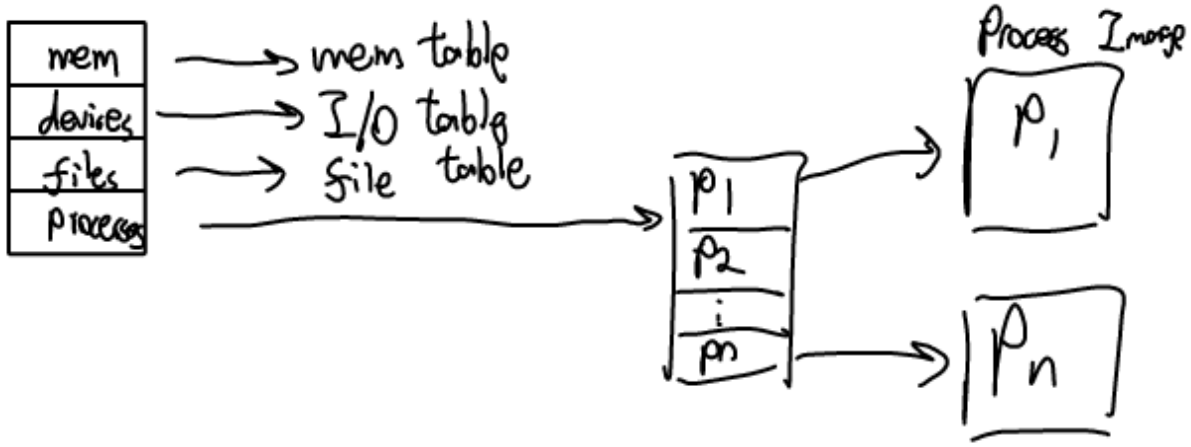
Process Description

OS control structures: 3.3

OS must keep track of processes & resources.

4 different types of tables:

- memory
- I/O
- File
- Process



Memory Tables:

keep track of main & virtual memory

- allocation of main mem to process
- allocation of secondary (virtual) memory to process
- protection attributes of blocks of memory
  i.e. which process can access which shared mem regions.
- info needed to manage virtual memory

I/O Tables:

manage I/O devices & channels.

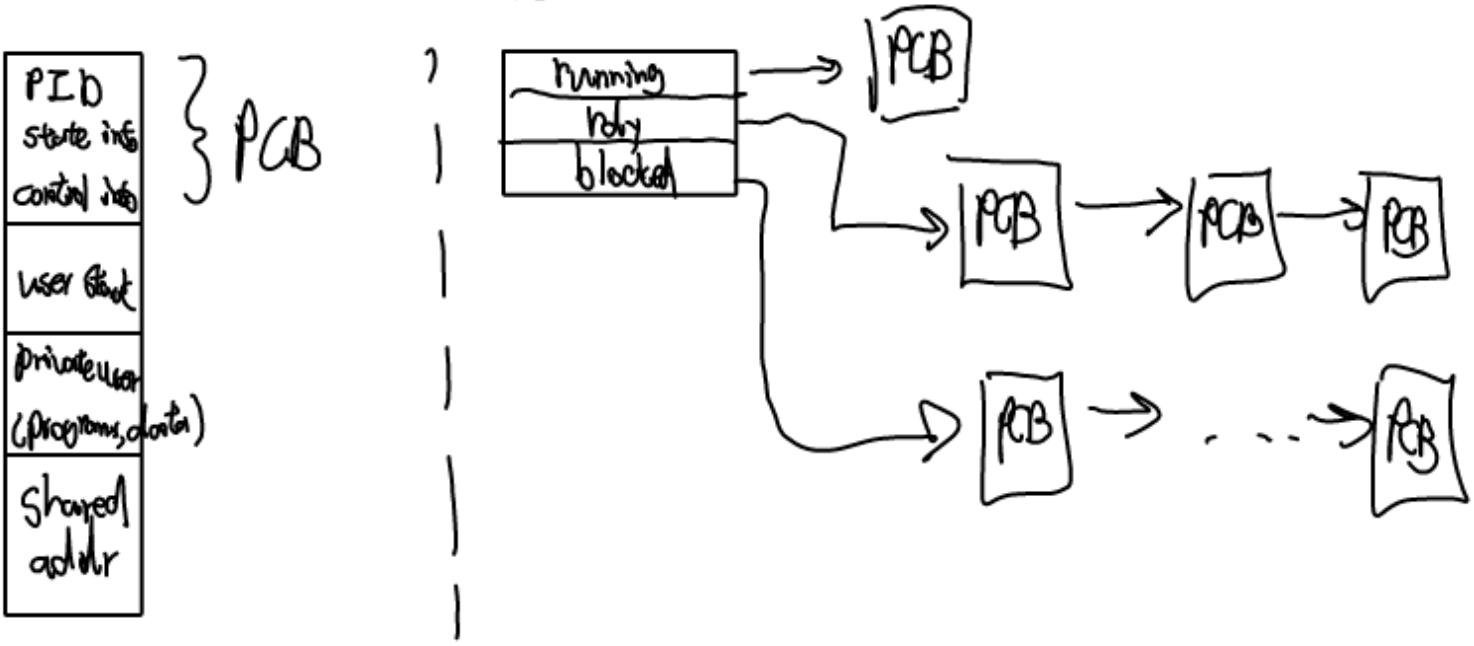
- status of I/O operation
- I/O device free? or assigned to process
- location of src & dest of I/O transfer in main mem

File Tables:
- existence of files, their location on 2nd mem, status, etc

Process tables:
- manage processes
- reference to other tables

Process Control Struct

- Process Location
- Process Attributes
  - Process ID - unique #
  - Processor State info - contents of processor registers
  - Process Control Info - info to control & coordinate proces
    - process state
    - priority
    - scheduling info
    - events



3.4 Modes of execution

user-mode (less-privileged)
system/control/kernal mode (more-privileged)

Typical Fuctions of OS Kernal:

Process Management:
- Creation/termination
- scheduling/dispatching
- switching
- synchronization & communication
- PCB management

Mem:
- allocation
- swapping
- page & segment management

I/O:
- buffer management
- allocation of channels/devices to processes

Support:
- ITR handling
- accounting
- monitoring

Process Creation

1. Assign unique PID
2. Allocate space
3. Initialize PCB
4. set appropriate linkages (i.e. put in Rdy queue)
5. Create/expand other data structure (e.g. accounting file)

Process Switching

when?

1. interrupt, 2. trap, and 3. supervisor call.

1. clock interrupt / time slice → switch to rdy
   I/O → resume or preempt to higher priority
   Memory Fault → blocked, switch. If memory is available in main, unblock
2. if error / exception is fatal, put to exist state & switch
   else depends
3. may block process.

Mode Switching

when interrupt is pending, the processor:

1. sets PC to interrupt handler
2. switch from user to kernal mode.

Change of Process State

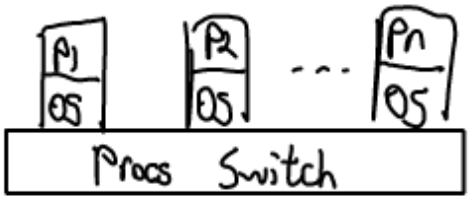
1. save processor context: pc & other regs
2. update PCB of running process
3. move this process to right queue
4. select new process
5. update PCB and change state to run
6. update memory management data structure
7. restore context: pc & regs

3.6 Execution of OS

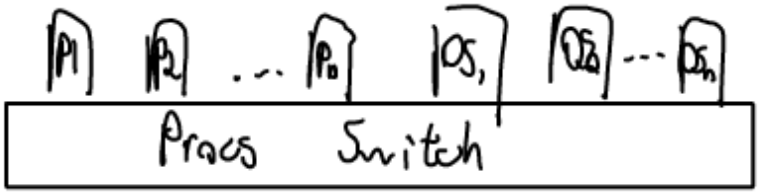
1. Non-process Kernal

2. Execution within User Procs:

Additional kernal stack in Process image



3. Process-Based OS:



2. avoids 2 process switches
3. clean, minimal interfaces Modular OS.

3.6 Security

System Access Threats

intruders:

outsider → Masquerader: not authorized & exploits legit user

insider → Misfeasor: legit user, but access is unauthorized or misuse priv.

both → clandestine user: seizes supervisory control
evade auditing & access control
suppress audit collection

Malware:

parasitic: needs host program

independent: x

Counter intruder detection System

- Host-based IDS
- Network-based IDS
- sensors
- analyzers.
- UI

Authentication
- Identification step
- verification step

4 means:
something user knows: password, PIN, answers
user has: keycards, keys, token
user is: fingerprint, retina, face
user does: voice, handwriting, typing rhythm

Access Control:
authentication determines if system access is ok
access control if specific access is ok by consulting authorization db.
auditing keeps track of accesses.

Firewalls: all must pass thru
only authorized can pass
immune to penetration

4.1 Processes & threads

A process may have ≥ 1 threads, each has:
- thread exec state
- thread context when !running
- exec stack
- per-thread static storage for local var
- access to mem & resource

thread: less time to create, terminate, switch than proc
more efficient in communication.

e.g. . fore/background work
· async processing
. speed
. modular program struct

States: spawn, block, unblock, finish

4.2 types:

User-level threads: managed by application
kernal is unaware of the existence of threads.

kernal-level threads: managed by OS.

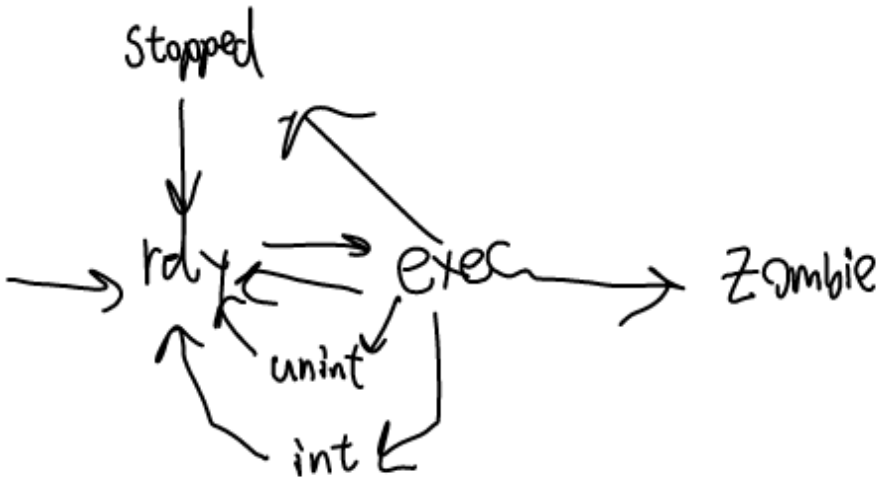
4.3

$$\text{speed up} = \frac{\text{time to exec on 1 processor}}{\text{.. .. .. .. N parallel processes}} \leq \frac{1}{(1-f)+\frac{f}{N}}$$

for fraction f of code that is infinitely parallizable.

4.6

Linux thread State



5