

# Gestion des ressources par un SE.

Processus : entité

## ENTITÉ

- instance d'un programme chargé en cours d'exécution
- chargé en mémoire (RAM), qui s'exécute
- possède ses propres ressources

Important  
- peut être simple ou bien composé par plusieurs processus. (PID ou PPID)

\* Cycle de vie du processus :

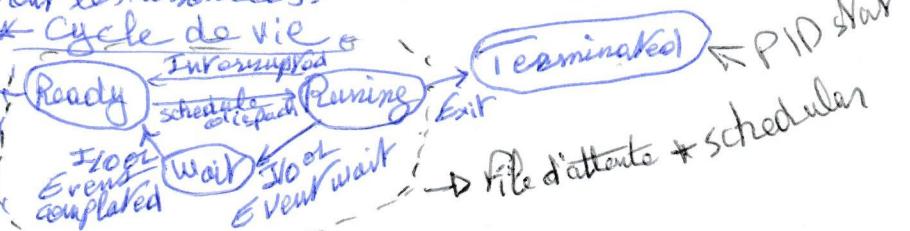
\* Création: lorsque un programme chargé à la mémoire pour s'exécuter.

\* Prêt (Temps d'attente) dans l'état prêt le processus a ses ressources nécessaires pour s'exécuter, mais il se place dans (file d'attente) Queue pour son tour.

\* Exécution: le processus passe à l'état d'exécution sur le processeur et dernière utilise le Temps CPU.

\* Attente: si le processus nécessite des ressources externes par exemple, il peut entrer en état d'attente. Le CPU exécute d'autres processus en attendant.

\* Fin: soit terminé ou interrompu, le système libère toutes les ressources.



\* PCB: contient tout les informations pour gérer et suivre l'exécution d'un processus.

- chaque processus a son propre PCB.
- créer au l'état [création].

\* File d'attente (Queue): le planification du processeur de fait entre deux états, chaque état a sa propre queue.

Prêt - Bloqué - En exécution

\* Table de processus: contient le processus Id (PID) et son PCB.

- permet à l'OS de savoir les processus actifs, leurs priorités, ressources en cours d'utilisation et located in PCB.

- l'ordonnaseur l'utilise pour accéder aux processus éligibles pour être exécutés.

[ - Table des processus référence tous les PCB  
- files d'attente regroupent certains PCB.  
- PCB contient toute information sur Processus ]

\* Traitement des processus et leur enfant.

\* PIP processus auquel de P1.

- chaque processus a son propre PIP, PCB  
- scheduling sont les même traité indépendamment.

- Priorité des processus sont indépendants.

- Parent peut attendre ou recevoir des signaux.

- child peut hériter certaines ressources mais gère indépendamment.

PPID (faut référence à son parent)  
- Différence en PCB.

Remarque :

Tous car se regroupe en ordonnancement

# Scheduler

## ordonnancement (qui regroupe tous les systèmes qui ont à faire de procédures).

- entité
- mécanisme de gestion des processus
  - attribue à chaque processus les ressources nécessaires pour exécuter
  - \* Des Algorithmes qui suivent différents stratagies en fonction du type du système et la priorité du processus.

### Two main types

#### Prematif

#### Non-prematif

Interrrompre un processus en cours d'exécution, pour exécuter un autre

- Round Robin (RR) ✓
- Shortest Remaining Time First (SRTF) ✓
- Ordonnancement par priorité prémetitive ✓

#### Prematif

#### Non-prematif

Processus ne s'arrête que lorsque il se termine ou passe à l'état bloqué.

- First Come - First Served (FCFS) ✓
- Shortest Job First (SJF) ✗
- Ordonnancement par priorité non prémetitive ✓

## évaluation de performance d'ordonnancement

module d'évaluation. - Repose sur des indicateurs pour mesurer l'efficacité et optimiser la gestion des ressources.

### But:

Temps d'exécution

Temps total entre création et fin du processus.

located in PCB or Table de suivi de processus.

- doit pas être confondu avec temps CPU / CPU Burst.

$$T_{turnaround} = T_{fin} - T_{arrivée} (Arrival)$$

### Temps moyen d'exécution

moyenne de temps d'exécution de tous les processus

$$Prof(méthode) \quad AverageTurnaround = \frac{\sum Turnaround}{N \text{ nombre processus}}$$

Local: calculé par le scheduler ou un module d'évaluation.  
utilise les temps d'exécution stockés pour chaque PCB.

But: Mesure l'efficacité du scheduler à minimiser l'attente CPU.  
Indicateur clé pour réactiver les applications interactives.

### Temps moyenne d'attente

moyen de temps d'attente de tous les processus.

$$AvgWaiting = \frac{\sum Waiting}{N \text{ n processus}}$$

- calculé par le scheduler ou module d'évaluation.

### But:

Comparer l'efficacité des différents algorithmes  
optimiser le temps réponse moyen.

### Rendement du système (Throughput / Débit)

nombre de processus terminés [Processus Par Second]

Par unité de Temps

$$Throughput = \frac{\text{Nombre Processus Terminés}}{\text{Temps Total}}$$

- Mesuré par le noyau & scheduler

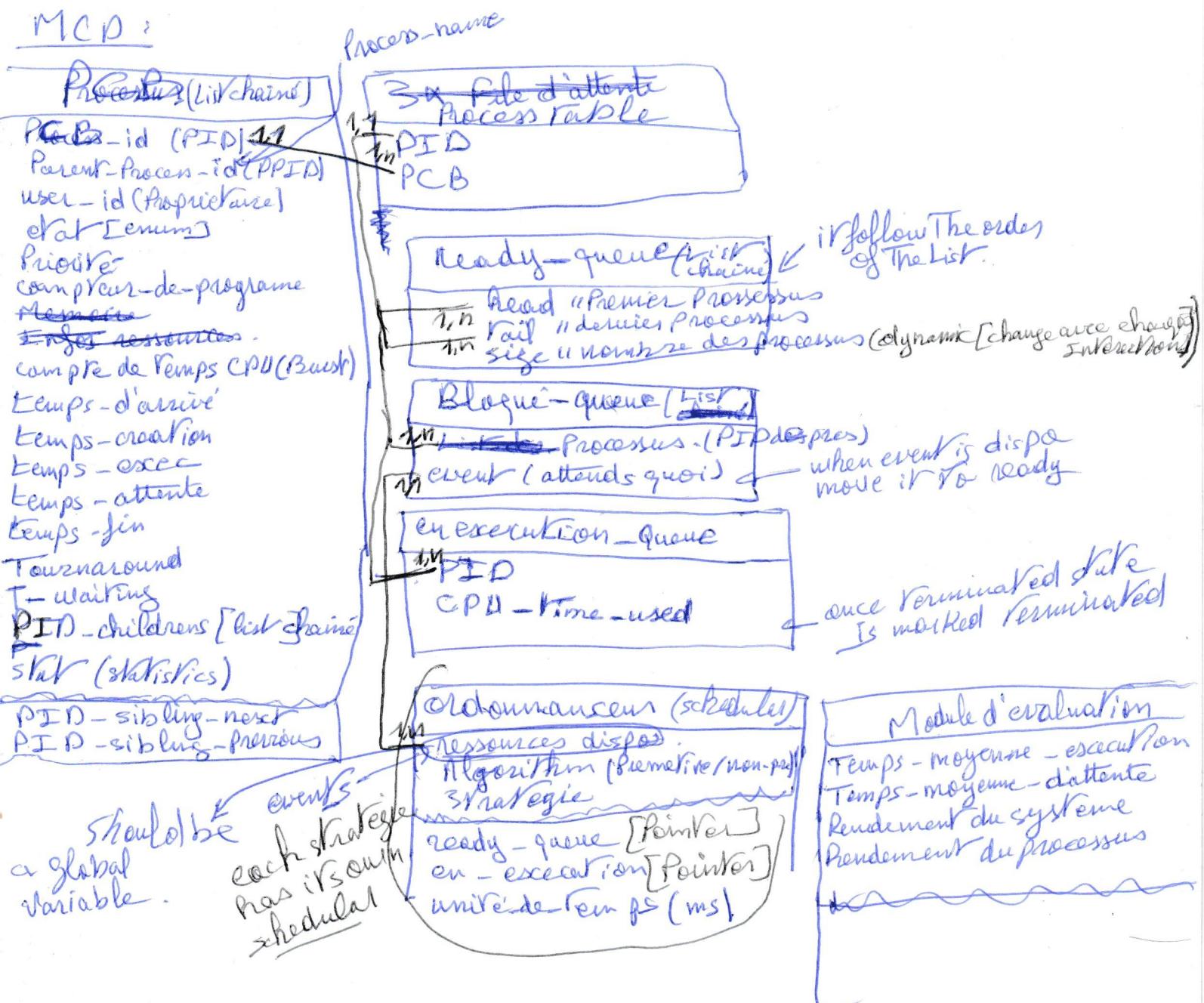
- Indicateur clé pour serveurs et systèmes multi-utilisateurs.

Rendement des processus : taux d'utilisation du CPU

## Définition des entités

Processus, Processus-enfant  $\rightarrow$  PCB, PCB-enfant, file d'attente, Table de processus, ordonnanceurs, [Ressources (chaque ressource entité)], module-d'évaluation.

### MCP :



## \* Round Robin :

- + Quantum fixe
- + File circulaire (chaque processus reçoit un quantam égal)
- + Préemptive (interruption quand quantum expire)
- + FIFO : first in, first out.

## \* SRTF (shortest Remaining Time First).

- + Exécute le processus qui a le temps min.
- + Préemptive : Interruption si un processus arrive qui a moins de temps par rapport au temps restant
- + Besoin d'estimation de temps d'exécution.

## \* Par Priorité (preemptif)

- + Processus avec plus haute priorité s'exécute
- + Préemptives si un processus avec plus haute priorité arrive
- +

## \* Par Priorité - (non-preemptif)

- + Processus avec plus haute priorité s'exécute
- + Pas d'interruption pendant l'exécution.
- + Attend la fin du processus.

## \* First come ... (FCFS)

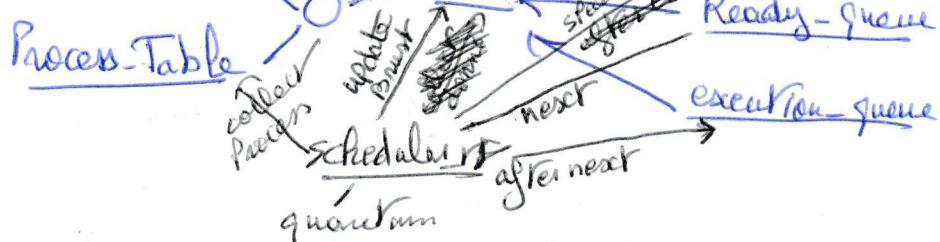
- + Premiers arrivés, premiers exécutés
- + Non préemptif.

First in, first out.

## \* SJF (shortest Job First)

- + Exécute le processus avec durée totale la plus courte.
- + Choix où le CPU devient libre.
- + Non préemptif, pas d'interruption.

# PCD-ID



R/R

## 1- Process Table :

Prend chaque PCD, attribue PID  
lui attribue un PID structure  
L'ajout au ready queue Process Table  
commune

## 2-scheduler :

Prend les PID, les affect au ready queue après checker  
que les ressources nécessaires sont présent.

- si un process nécessite une ressource absent l'affect au blocked-queue
- chaque process prend quantum puis s'effectue
  - \* fait verifier pour chaque interaction
  - \* si apres vérification Burst = 0 le PID marked Terminated in PCB

ordre par temps - dans le

blocked queue et son

Burst's update

Just avant chaque execution:  
 - sched vérifie les états de procs  
 Most Terminate and  
 Delete from queues affect a ready queue  
 - sched vérifie ready queue

Début → PCD

→ Process Table

→ scheduler

→ Ready-queue

→ Blocked-queue

→ Execution queue

→ Terminated



Début main  
 charger\_list\_Processus();  
 start\_scheduler();

fin main

Début statscheduler :

- Prendre Premier element List.
- Vérifier du PCD Burst, si null Terminated,

si Burst = 0 :

- Mark Terminated.
- Delete from all queues.

sinon

- check ressources
- si present Time Trackers
- nexecute - queue.
- Pour le quantum Time, Track-Time.
- Execute instruction par instruction

si instruction necessite ressource

- check ressource

si Present

continuer free(ressources) after usage.

sinon move Blocked

sous tract le temps restant du Burst.

[chaque ms, blocked list need to check ressource]

if ressource is available

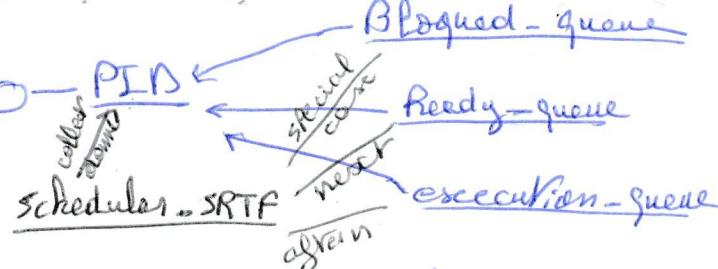
add to ready queue

else

continue.

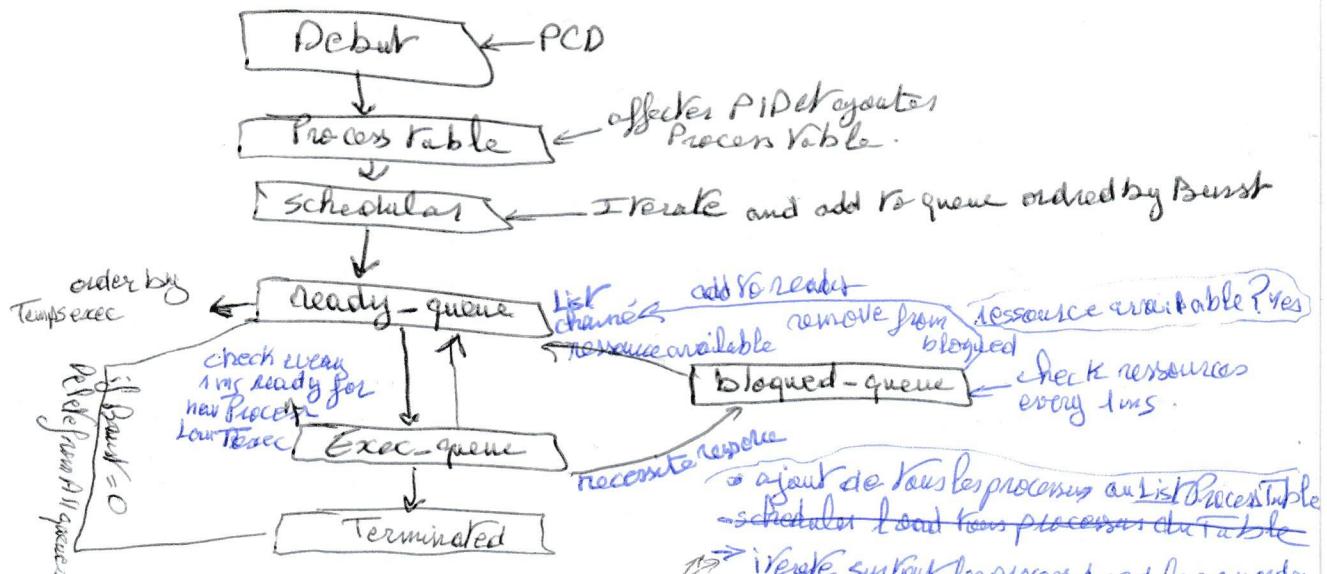
PCD-Id

Process-Table



1- ProcessTable

- 2- Scheduler :
- Iterat over PID's PCD, add them to ready queue based on Burst.
  - Debut du processus mains burst.
  - Chaque 1 ms cheque : Iterate over PID's PCD.
    - if a processes that has less Burst, move the current to ready-list.
    - if the process need resource
      - Instruction
      - check available resource
      - if present, continue
      - else move to blocked



Debut main:

charger-list-processus();  
start-scheduler();

fin main.

Debut start-scheduler

- Iterate sur PID's PIB.

- add min.

- get minimum ~~min~~ Temps execut. (if = 0, mark it Terminated)

- check if it's not already Present in ready.

- if no add it.

- move to process + TimeTracker

- check every 1ms for low Burst.

- if instruction need resource, check available resources

if present &

- Allocate it

Par

- free resource

- not present

- move process to Blocked

Redo - subtract Time from Burst

Absent:

move to blocked

queue.

continuer (redo)

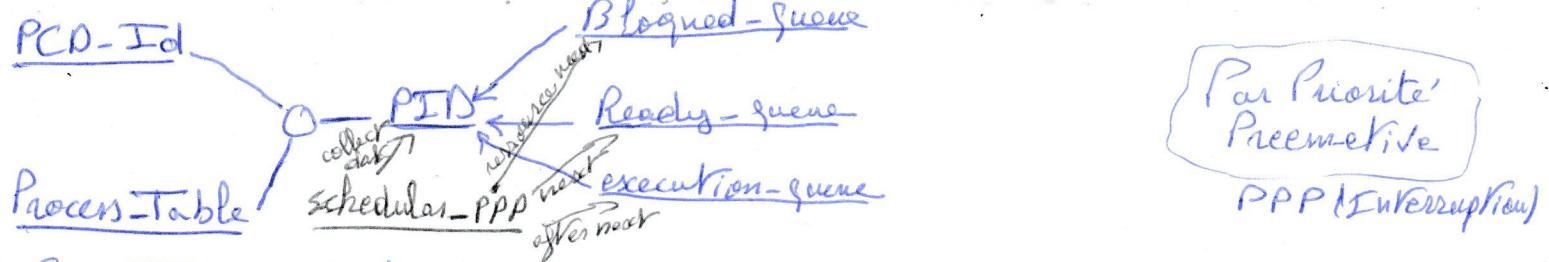
- subtract Time from Burst

Redo - move Mark process Terminated

remove from all queues.

Blocked state → ready-queues.

check every 1ms.



## 1) Process Table

### 2) scheduler

- Iterate PID's PCD. & check burst is terminated. add them to queues in order of Priority.

- Debut process.

check PID's PCD for prior process from ready.

if found  
move this one To ready  
in its proper place (order)  
redo

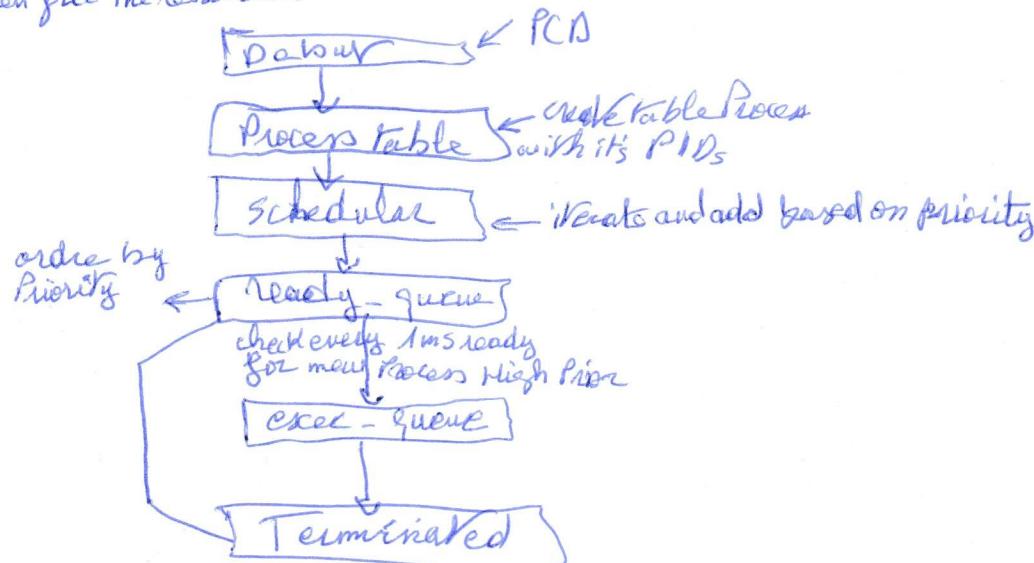
if no  
Pass

if instruction need resources  
check availability

if available if no  
Locate it move this to Blocked  
and Allocate redo

execute instruct

Then free The resources



Debut main  
chargerTable-process();

start-scheduler();

fin main;

Debut start-scheduler

- Iterete over PID's PCD

if Burst=0:

Mark Reminated

remove from queue

- with every iteration add the Higherprior

- move first elem (high prior) to exec-queue

- track time

- check every 1ms for Higher prior .

- instruction need resources

resource available

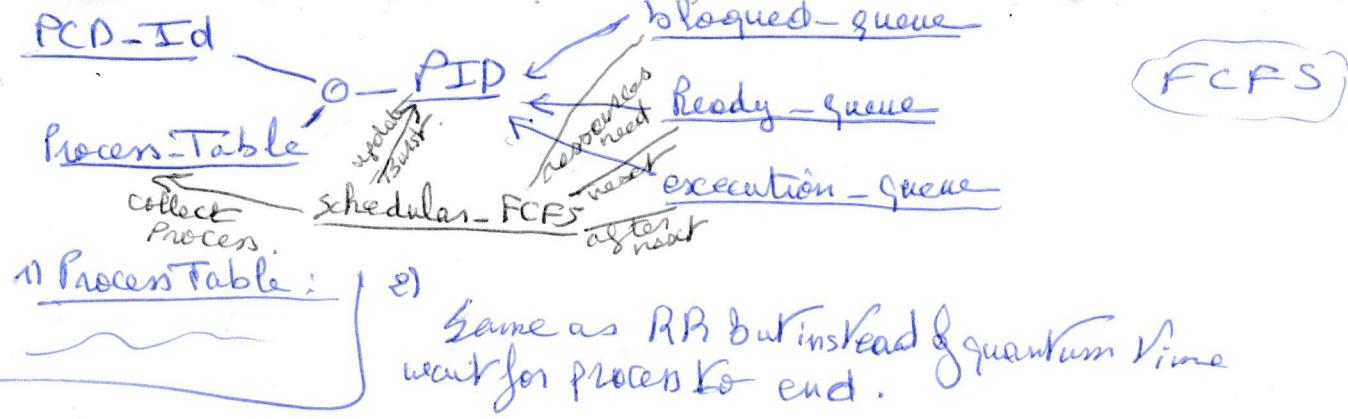
Allocate it

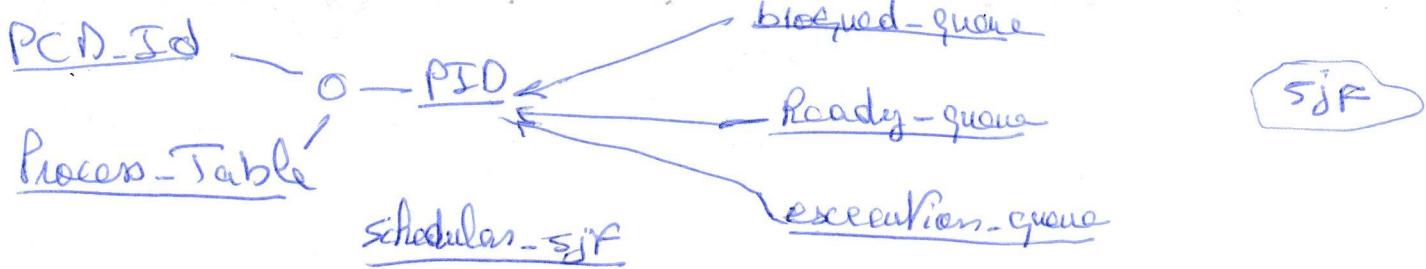
- do instruct

- free resources.

if unavailable  
- move to Blocked queue

- redo





Same as as SJTF without  
Interruption.

