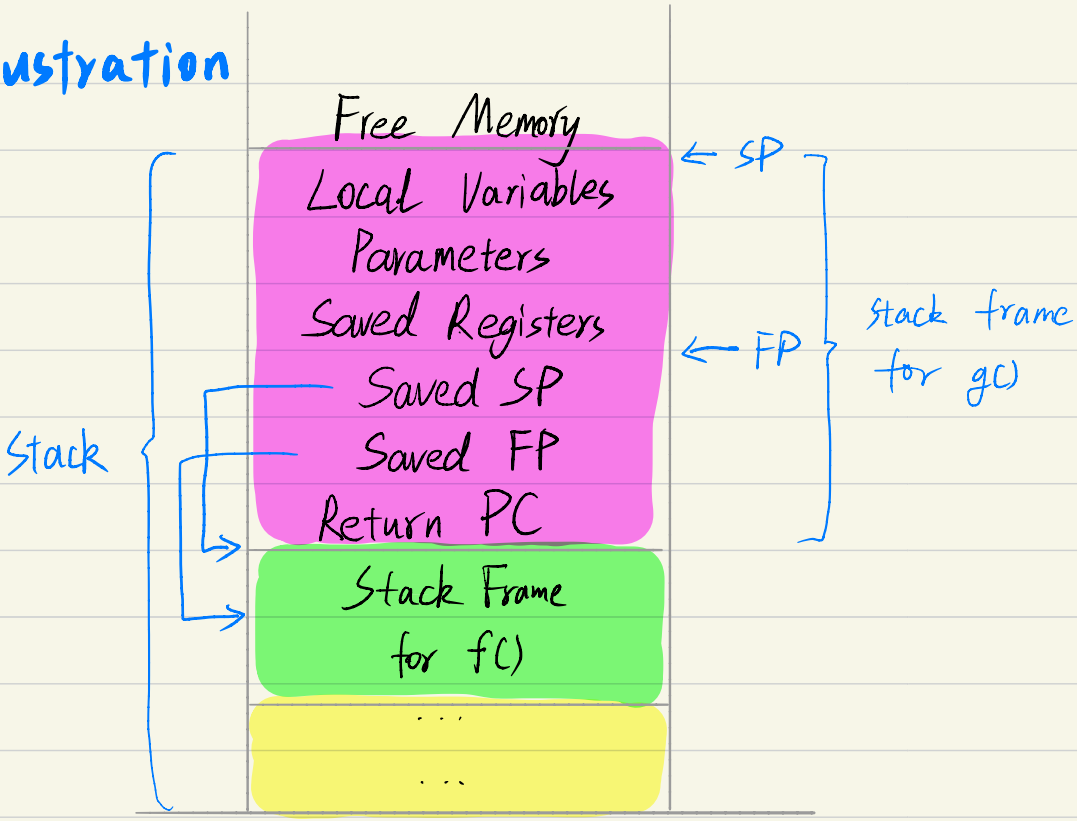
stack frame: information of one functio:

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

stack pointer (SP)

frame pointer (FP)

Illustration

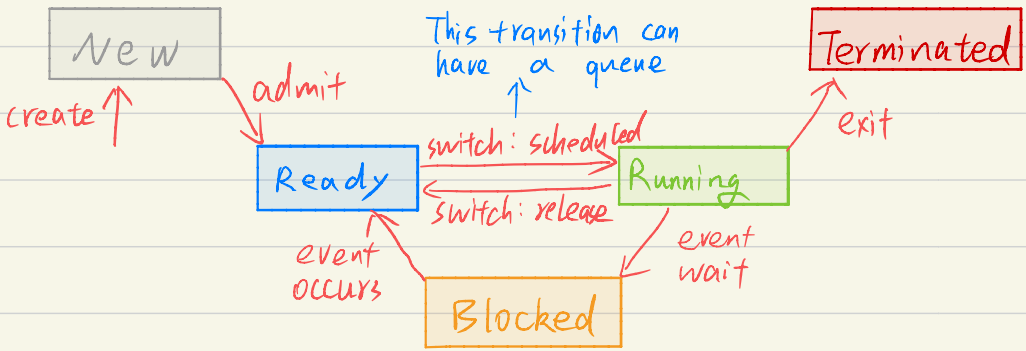


Dynamically Allocated Memory : heap memory

Process Id & Process State

PID

Generic 5-State Process Model



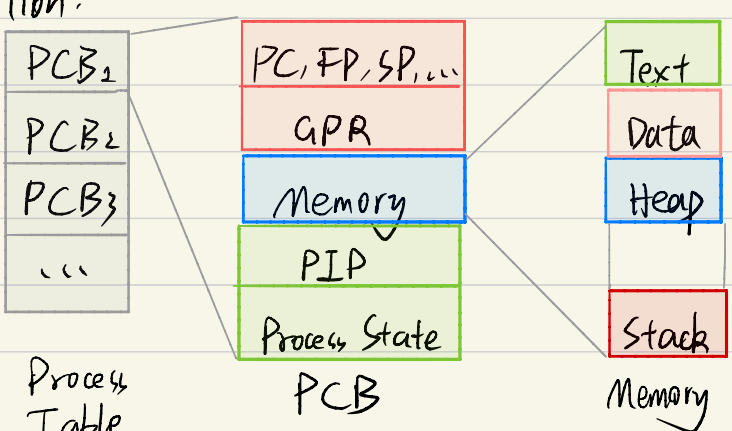
no more than 1 process can running on 1 CPU

Process Table & Process Control Block (PCB)

PCB is also Process Table Entry (PTE)

should have: Scalibility, Efficiency

Illustration:



System Calls

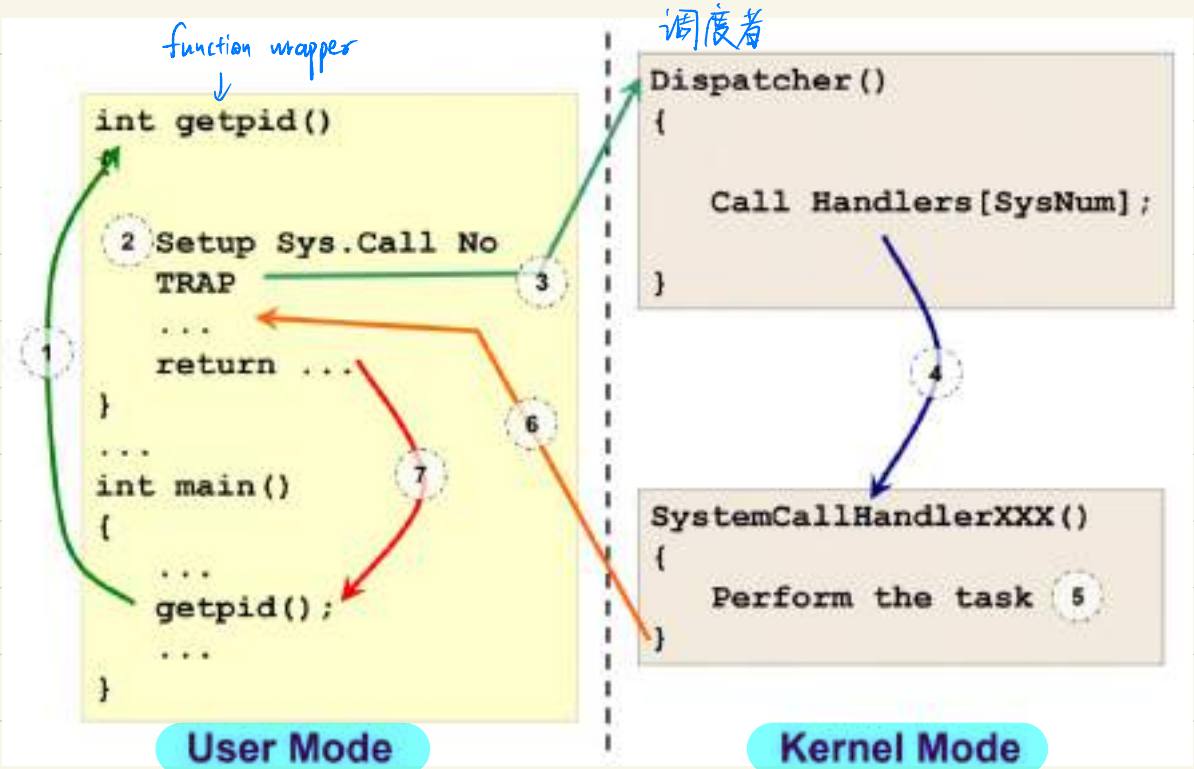
Unix : ~ 100

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)

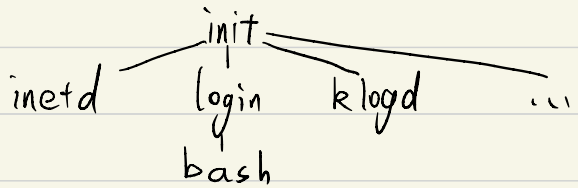
`execl()` : replace current process image with a new one

```
int execl(path*, arg[], NULL)
```
path*: ↑ new program
NULL: ↑ 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()

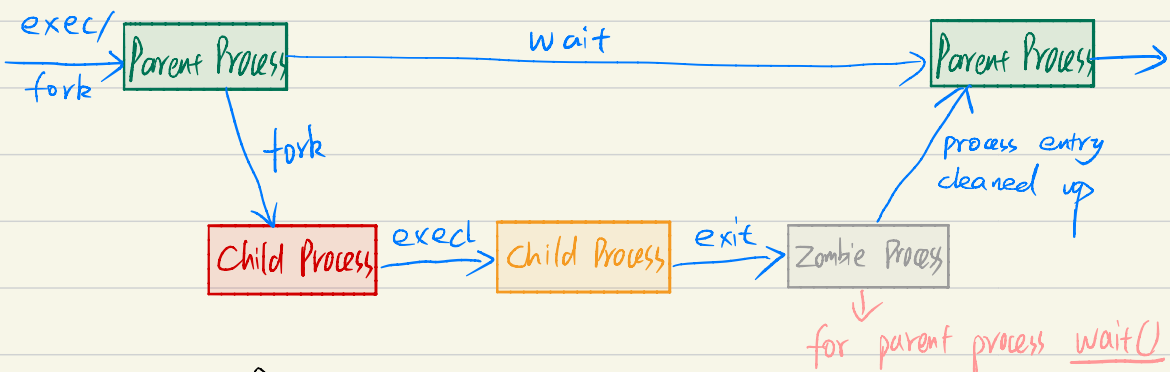
Parent process can `wait(int *status)` for child process to terminate

↑ could be NULL

Others:

`waitpid()` for a specific child process

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

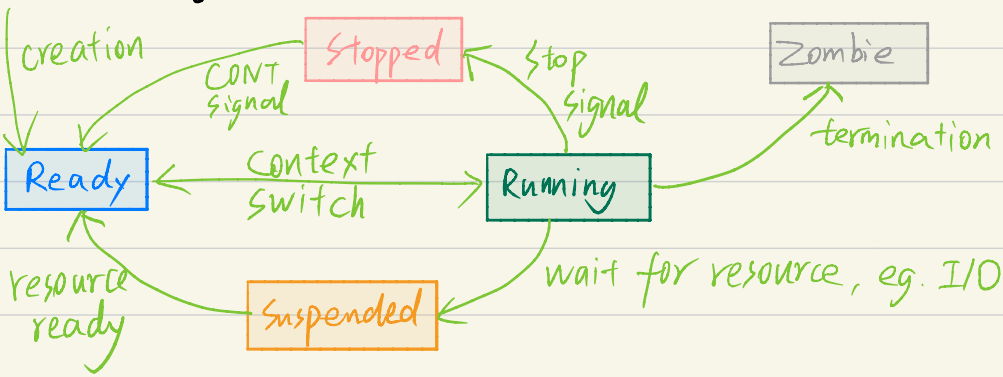


For Zombie Process:

- 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()

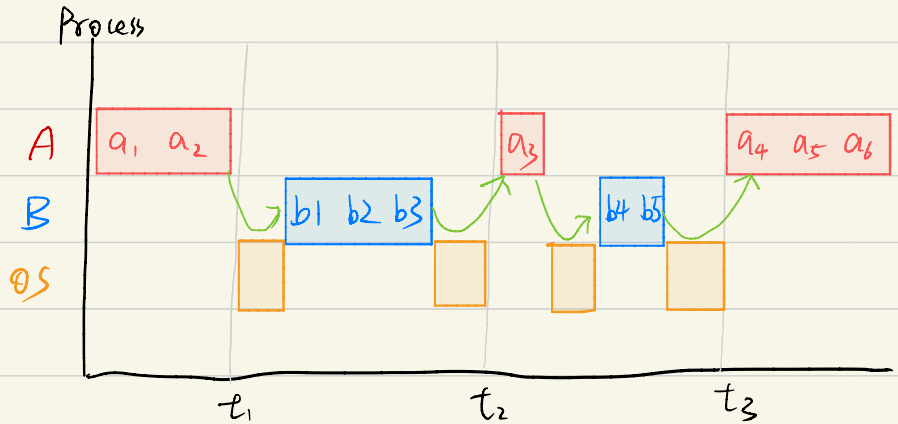
1. create address space for child process
2. Allocate p' = new PID
3. Create kernel process data structure (e.g. PCB)
4. Copy kernel environment of parent process (e.g. priority)
5. Initialize child process context (PID, PPID, zero CPU time)
6. Copy memory region from parent (expensive)
7. Acquaires 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

Process Scheduling

Concurrent processes: could be virtual
could be physical



timeslicing

Scheduling problem: Scheduler ; Scheduling algorithm

- CPU-Activity: Compute-bound process
- IO-Activity: IO-bound process

Processing Environment
- Batch processing
- Interactive: responsive
- Real time: periodic, deadline

Criteria for all processing environments:

fairness (no starvation), balance

Scheduling policies:

- Non-preemptive (cooperate): stay running till blocked
- Preemptive: a fixed time quota of running each time

good for batch processing system

needs scheduling algorithm:

- First-Come First Served (FCFS)
- Shortest Job First (SJF)
- Shortest Remaining Time next (SRT)

Criteria for batch processing

- Turnaround time: total time taken
- Throughput: finished tasks per unit time
- CPU utilization

FCFS: no starvation, could have Convoy Effect

SJF: minimum avg waiting time, possibly starvation

need to know total CPU time in advance

variation

$$Predicted_{n+1} = \alpha\, Actual_n + (1-\alpha)\, Predicted_n$$

SRT: Preemptive: New job with shorter remaining time can preempt current running time

Criteria for interactive environment
- Response time
- Predictability

For Periodic scheduler → Timer interrupt → invokes scheduler

Interval of Time Interrupt (ITI)
Time Quantum (n * ITI)

Scheduling Algorithm: - Round Robin (RR)
- Priority Based
- Multi-Level Feedback Queue (MLFQ)
- Lottery Scheduling

Round Robin: Basically a preemptive version of FCFS
A FIFO queue for all ready process, running at most one Time Quantum

Priority Based: has preemptive and non-preemptive version
can cause starvation
Priority Inversion: Low-priority preempts higher priority.
By locking resources

MLFQ

Basic: Priority higher : run
Priority equal : round robin

Priority: New with higher priority
fully utilize time slice → priority ↓ (likely CPU..)
give up / block → priority ↑ (likely I/O activity)

Lottery Scheduling: choose randomly with weight

simple implementation

IPC Inter-Process Communication

- Shared memory
- Message passing
- Unix: Pipe, Signal

Shared Memory

P_1: create M, read/write to M

P_2: attach M, read/write to M

Efficient but hard to synchronize

Message Passing

Direct: Send(P_2, Msg), Receive(P_1, Msg)

Indirect: Send(MB, Msg), Receive(MB, Msg)

Mail Box

could be: Blocking Primitives (synchronous) 可能等待

Non-Blocking Primitives (asynchronous)

Portable, easier synchronization but inefficient

Unix Pipe / Signal

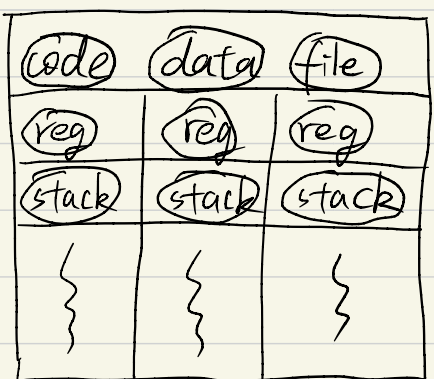
略

Threads

- process is expensive : - duplicate memory space
  - duplicate process contex
  - switching all above
- Communication is hard

Threads share : Memory context, OS context
unique resource : id, register, stack

Benefits: - Economy
- Resource Sharing
- Responsiveness
- Scalability



multi-threading

Problems: - System Call Concurrency
- Process Behaviours

Thread Models

- User Thread: implemented by user library
- Kernel Thread: implemented by OS
- Hybrid Thread

Synchronization

"Race condition": synchronization problem

Solution: Critical Section (CS) 临界区块

CS properties: mutual exclusion (互斥)
progress
bounded wait
independence

Incorrect Synchronization: · Deadlock · Live lock · Starvation

Implementation: - Assembly - HLL - High Level Abstraction

TestAndSet Register, MemoryLocation

it's atomic

① Read ② set to 1

it employs busy waiting

shared memory

Turn 0
Want[0] 0
Want[1] 0

Peterson's algorithm

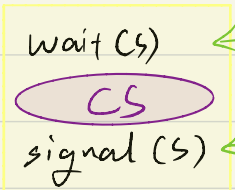
- busy waiting
- low level
- not general

P_0	P_1
Want[0] = 1	Want[1] = 1
Turn = 1	Turn = 0
while (Want[1] and Turn == 1)	while (Want[0] and Turn = 0)
CS	CS
Want[0] = 0	Want[1] = 0

atomic

High Level Abstraction

Semaphore 信号量 proposed by Dijkstra in 1965



Conditional Variable can broadcast

Classical Synchronization Problems

- Producer Consumer
- Reader Writer
- Dining Philosophers

Memory Abstraction

Address reallocation: Base + Limit registers
(use as offset, check)

Good idea: logical address —map→ physical address

Contiguous Memory Management

free up memory by:
- Remove terminated process
- Swapping blocked process to secondary storage

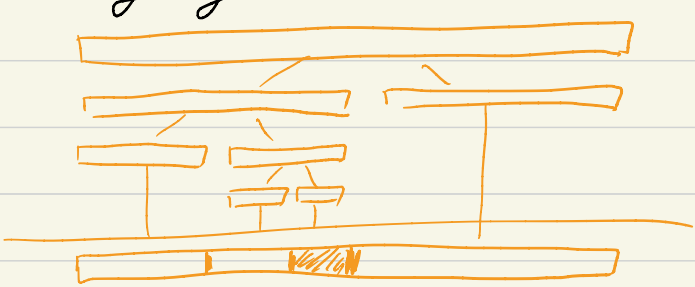
Memory Partition: fixed-size (easy but wasty), variable-size

variable-size → have "holes" → free space by: merging holes (adjasent), compaction

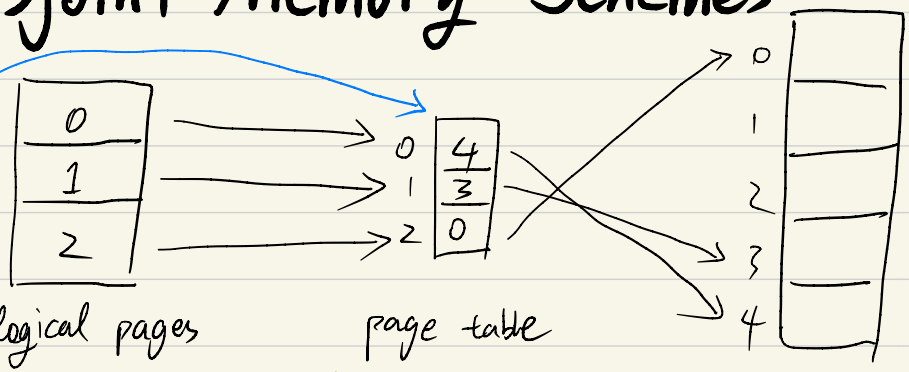
holes: first-fit, best-fit, worst-fit

use Linked List or Bitmap to maitain the partitions

Buddy System



Disjoint Memory Schemes



have internal fragmentation

no external fragmentation

Hardware support:

Translation Look-aside Buffer (TLB)

↖ cache

can hit or miss

Protect: Access Right Bits

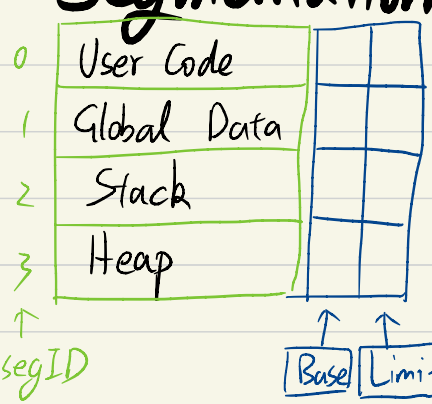
Valid bits like cache

Page sharing: different logical pages to the same physical frames

used for - code sharing - Copy-On-Write

Segmentation

can be combined with paging: <pageID, pageOffset>



∈ 4 kinds of segmentation

Logical Address: <segID, offset>

↓

if offset ≤ limit

OK

if offset > limit

address error

← segment table

∈ physical address