Scheduler

Selects a process to execute and passes control to it.

- proc. is chosen out of “READY” state proc
- save context of the “earlier” proc. and loading context of “new” proc. process need to happen.

When is scheduler called
Intricacy of “passing control”
What is “context”

Steps in scheduling:

if p1 and p2 is running, if p1 has timer interrupt, switch to kernel (scheduler) which then schedules p2.

---

4 times stacks have to change.

1. User stack of proc -->
2. kstack of proc --> seq. during interrpt
3. kstack of scheduler -->
4.kstack of new proc -->
5.user stack of new process

If shell is running and cat command is called.
Shell does fork-exec, now out of the two kernel will have to sched new.
When there is timer interrupt, context of shell is saved on kernel stack of shell.
Scheduler cannot run with same stack as of shell (as shell is application proc). Context of shell is currently on kstack of shell.

Scheduler doesn’t run on behalf of the process, it runs on behalf of the kernel. It does the job of the system.

To make scheduler run it will need stack of their own (as it will call lot of functions).
So here kstack of proc. changes to kstack of the scheduler, then only scheduler runs.

Then scheduler wants to pass control to “cat”, then again stack has to change to kstack to cat.
Because kstack of cat earlier would’ve loaded with its earlier context, So then cat will start executing

userinit in main creates proc called init. Process int is here, which we have to scheduled

mpinit : detects all processor and initialze

mpmain : makes all processor start execution

in scheduler, same kernel code runs on all cpus

schedule should iterate over those oric who are ready, then select a proc to exec then pass control.