System (4)

Mise en RAM de l'emplacement théorique de la FAT

## Scheduling:

→ Ordonnancement → Planification
↳ Tâches
↳ Programmes

## Actuellement:

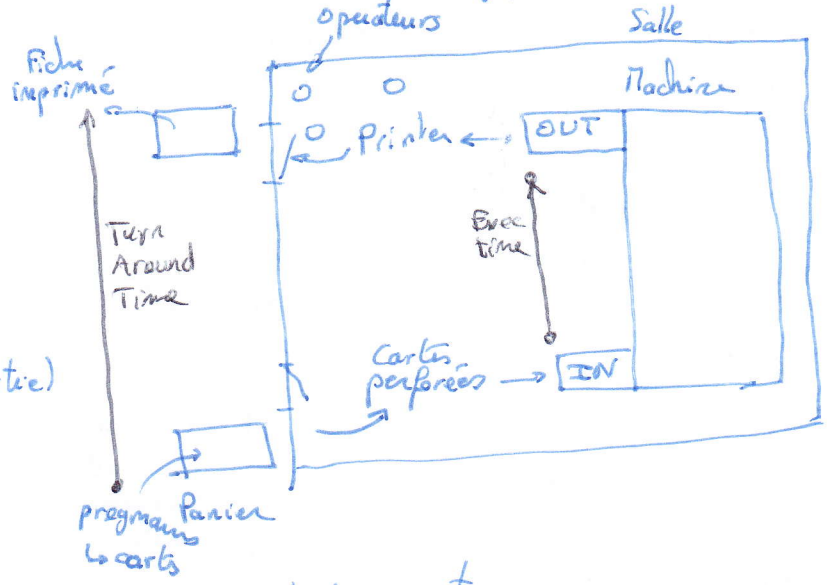
→ TPs partagés
↳ Plusieurs progmas simultanées
↳ Ne s'arrête pas.

## Scheduling

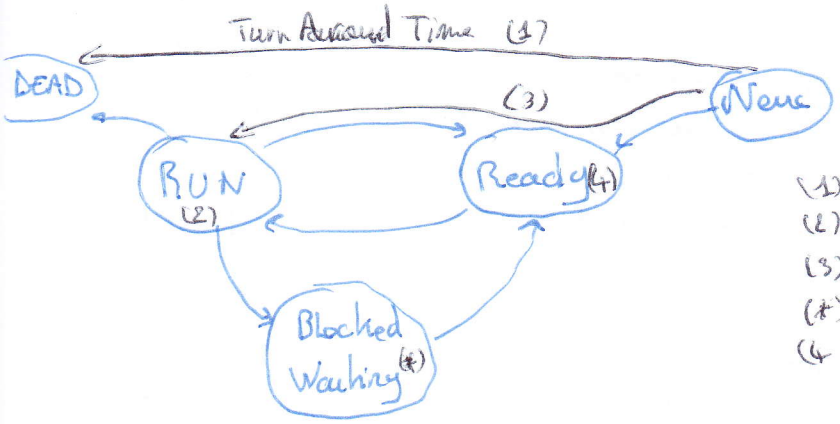
* Maximiser l'utilisation du CPU
* Débit important
* Temps d'exécution d'un programme
* Turn around time (Entre soumission et sortie)
* Eviter les situations de famine / starvation

## Historique

↳ Badge systèmes
↳ Pas OS
↳ Monotâche.
↳ Donne un program
↳ Wait
↳ Result.




But des opérateurs :
↳ Finir les programmes
↳ Optimiser le temps

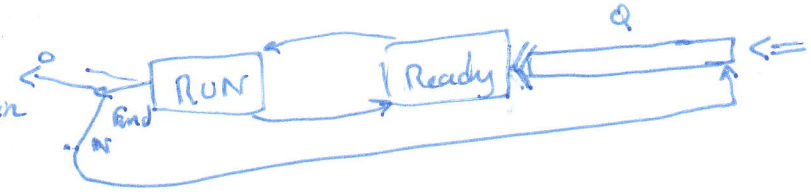



(1) TAT
(2) Exe time : Temps in Run state
(3) Starting Time
(*) On a aucune influence pour le temps d'exécution
(4) Waiting time

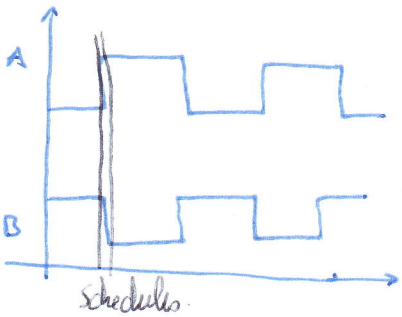
## Système Préemptif:

↳ Arrête la tâche pour reprendre la main
↳ Timer
↳ A chaque tic, on crée une interruption qui permet au kernel de reprendre la main
↳ Scheduler
↳ Algo en temps constant (algo sys)
↳ Equitable ≠ Egale
↳ Donner suffisamment de tps et de ressources à chaque program

## Round Robin

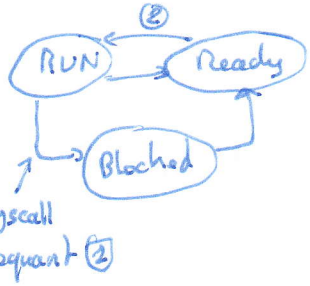


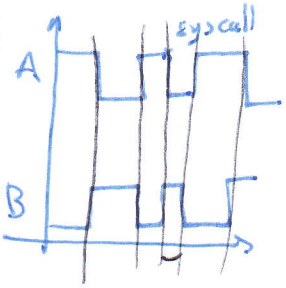

* Time Quantum




* Présence de caches
  - ↳ Pagination
  - ↳ Mémoire
  - ↳ Instruction

⇒ Système Déterministe




⇒ 2 types de tâches :

* Calculatoire : Gourmande en ressource CPU — Plus longtemps ↗ Quantum / Moins souvent ↘ Schedule
  Non interactive
* Sociale / Interactive : Attends une interaction — Moins longtemps ↘ Quantum / Plus souvent ↗ Schedule
  utilisateur / réseau / autre programme
  Elle exécute beaucoup de syscall

Bottleneck (Goulot d'étranglement)
- ↳ Non interactive : Vitesse CPU (CPU Bound)
- ↳ Interactive : Syscall bloquant, Vitesse I/O (I/O Bound)

03/12/19

Stack :
- variables locales
- valeur de retour
- Appels de fonction

Registres : %rsp, %rbp

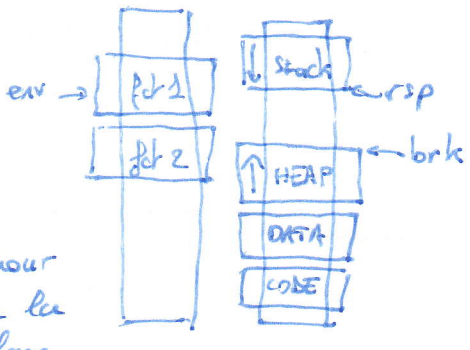
Instructions : push, pop

Ex : push %rax
pop %rbx

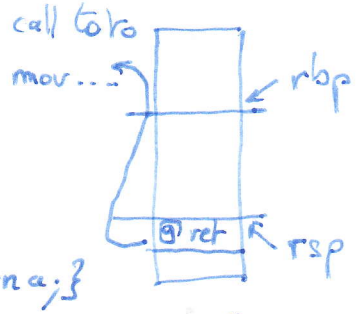
%rbx : base pointer ⇒ permet de save le ptr de la pile.
⇒ pointer de la stack frame courante
stack frame : espace mémoire sur la pile pour les variables local

%rax : registre de retour sur x86

Fonctions :
Instructions : call, ret
↳ Permet de faire des appels à fonction sur la stack.




call push une adresse de retour sur la pile. Pour récupérer la valeur de retour, on la place dans la stack. Cette dernière ne bougera pas par rapport à rbp



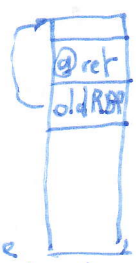

call toto
mov %rax, %rbx

toto : // long toto { long a = 12; return a; }

```
sub $16, %rsp
mov $12 ....
add $16, %rsp
ret
```

prologue
epilogue

```
push %rbp
mov %rsp, %rbp
sub $16, %rsp
//....
mov %rsp, %rbp
pop %rbp
ret
```




x leave
// instruction