Some other Possible Process State

- A process could be **Waiting** for something to happen
  - ↳ Example: parent process waiting for child process to terminate
    - through a wait() system call
- A process could be **Ready** for the OS to cause it to run
  - ↳ Example: After a child process terminates, the parent process may not be able to run right away because some other process is running.
- Running, Waiting, Ready → to be allowed to run by the OS
  - Running → on CPU
  - Waiting → for something to happen

Process Management

- OS job of managing processes that are waiting, running or ready
- What should the OS do when a process does something that takes a long time?
  - ↳ eg anything that involves the hard disk access
    - ↳ file read/write operation, page fault

An OS designer Question → does give us perspective

1. Solution → do nothing.
   - → Problem ⇒ processor is idle billions of cycles
     - hd seconds
     - CPU Nsecond
     - $\frac{1s}{1ns} = 10^9$
   - ⇒ the processor could have executed billions of instructions instead during that time

2. OS should try to maximize processor **utilization**
   - utilization ⇒ fraction of the time that the processor is busy → get to 100%
   - ↳ ...

busy → get to 100%

↳ How?

- OS could change status of that process to "waiting" and make another process "Running"
  - ↳ Question: Which other process?
    - ↳ Determined by the process scheduler

Process Scheduler

- The part of the OS that manages the sharing of CPU time among processes.
- Possible considerations that the scheduler could use in making scheduling decisions.
  - ↳ Minimize average program execution time

$$T_{ave} = \frac{T_{prog1} + T_{prog2} + T_{prog3}}{3}$$

↓ make scheduling decisions based on well being of all programs

BUT ↳ no guarantee of fairness

- ↳ Fairness to all the programs in execution
  - ↳ Do not sacrifice one or two processes in the interest of the average value

Process Scheduling Policies

Idea 1: Let the currently running process continue to do so until it does something that involves a long time

↳ disk access page fault

Non-preemptive policy

- ↳ then switch to a Ready Process
  - ↳ Problem → what happens if the currently running process is executing an infinite → debugging your program

very dangerous

while(1);

↳ No other process would ever get to run on the OS with Idea 1

↳ NOT used for Linux/Unix systems but used for some embedded systems.

Idea 2 → Preemptive Policy

for fairness or benefit of other processes

⇒ one where the OS "preempts" the running process from the CPU even though, the CPU is not waiting for something.

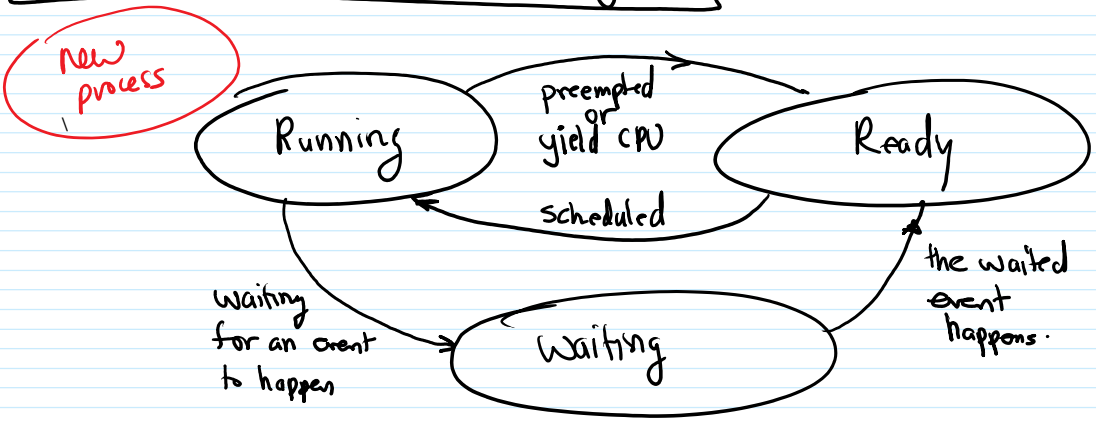
How? When do I evict?

↓

Idea ⇒ give a maximum amount of CPU time before preempting the process.

CPU time slice ⇒ maximum amount of time alotted to a process before preempting it from the CPU

Process State Transition Diagram



Context Switch

- The event of going from Waiting to Running

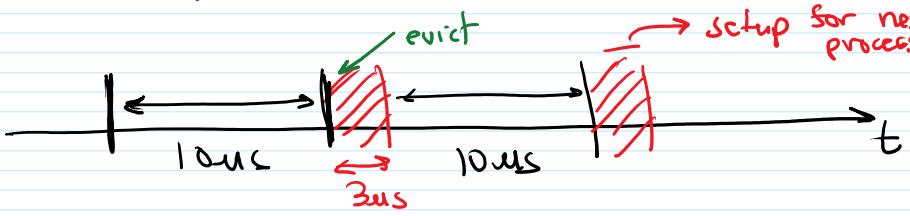
- The event of going from Waiting to Running
- The context switch takes time

P1 → P4

↳ save HW state of previously running process (contents of PC registers, IR etc)
↳ restore HW state of newly scheduled process

- The amount of time context switch takes determines CPU slice time

↳ CPU slice time >> context switch time



↳ generally CPU slice time ~ 1 second

Non-preemptive Scheduling Policies

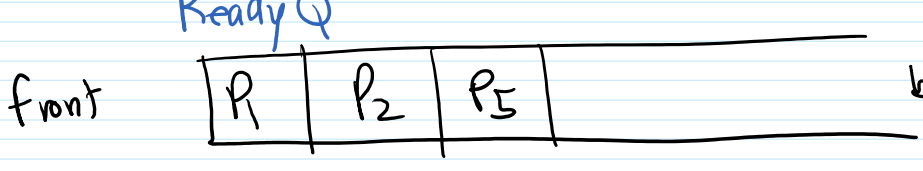
① First Come First Serve (FCFS)

Idea: maintain a queue of ready processes.

Queue ⇒ a data structure with 2 operations

Insert → add new process to the back of the Queue

Delete → remove the process from the front of the Que



schedule next process from the front of the Ready Q

Problem FCFS not a good policy. ⇒ doesn't do anything about processes that have small time requirements

② SPN – shortest process next

- The policy which results in the lowest possible average program execution time
- Schedule next process that which requires the least CPU time

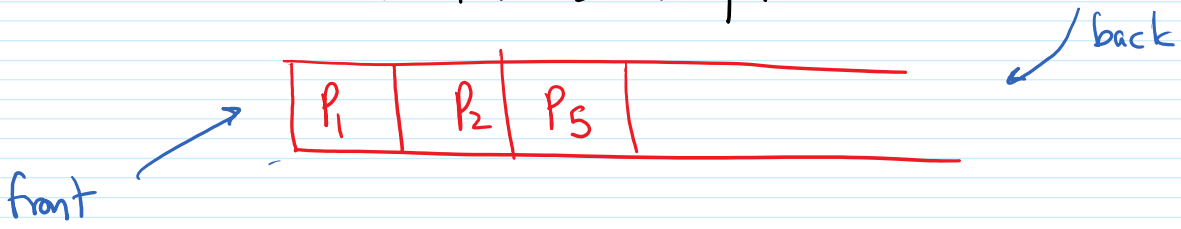
Problem → not very practical

→ looking into a future! ⇒ need a good estimate

Preemptive Scheduling Policies

① RR – Round Robin

↳ maintain a FCFS ReadyQ



much more fair than non-preemptive policies

- As before, when the currently running process is preempted, schedule the process from the front of the ReadyQ

BUT

- Insert previous process to the back