System Model for Scheduling

What all do we have in this system?

- A CPU
- A scheduler
- And a process queue

The CPU has absolutely no control over the process queue. It only has the control over the process it is currently running.

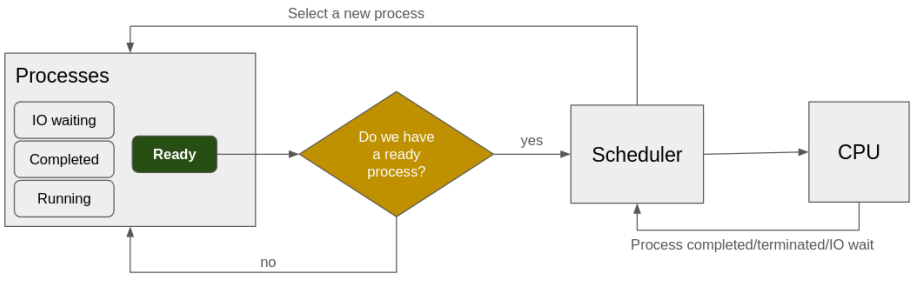
The scheduler is responsible in selecting a process from the process queue with a ready status and hand it over to the CPU. The process queue gets updated as the shceduler and CPU plays their respective roles. One thing to note is that process queue is not a system like CPU or scheduler, but is a data structure that contains process info.

High level overview of the mechanism

What all do we have in this mechanism?

- Processes in different states (Ready, Running, Waiting for IO, Terminated, Completed)
- CPU to execute the processes.
- Scheduler to schedule the processes.

Flow of the mechanism



When is the scheduling mechanism triggered?

- When a process in the CPU completes it's execution. (Software interrupt)
- When an external entity terminates the process. (Hardware interrupt)

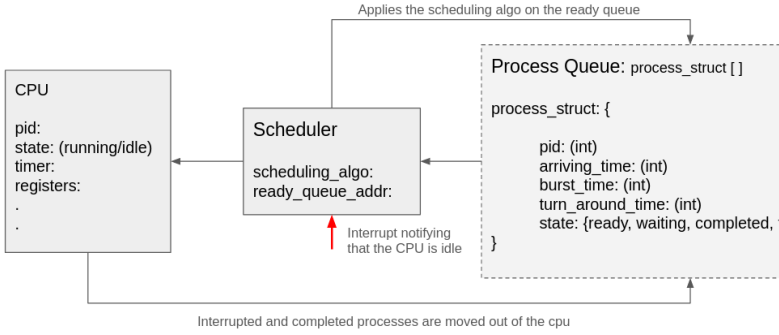
Based on the above understanding, we can categorize the system into following:

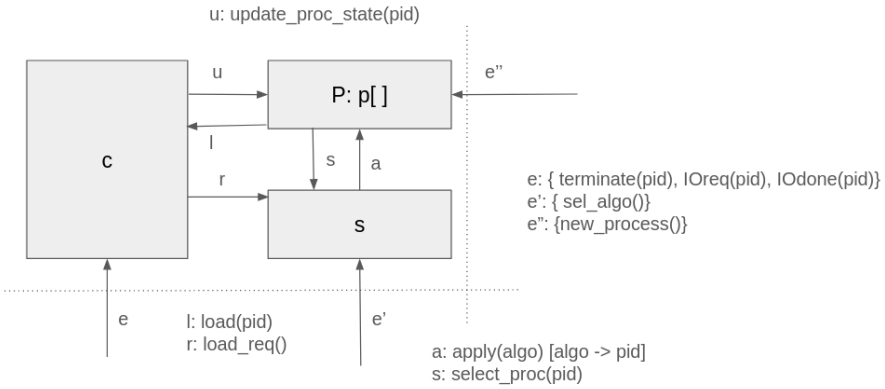
- Scheduler
- Environment
- Plant: CPU and Processes queue

Scheduler is where we focus on scheduling mechanism along with it's related functions like identifying the ready process, and selecting one based on the algorithm.

Environment is where we have other user dependent functions like creating a process, requesting for IO, fulfilling the IO request and terminating a process.

Plant is where we see the changes in the system as a result of our actions performed as an external user and a scheduler.





Transitional state view of scheduling

Let us look at how the system state changes as the execution of processes starts on the CPU.

Let us represent each state as

$$X_s = \begin{cases} \text{ReadyOrRunningQ:} \overrightarrow{id} & \text{A vectore representing all the processes ir} \\ \text{cpu: current pid} & \text{current pid represents the current process} \\ \text{Ptime Map:} \overrightarrow{id} \rightarrow \overrightarrow{executedTime} : \overrightarrow{TotalExecutionTime} & \text{Maps each process to the time they have} \\ \text{timer: T} & \text{The amount of time after which the CPU} \end{cases}$$

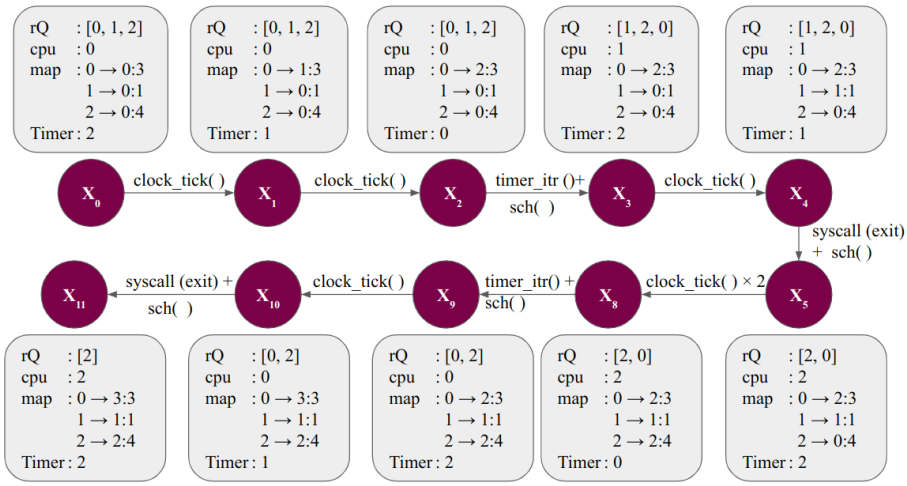
Due to space constraints, let us represent ReadyOrRunningQ with **rQ** and Ptime Map with **map**. A transition system can change it's state only when we have a function or action that triggers the state change. So let us define few basic functions that can change the state of a system.

timer$_{itr}$ (): The hardware generates an interrupt whenever the Timer count reaches 0. This implies that the currect process running on the CPU has used up it's time quanta and need to move out so that other process can use the CPU.

syscall (): The currently running process encounters a trap (read, or exit) which would need it to release the CPU. A new process would likely come and use the CPU resources then.

sch(): This schedule() function selects the next process to run on the CPU.

Cosider the Timer to be set to 2 clock$_{ticks}$ and the scheduler to select whatever is next in the queue to use the CPU. Now let us look at the below transition diagram and understand how scheduling mechanism changes the system state.



What's happening in the above diagram?

We initially had three processes with pids as 0, 1, and 2 respectively. Considering that it is also the order of their arrivals, we initially have three process in non-terminated state. The the process with pid 0 has arrived first, it will get to use the CPU first. It is shown in the diagram that the total time it needs to be use the CPU to finish it's execution is 3 $\text{clock}_{\text{ticks}}$ while the Timer is set to 2 clock ticks. So after 2 clock ticks the hardware generates a timer interrupt and now it is the job of the scheduler to allocate the CPU to a new process. The next process to arrive was the process with pid 1. Hence the CPU starts executing pid 1. The Timer is set back to 2 after the new process is loaded.

Now pid 1 needs only 1 clock tick to complete it's execution. Hence it now encounters an exit syscall. The CPU will have to first deal with the trap handler to deal with this but let us just skip the kernel level code executions for now. Once this is done, the scheduler will select the next process to allocate the CPU. The timer is again set to 2 when the new process arrives. **Note that the operation of timer varies from OS to OS. This is a very simple case of timer triggering scheduling.** The now terminated process 1 is no longer placed in the rQ.