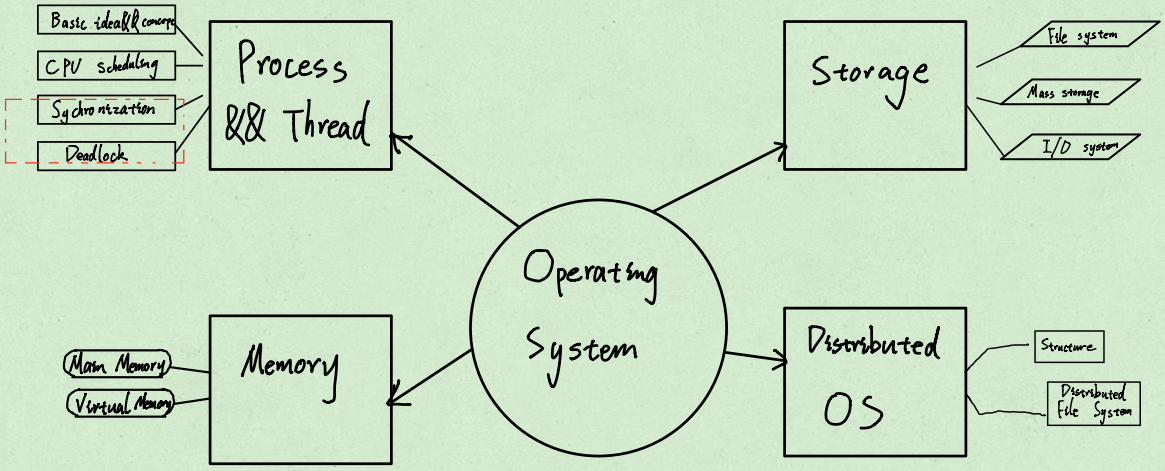
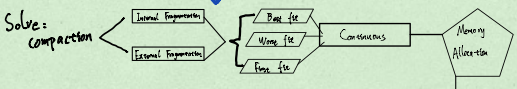
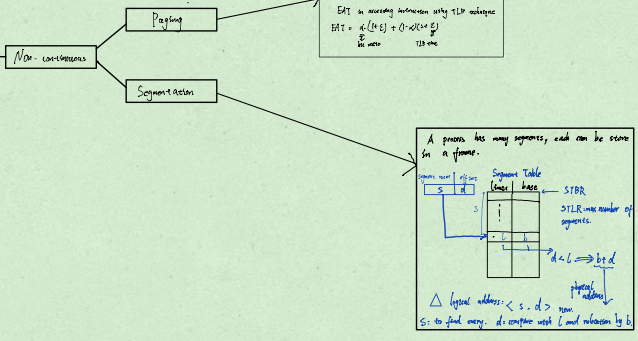
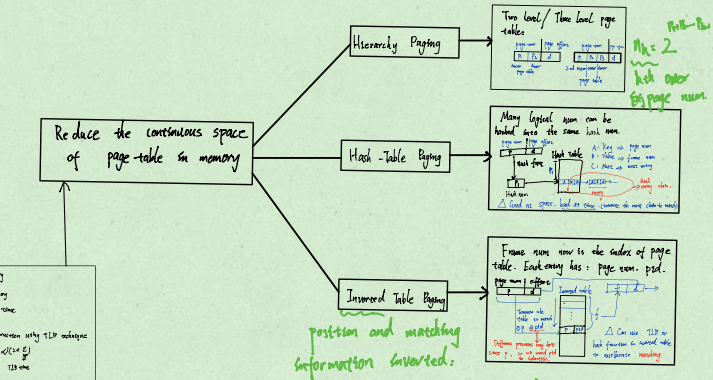


Main Memory:







position and matching information inverted:

original: page num ⇒ position information

matching ⇒ frame num

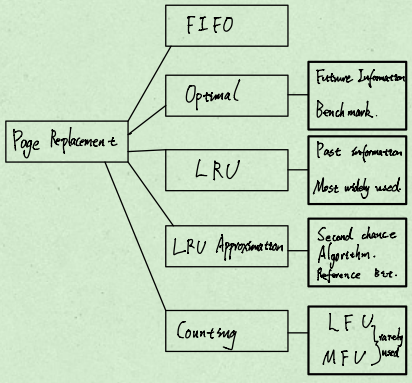
inverted: page num ⇒ matching (+ pid)

position information ⇒ frame num

Virtual Memory

4 elements:
Logical address. PageTable.
Main Memory. Virtual Memory

Page fault handling in Page reference:
① check se's valid or not.
② check page table's valid bit.
③ find empty frame in main memory.
④ Using Replacement strategy to choose victim and replace
⑤ update the page table, redo the page fault reference.



Virtual Memory's page table's difference with main memory's page table:

Type1: Virtual Memory's page table entry num = logical page num = virtual memory page num. > main memory frame num.
This case logical page maybe in valid or invalid state in page-table

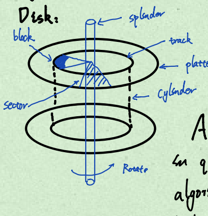
Type2: Virtual Memory's page table entry num = main memory frame num. < virtual memory page num = logical page num.
This case logical page maybe in valid or dirty cases or invalid (3 in total) state. in page table.

△ In main memory part, page table entry num = logical page num = main memory frame num.
So we can always have a valid logical page matched with a page table entry.

Mass storage system:

Typical: Magnetic Tape. Disk. HDDs. SSDs.

Disk:



Position Time (Random Access Time) = Seeking Time (main) + Rotation Latency (≈ 0)

A sequence of disk requests may need to in queue if disk is busy, so we need algorithms to schedule the service of disk.

Scheduling Algorithms: ① FIFO ② SSTF (Shortest seek time first) ③ SCAN ④ C-SCAN ⑤ LOOK. ⑥ C-LOOK. ⇒ in the same direction

File system

File: A collection of relatively data.

File attributes (name. size. type...) keep in a directory entry, all the entries are in directory structure.

File operation. (read. write. open...)

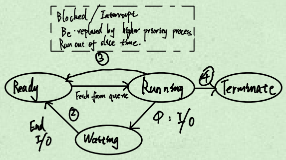
File access (group. user. public | read | write | exe)

File organization: single-level. two-level. tree-structure directories.

File Allocation

- **Continuous Allocation**
  - External fragment. problem.
  - Fast. (Read/write head don't move long). Best performance.
  - Simple.
  - Directory entry: | Name | Start | Length |
- **Link-list Allocation**
  - Slow (many times I/O and disk access)
  - No external fragmentation.
  - A file is divided into several blocks in a link list, each block has a pointer to the next block.
  - Directory entry: | Name | Start | End | (end block, start block)
  - File Access Table (FAT) stores the file allocation of all file blocks. Link-list in FAT, but FAST (be loaded into memory.).
    - Directory entry: | Name | ... | Start block num (in FAT.) |
- **Index Allocation**
  - Index block && Data block. && Index table block.
  - (store the information and direct pointer. single-indirect pointer. double-indirect ......)
  - Directory entry: | Name | ... | Index table block |
  - Index table mapping: ① start block is index table block ② bad block ③ indirected pointer point to index blocks, index blocks' element point to data blocks. ④ only get new block when needed.

CPU scheduling:



CPU scheduler: preemptive → ①.③ ; non-preemptive ⇒ ①.④

Dispatcher ⇒ Dispatch Latency × Switch Context

Scheduling Criteria:

① CPU Utilization
② Throughput
③ Turnaround Time } calculate
④ Waiting Time } average
⑤ Response Time (used in Time-sharing)

Process Scheduling

- **FCFS**: Convoy Effect: short process behind long processes. like if have one CPU bound and many I/O bound but CPU bound start first
- **SJF**: ① Starvation ⇒ Aging can solve. ② Need to know the CPU burst of next time ⇒ Exponential Moving average to estimate: $\tau_{n+1} = \alpha \cdot t_n + (1-\alpha)\tau_n$ ($\alpha: 0 \le \alpha \le 1$, commonly 0.5) ($\tau_{n+1}$: predicted; t_n: actual length; τ_n: predicted)
  ③ It's optimal. Using SJF can have the min waiting time.
  ④ Shortest Remaining-Time Job First (SRJF) ⇒ preemptive
- **PS**: ① SJF is PS's specific case. ② Starvation ⇒ Aging can solve ③ Priority num for each process
- **RR**: ① quantum: q. q >> 1: RR = FCFS; q << 1: Low Limit: q >> Dispatch Latency.
  ② When a process ran out of its time slice, if there are n process, it need to wait no more that (n-1)·q.
  ③ How to choose q: 80% of process CPU burst < q
- **Multi-level**: ① There are many Ready queue, and difference scheduling scheme in different queue. (System. Interactive (前台). Real-time. Batch (后台))
  ② Scheduling: Fixed Priority (May cause starvation. Low pri-queue need to wait) / Time slice (Higher priority queue, longer time slice)
- **Multi-level feed back**: ① Process can enter different queue. (In Multi-level, one process can only in one queue)
  ② Much like Multi-level, Fixed Priority scheduling. If don't finish, move to the next level queue.
  ③ When a process in lower level queue is preempted, then it don't change level, and start from a new quantum next turn.

Thread Scheduling

- **PCS**: Process Contention Scope: user level threads ⟹ kernel level thread
  ① Scheduled by thread library.
  ② Priority set by programmer.
  ③ Appear in Many-to-Many / Many-to-one Model
- **SCS**: System Contention Scope: kernel level threads ⟹ LWP (then to use CPU)

Examples: Linux, Solaris, Windows XP

They are all similar to each other.

① priority scheduling ② Multi-level queue. (Scheduling classes) — At first, ① real-time class ② default class

③ Relative priority && Global priority.

④ Some processes have fixed priority, others have dynamic priority.

Linux: (i) Two array — Running, Expire } higher priority = low index = longer quantum = higher decay fator

(ii) When put into Expire array, other processes (real time processes fixed priority) dynamically change priority (nice value)

(iii) When all run array entry done, exchange.

(iv) Each time scheduler select: highest class, highest task in the class.

Solaris: (i) Six classes. Priority (high → low): 1. (Interrupt) > 2. Real-time > 3. System > 4. fair shared > 5. fixed priority > 6. time-shared > 7. Interactive

Dead lock:

Dead lock:
what will cause deadlock: when these four conditions are all satisfied: ① Mutex Exclusion ② Hold and wait ③ No preemption ④ Circular Waiting

How to solve deadlock problem:

① No dead lock.
- Deadlock Prevention 预防
- Dead lock Avoidance 避免

② Possible deadlock: Detect and recovery

Dead lock Prevension: break one of four conditions.

① Avoid Mutual Exclusion ② Avoid Hold and wait (一次性全部分配, 申请时自己不可hold 资源. ③ Give preeptive process. 抢占进程, 可抢占 Hold and wait process的资源. 被抢占进程要等到 (i) old resource get (ii) request resource get 才能继续执行.

④ 给 Resource Total order (全序), process can only request resources with increasing number.

Periodically call Detect Algorithm. If has deadlock, need to recovery. Select some/one process to kill and preempt some resources. → minimum cost.
(lowest priority, shortest exe time,
(largest number of resources,
have a long time to complete)

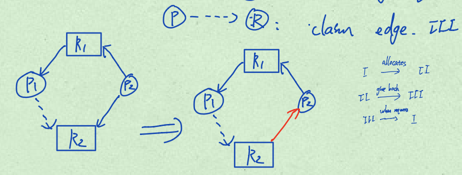
Dead lock Avoidance: when allocate resources to a process request, always ensure the system in safe state.

Avoidance Algorithm — resource
- single instance: Resource allocate graph
- Multiple instance: Bankers Algorithm.

Process states: Safe / Unsafe / Dead lock

△ Need priori information (maximum demands)

Resoure allocation graph:
- P → R: request edge I
- R → P: allocate edge II
- P - - -> R: claim edge. III



△ request edge turn to allocate edge only when it will not form cycles.

Banker's Algorithm:
ⓐ Max. ⓑ Allocation. ⓒ Need. ⓓ Available (关键)

① Max - Allocation = Need.
② Need ≤ Available, 分配给该process.
③ Available += allocation

Each time when we want to allocates resources, assume we have allocated and update the ⓐ,ⓑ,ⓒ,ⓓ values and call detection Algorithm, find whether it's safe.

Process Sychronization:

Sychronization technique
- Hardware ⟹ Atomic instructions (e.g. TestandSet) I
- Peterson's Solution ⇒ Solve two process problem. II
- Semaphores III
- Monitors IV

I.

Sychronization Harware: TestAndSet()
lock is initialized to **false**.

Requirements
- Mutual Exclusion
- Progress
- Bounded Waiting

```
do {
  waiting[i]=true;
  key = true;
  while (key && waiting[i]) {
      key= TestAndSet (&lock);
  }
  // do critical section.
  j = (i+1)% size;          → Assume that we know total process number.
  while (j!=i &&(waiting[j])
      j=(j+1)% size;
  if(j==i) lock=false;
  else waiting [j]= false;
} while (true)
```

key → 为局部变量
lock → 为全局变量

Swap ():
lock is initialized to false;

```
do {
  key= True;
  while( key) {
    swap ( key, &lock);
  }
  //do critical section;
  lock = false;
} while(TRUE)
```

key为局部变量, lock为全局变量

△ Process Syndronization 核心即确定三点:
③ 先到的先执行 ①两个process不能同时执行
② 后到的在等先到的执行完毕后也应该被执行.

e.g TestandSet () 中, lock为了满足①, key 为了满足③, waiting 为了满足②

II.

Peterson's Solution:

Use flag[2] and turn variables.

△ It's used in two process synchronization.

```
do {
  turn = j;
  flag[i] = true;
  while ( turn == j && flag[j]) # wait;
  // do critical section
  flag[i] = false;
} while ( True)
```

均为全局变量

① 将turn让给对方, 确保了次序先后.

② First arrived process's while is broken by ②, later arrived process's while is broken by ②.

flag [1 i ; 1 j]
turn : i

i process:
```
do {
  turn = j;
  flag[i] = true;
  while ( turn == j && flag[j]);
    // do critical section
  flag [i]= false;
} while (true)
```

j process:
```
do {
  turn = i;
  flag[j]= true;
  while ( turn == i && flag[i]);
    //do critical section
  flag [j]= false;
} while (true)
```

Distributed System.

Speed up = $S = \frac{T_1}{T_p}$ Effciency = $\frac{S}{P}$.

Network Structure
- LAN
  - Bus
  - Star Topology
  - Ring
- WAN — Network Topology. How to assess: Installation Cost. Communication Cost. Reliability

fork. vfork.

Create: subprocess
特征：子进程完全copy父进程一份，开销很大。

shared space subprocess
减少不必要开销
特征：父进程被阻塞直到子进程exit/execl.

clone. pthread_create.

create LWP.
与pthread_create类似，但能访问kernel

LWP与HWP：一个process独自占有资源 ⇒ HWP.
与其它process共享资源.

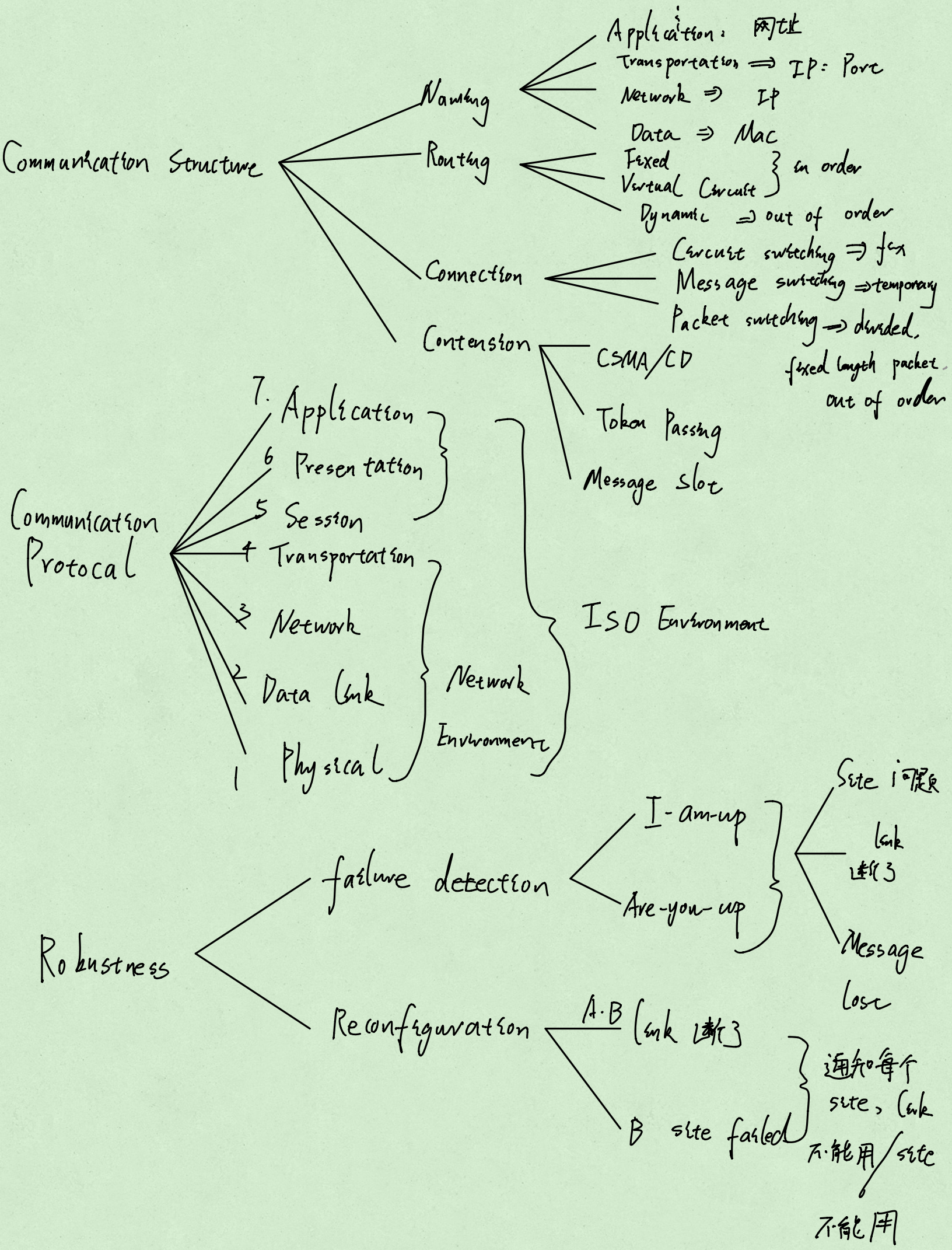
create user threads.
由线程库管理，无法访问内核。
创建. 同步. 销毁.

Lecture 12: Distributed System Basic Knowledge Introduction

- Motivation (分布式系统优点)
  - Resource Sharing ⇒ 可用不同设备特种功能
  - Computing Speedup ⇒ 提速
  - Communication ⇒ 交换信息
  - Reliability ⇒ 更可靠
- Metrics
  - Speed: $\frac{W}{T_P}$ ⇒ W: workload; T_P ⇒ 单 node 时间
  - Speedup: $S = \frac{T_1}{T_P}$
  - Efficiency: $\frac{\text{Speedup}}{N}$ ⇒ N processors
- Types
  - Network OS: User be aware of Multiplicity
  - Distributed OS
    - Not aware of Multiplicity
    - Data
    - Computation
    - Process

    } + migration
- Network structure
  - LAN Topology
    - Bus
    - Ring
    - Star
  - WAN Topology
    - Tree
    - Partial Connected
    - fully connected

  Compare Criteria:
  ① Installation cost
  ② Communication Cost
  ③ Reliability


Application: 网址
Transportation ⇒ IP: Port
Network ⇒ IP
Data ⇒ Mac
Naming
Routing
Fixed
Virtual Circuit
in order
Dynamic ⇒ out of order
Communication Structure
Circuit switching ⇒ fax
Message switching ⇒ temporary
Packet switching ⇒ divided, fixed length packet, out of order
Connection
Contension
CSMA/CD
Token Passing
Message slot
7. Application
6 Presentation
5 Session
4 Transportation
3 Network
2 Data Link
1 Physical
Communication Protocal
Network
Environment
ISO Environment
Site 问题
link 断了
Message Lose
I-am-up
Are-you-up
failure detection
Robustness
A-B link 断了
B site failed
通知每个 site, link 不能用/site 不能用
Reconfiguration

Lecture 13: Distributed file system.

Naming
- Location Transparency: file name doesn't reveal the file's physical location (Most DFS)
- Location Independence: file's physical location changes. file name is no need to change. (AFS)
- Neither of them: host name + local name.

Stronger

Remote File Access

Caching
- Cache update policy
  - Write back
    - periodically
    - when closed. (AFS)
  - Write through
- Cache Location
  - Disk: more reliable
  - Main Memory: ① Quick ② Client can be diskless ⇒ (Server cache's location)

Consistency
- Client-Initiated: Client raise an validity check and server response.
- Server-Initiated: Server records each client caches and detect and react

File replication.
- Advantages
  - ① Reduce Network Traffic and server load., increase scalability
  - ② Quicker
  - ③ Secure when run into error.
- Desadvantages
  - Cache-Consistency problem

Comparence:

① Caching && Remote Service

优势: Caching 在不频繁写入中更有优势, 而 Remote Service 在频繁写入有优势

应用: Cache ⇒ Disk, large main memory

Remote file Access ⇒ Diskless small memory

Lower intermachine interface: Cache ⇒ 不同

upper user interface Remote file access: 相同

② Stateful File Service && Stateless File Service.

优势: Stateful: ① Quick — reduce disk access; can read ahead the next blocks

② Shorter Request messages: 每个 request 信息只包含 file, 并无 position.

Stateless: ① Robustness: 打开·关闭时 无需建立. 断开 connection identifier

When run into a crash, stateful server lose all states.

While in a stateless server, it doesn't be affected.

DFS structure — Server, Client — Service