info needed for process management:

- pid, user id, ppid
- state info (registers)
- control info

  state, privileges, etc.

states

- Running -> access to the CPU
- Ready -> when process interrupted, but can still use CPU
- Blocked -> wait input from 3rd party
- Suspend/swapped -> have as much active as possible

Privileges

User mode -> not all instruction executable, some memory not accessable

Kernel mode -> no restrictions

when system call need mode switch

process switch -> i interrupted, next process in queue is j
register preempted state of registers
update process table of i, j

mode switch -> after interrupt handler same process executed
just have to swap state info

Threads

partition a program more efficiently than with multiple processes.

they share info without switch to Kernel mode for shared memory

processes

(-) slower creation

(-) Kernel for communication/sharing

(+) concurrency guaranteed by OS

(+) different process states

(+) swapped indipendently

threads

(+) fast creation, switch

(+) immediate info exchange

(-) synch. and mutex not guaranteed

(-) shared process state (user level threads)

(-) jointly swapped

User level

thread management by process

Kernel not aware. Made by library

Kernel level threads

handled by Kernel, scheduling at thread level.

low efficiency -> single thread change mode switch