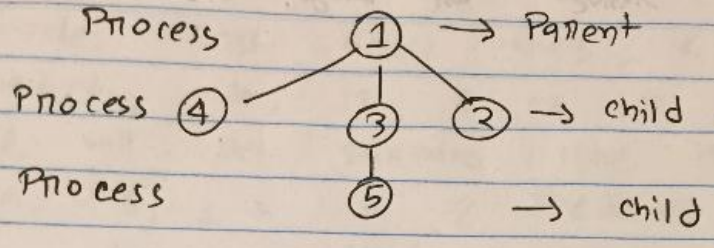
Project



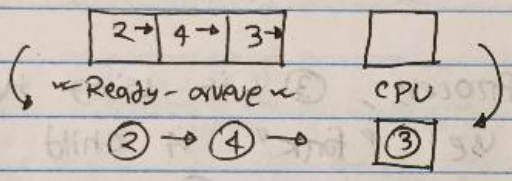

* For the Project, (1) is Parent Process, the original Process, It never uses the CPU or memory.
* We don't need to create it, its already there, It's Pid is 1.

* For simplicity we don't create any process, It's just simulation.
* For child Process we create object of a class & assign Pid with it. like (4) gets Pid = 2, (3) gets Pid = 3, (2) gets Pid = 4,

* Two diff Process can't have same Pid,
* If we delete one Process we can't use the same Pid again.
* Pid can be considered int valve increasing with new Process.

>> A means a new Process is created.

* suppose initially there is only one Process
* when we input "A" we create Process ④ & assign Pid = 2, where Process ① has Pre defined Pid = 1.
* we have a ~~waiting~~ Ready - arrive, ~~that~~ means, Process are waiting in the arrive, to use the CPU, the Process to the front will get the opportunity to use the CPU first.




* Process ③ gets chance to use the CPU.
* Later Process ④ gets chance & later Process ②.
* Right now Process ③ is using the CPU.

>> Q means, the current Process which is ③ has used CPU for some time, we want to take out this Process from CPU. Put this at the end of Ready - arrive & place the next Process in - line to the CPU.

* like if we consider this case :

② → ④ → [③]

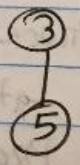
* "Q" we will do like this -

③ → ② → [④]

Ready - queue CPU

* To create a new Process we use "A"
* To " " child of Process " "fork"
* Like to create new Process ④, ③, ② we will use "A"

* when Process, ③ is using the CPU, if we use "fork" A child will be created of Process ③.

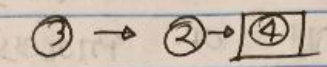



* Every - time we have to look which Process is using the CPU ? — most imp,

* If we use " S space, π " it shows the Process which is using the CPU, & which Process are waiting in Ready queue,

* It will show like this :

"S π"

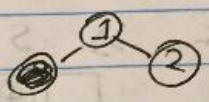



output

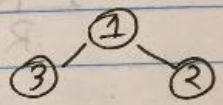
| | | |
|---|---|---|
| CPU : | Pid | 4 |
| R-Q : | Pid | 2 |
| | Pid | 3 |

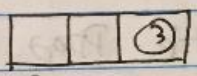
* Let's look at another example_


* >> A




* >> A





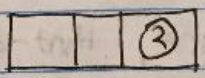
*» S π

| | | |
|---|---|---|
| CPU : | Pid | 2 |
| R-Q : | Pid | 3 |

output of "S π"

* If there are more than one Process in Ready-queue, show them in order.

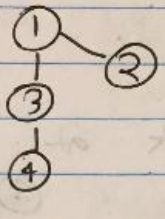
* >> Q



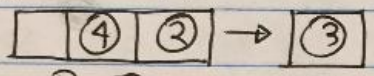


takes Process ③ from CPU & Put it in R-Q, & Puts Process ③ gets chance to use CPU.

* Process ③ is using the CPU,
* If we use >> fork, it will create a child of Process ③.

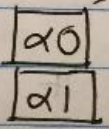



* >> s n




| | |
|---|---|
| CPU : | Pid 3 |
| R-Q : | Pid 2 |
| | Pid 4 |

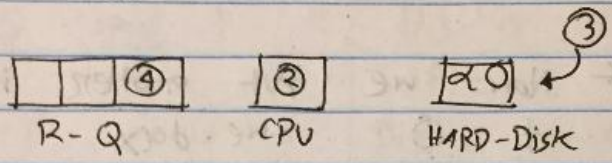
* In the beginning the Proo asks "How many hard-disks does the comp have"?
— Let's say the user inputs '2';
— So we have two hard-disks.




* Right now Process ③ is using the CPU.
* >> d space Hard-disk num space file name
* d 0 asd.txt
* It means the Process which is using the cpu, which is ③ nearests hard disk number 0 & it wants to read

a file named "asd.txt".

* So Process ③ comes out from the CPU, & asks hard-disk 0. Process ② gets chance to use the CPU, Process ④ is still in Ready-queue.




* Again, we don't create any file, its just any random file name.
* we have another queue, called I/O queue, which is the Processes waiting in I/O queue to use the hard-disk if there are more than one Process.

* Make sure, if any Process is in R-Q & wants to use the CPU, if the CPU is free, Let the ~~User~~ Process use the CPU.
* Make sure the CPU is not idle.

* Make sure, if any Process is in I/O queue, & wants to use the hard-disk, if the hard-disk is free, Let the Process use the hard-disk, make sure its not idle.

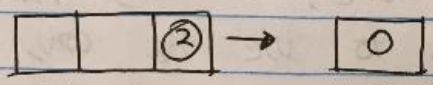
* Now Process ③ is using hard disk 0 to read file "asd.txt".

* Process ② is using CPU, Process ④ is in R-Q.

* Now we put another input like:
d 0 2we.docx

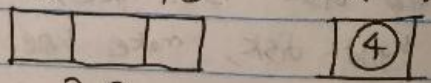
* It means, currently running Process using CPU, which is Process ② is asking hard disk number "0" to read file "2we.docx"

* since Process ③ is already using hard disk 0, Process ② comes out from CPU but waits in I/O queue, to get chance to use the hard disk "0".




I/O queue HARD-Disk "0" currently used by Process ③

* on the other hand, Process ④ gets out from Ready-que, & gets chance to use the CPU, as CPU is idle. R-Q is empty now.



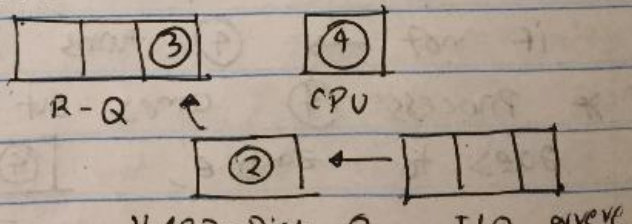

R-Q queue CPU

* Now, we use another input like
» S space i ' — it shows which process are using hard disk & what process are waiting in I/O queue. Show the file name also.

* » S space i

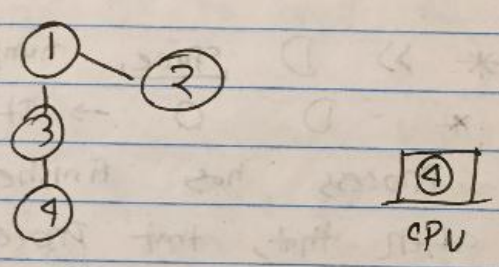
| |
|---|
| Disk 0 : Pid ③ |
| file : 'asd.txt' |
| Disk 0 I/O queue : |
| Pid ② |
| file : "we.docx" |

* » D space number
* D 0 → It means currently running process has finished using hard disk 0, after that, that process goes back at the end of Ready-queue. disk 0
* Suppose Pid ③ was in ~~CPU~~, then the user inputs 'D 0', Pid ③ has finished using hard-disk 0, & goes back to Ready-queue.



* >> "exit" it means the cuently runing Process using the CPU terminates, all of its childnen is terminated. Release all the memony with that Process.

Then checks if its Panent has called "wait" on not ? if the Panent Process has called "wait" the Process terminates immediately, if not - then the Process turns into zombie Process.




* suppose Procss ④ calls "exit", all of its childnen will be terminated, checks Panent of ④ which is ③ has called wait on not !
* if it has called "wait" ④ terminates, if not → ④ turns into zombie Process.
* Process ④ comes out from CPU & goes to zombie,


Zombie Process

* The next Process in R-Q gets

chance to use the CPU.

* >> "wait" it means the process using the CPU, is waiting for its child process to terminates,

* suppose, Process ③ is using CPU. has called "wait", it checks if Process ③ has "zombie child",

* If it has zombie child, the zombie child terminates, even here, ④ terminates, as ④ was zombie child of ③, & Process ③ continues using the CPU.

* if Process ③ has no zombie child, it means, none of its child is willing to be terminated. So Process ③ has to wait, It come out from the CPU, goes to waiting state. [③] wait

* once the zombie child terminates by using "exit" the Process ③ goes from "wait" to end of R-Q.

* if more than one zombie - child exists, system use any one of them to re-start the Parent,
* often zombies keep waiting until they get "wait" from Parent

* for the Project, we have "PCB" which means "Process control Block",
* It is just a "vector".
* It stores info of all the Process
* like Pid, R-Q, CPU, Hard-disks
* Parent Pid, child Pid,

* Don't need to use any tree structure,
* for simplicity try to use vector,
* Pid is int value,
* Parent - child relation using Pid!

* if Process ③ asks "wait" & it has no child, it should tell you Process ③ has no child, no need to wait!
* If Process ③ asks "exits" but its child ⑥ is waiting in I/O queue, it should remove ⑥ from I/O & both Process ③ & ⑥ terminates.