Chapitre 7

Ordonnancement de l'UC

- Ordonnancement sans réquisition de l'UC
- Algorithmes
- Ordonnancement avec réquisition de l'UC
- Algorithmes
- Ordonnancement avec priorité de l'UC
- Ordonnancement de Disque
- Algorithmes d'ordonnancement de Disque

⊡ But : maintenir un taux d'utilisation élevé de l'UC

⊡ Idée première : recouvrement des E/S

⊡ Quelques mesures :

– le **taux** d'utilisation de l'UC :

- théorique : 0 à 100%

- pratique : 40 à 95%

– le **débit**

- nombre moyen de programmes utilisteurs traités par unité de temps

- le taux augment le débit

- le débit ne tient pas compte de la taille des programmes

– Optimisation de ces valeurs revient à optimiser :

- temps de **traitement moyen** d'un système de tâches :

→ la moyenne des intervalles séparant la soumission et la fin de la tâche.

- temps de **traitement total** d'un ensemble de processus donné

- temps de **réponse maxi** : soumission - réponse à une requête (important dans les systèmes T.R.).

⊡ Pour optimiser ces temps, il ne suffit pas de recouvrir les E/S par des calculs.

⇒ l'odonnanceur doit choisir le processus qui améliore les critères.

⊡ **Définitions** : pour une tâche

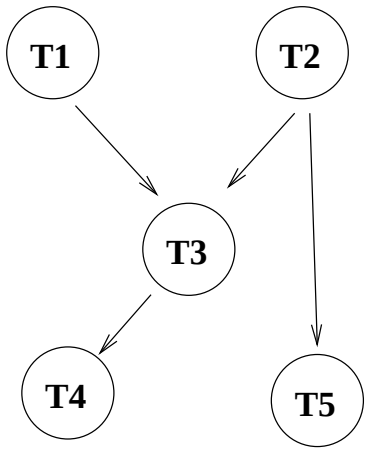
- Ti : la tâche ;
- τi : sa durée d'exécution ;
- ti : sa date d'arrivée dans la file des prêts
- **assignation** : la description de l'exécution des tâches dans le système (mono / multi processeur) construite par l'odonnanceur → un $ordonnancement$

⊡ Un exemple :

Ti	τi	ti
$T1$	1	0
$T2$	2	0
$T3$	1	0
$T4$	1	0
$T5$	2	0

⊡ Exemple : 5 tâches, 2 processeurs

	ti	τi
T1	**1**	**0**
T2	**2**	**0**
T3	**1**	**0**
T1	**1**	**0**
T5	**2**	**0**



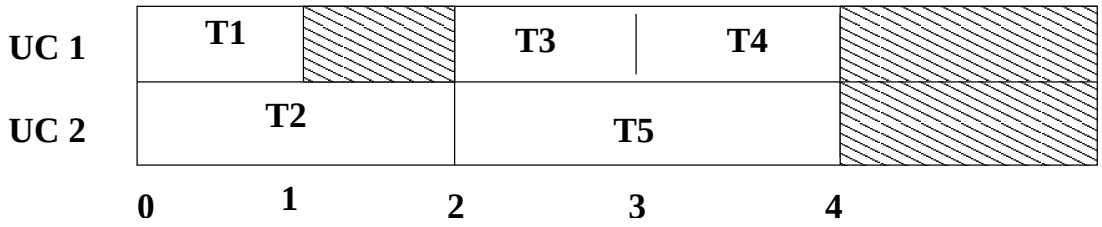


Diagramme de **Gantt** : une représentation d'une assignation

Remarques :

- On ne représente pas les temps de commutations (très faible).

- τi peut représenter le temps exact d'exécution d'une tâche si aucune autre n'existe dans le système.

- Cette valeur est incalculable par avance ; on s'en sert a posteriori pour analyser un système.

- Le τi pris en compte est une borne sup de la durée (utile dans les systèmes TR)

- Le calcul d'une estimation de τi est possible (V. biblio).

⊡ Les algorithmes d'ordonnancement pratiqués sont classés par :

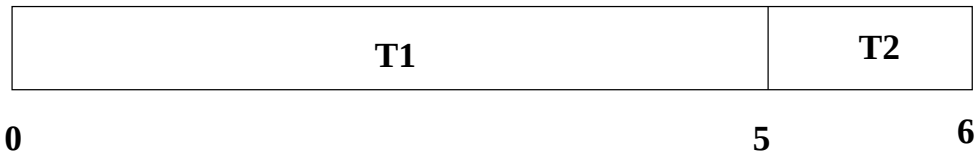
avec (sans) **réquisition**, avec (sans) **priorité**, avec (sans) **contrainte de précédence** (et sans tenir compte des it°).

⊡ Exemple (T1 et T2 indépendantes, sans synchro) :

Ti	τi	ti
$T1$	5	0
$T2$	1	2

Sans réquisition : $tmoyen : \frac{5+(6-2)}{2} = 4,5$

L'assignation :



Rappel : $tmoyen$ = la moyenne des intervalles séparant l'entrée et la fin effective d'une tâche.

- Même exemple <u>avec</u> réquisition : $tmoyen : \frac{6+(3-2)}{2} = 3,5$

7.1 Ordonnancement sans réquisition

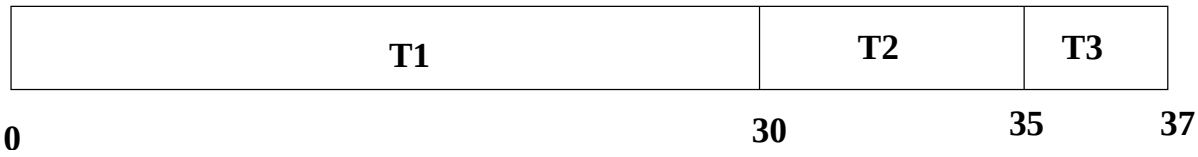
7.1.1 ordre = dans l'ordre d'arrivée (FIFO)

Méthode souple mais ne minimise pas le temps moyen.

Ti	τi	ti
$T1$	30	0
$T2$	5	ϵ
$T3$	2	2ϵ

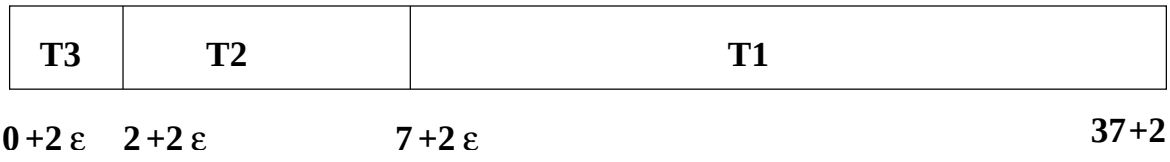
$$tmoyen : \frac{30+(35-\epsilon)+(37-2\epsilon}{3} = 34$$

L'assignation :



La tâche $T1$ avec un temps long bloque la file car elle est arrivée en avance.

On peut faire mieux avec $tmoyen : \frac{2+7+37)}{3} \approx 15,5$



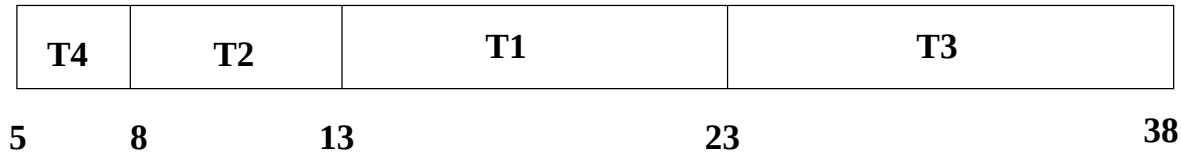
7.1.2 Dans l'ordre inverse des temps d'exécution : PCTE (SET)

Exexmple : A l'instant **5**, l'état de la file des prêts est :

Ti	τi	ti
$T1$	10	0
$T2$	5	2
$T3$	15	3
$T4$	3	4

$$tmoyen : \frac{(8-4)+(13-2)+(23-0)+(38-3)}{4} = 18,25$$

L'assignation :



On peut démontrer que l'algorithme PCTE **minimise** le temps moyen pour l'ensemble des algorithmes d'ordonnancements **sans réquisition**.

$\Rightarrow$ Cette proposition n'est pas vraie si les tâches entrent au cours d'une assignation (à moins de refaire l'assignation à intervalle X donné).

$\Rightarrow$ PCTE est intéressant pour permettre de comparer, après coup, les perfomances des algorithmes réellement implantables, aux valeurs min qu'il est possible d'obtenir (avec des estimations).

7.2 Ordonnancement avec réquisition

- Nécessaire dans un système à temps partagé (ou TR)
- Un SE doit fournir une machine virtuelle ;
- Les tâches longues ne doivent pas bloquer les autres ;
- Question d'équité ;

7.2.1 Méthode Tourniquet (Round Robin)

- On utilise des It°, commutations, Horloge ;
- On choisit un intervalle **= Quantum**

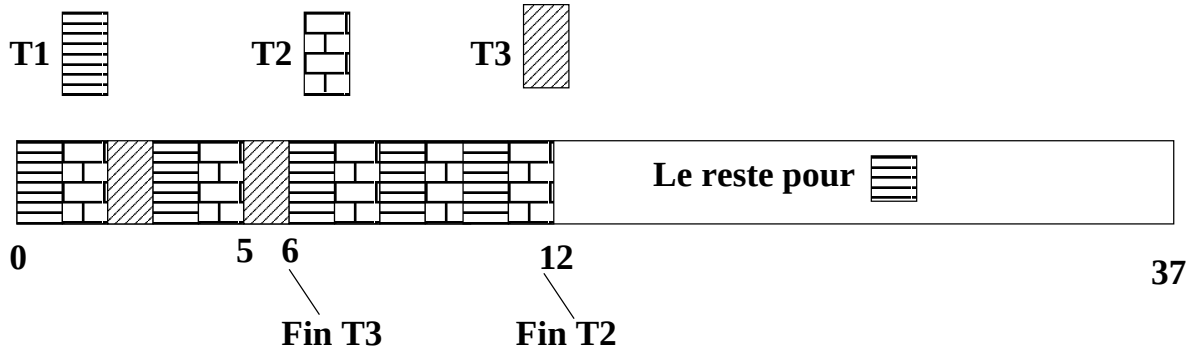
  Exemple : dans Vax/VMS, le quantum est multiple de 10ms (intervalle entre 2 tops d'horloge = 10ms) ; par défaut, quantum = 200ms

- File d'attente circulaire (ou double file) ;
- L'ordonnanceur choisit une tâche, le distributeur (*dispatcheur*) lance son exécution ;
- Si la tâche termine avant le quantum → remise à 0 ;
- Sauvegarde du contexte, cumul des temps, choix du prochain, commutation ;

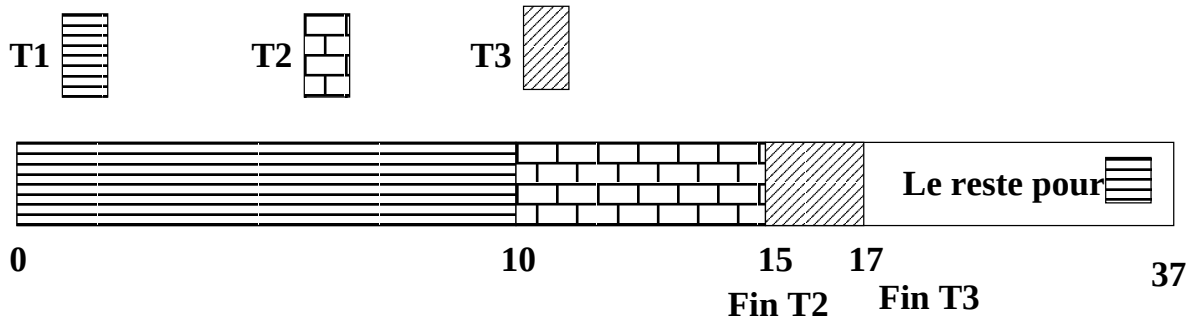
⊡ Exemple :

Ti	τi	ti
$T1$	30	0
$T2$	5	ϵ
$T3$	2	2ϵ

L'assignation (Quantum = 1) : tmoyen = $\frac{37+12+6-3\epsilon}{3} = 18,33$



L'assignation (Quantum = 10) : tmoyen = $\frac{37+15+17-3\epsilon}{3} = 23$



⇒ Importance du Quantum :

- si quantum trop grand : temps de réponse augmente ;

- si quantum trop petit : trop de commutations !

⇒ Une idée de CDC6000 : 10 mots d'état en mémoire

⊡ NB : Round Robin avec une double file des prêts.

7.2.2 Plus court temps d'exécution restant : PCTER

⊡ Principe : à chaque réquisition, on calcule le temps d'exécution restant et on choisit la tâche dont le TER est minimum....

7.3 Ordonnancement avec priorité

⊡ Principe : égalité pour tous (équité)

→ Mais pour des raisons d'efficacité, les tâches systèmes sont plus prioritaires ;

→ d'autres devraient fournir des résultats avant certaine date.

⇒ Solution : attribution de priorités ;

⇒ 2 méthodes :

- Priorité **Fixe** : problème de famine si priorité faible

- Priorité **Variable** : recalcul des priorités selon l'avancement des exécutions.

e.g. : on peut ajouter +1 à la priorité des tâches en attente (et/ou enlever +1 de la priorité de ceux qui s'exécutent).

⊡ Exemple VAX/VMS :

- Ordonnancement avec priorité

- Tâches normales (user) et tâches systèmes (TR)

- 32 niveaux de priorité : 0 (min) .. 15 : user, 16..32 : TR.

- Priorités modifiables

- UC réquisitionné à l'arrivée d'une tâche plus prioritaires (surtout tâche TR) ;

- Les priorités 0..15 évoluent à l'arrivée de certains événements. Chaque événement apporte une valeur (positive ou négative) de priorité (e.g. fin E/S, libération d'une ressource, fin d'une entrée/sortie depuis/vers un terminal, création d'un processus, etc.)

- L'exécution d'une tâche fait baisser sa priorité (limitée à un seuil) ;

- Round Robin pour la même classe / valeur de priorité ;

⊡ Exemple OS2 : proche de VAX/VMS :

- augmentation si phase de calcul courte (avec des E/S).

→ Truc : ajout fictif d'une lecture de temps en temps !

7.3.1 Ordonnancement avec plusieurs files de priorité

- Permet de réduire le temps d'attente d'une tâche sans modifier le quantum ;

- Les files ont des N° de priorité fixes ;

- Les tâches vont de file en file ;

- Le temps de l'UC peut être partagé entre les files selon les priorités ; par exemple, la file la plus prioritaire prendra 70% du temps de l'UC.

- Les files peuvent être gérées par des algorithmes différents (adapté au type des tâches de la file) ;

⊡ Autre variante :

- Ordonner hiérarchiquement les files

- On n'exécute une file que si les files de priorité supérieure ont été traitées

→ famine sauf si les priorités sont modifiées en permanence et les tâches changent de file ;

⊡ Cas d'UNIX : (**préemptif** = tps partagé avec réquisition)

- Un processus - une priorité - une file

- Processus *utilisateur* et *système*

- Les processus systèmes sont toujours prioritaires

- Dans chaque classe (user, system), différents niveaux de priorité = file différentes

- Les priorités évoluent dynamiquement

- Quand un processus est mis en attente, sa priorité est recalculée en fonction de la raison de l'attente

→ but : diminuer les blocages et favoriser les processus qui au réveil se terminent et libèrent rapidement des ressources ;

→ un processus en attent de la fin d'une E/S avec le disque sera plus prioritaire que celui qui demande une E/S avec le disque ;

→ en plus, le demandeur a besoin d'un tampon que le premier libère ;

- A chaque it° Horloge (1s sous sysV), la priorité d'un processus prêt est augmentée en fonction du temps passé dans sa file (changement de file possible) ;

7.4 Ordonnancement de Disque

- L'ordonnancement de l'UC est crucial ;

- L'ordonnancement de disque affecte les performances du système

- Les programmes sont chargés de la Mémoire Secondaire (MS = Disque) vers la Mémoire Centrale (MC).

- Les échanges MS $\leftrightarrow$ MC sont très fréquents ;

- Les supports : bandes (séquentiel), Disques (direct)

- Les ordonnanceurs tiennent compte des caractéristiques physiques (ϕ) du support

7.4.1 Caractéristiques des disques

- 2 types de disques : Magnétique et Optique

- Les disques optiques : grande capacité mais plus lents ;

- Disques Magnétiques : plateau circulaire (double face) + enduit magnétique ;

- Capacités : quelques Ko (floppy) aux débuts .. 500 Go et plus now !

- Les disques de stockage : plusieurs plateaux

- Chaque plateau organisé en :

  - **pistes** (cercles concentriques)

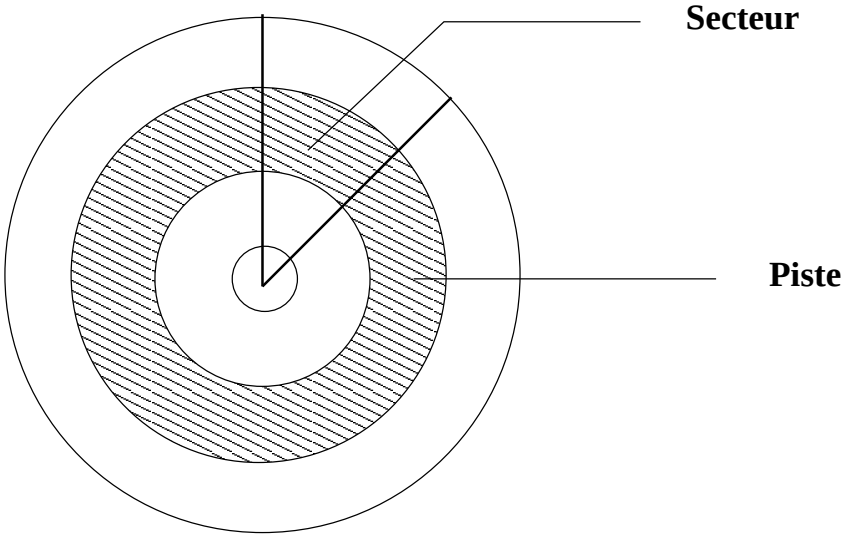
  - **quartiers** d'angle fixe

  - un **secteur** : l'intersection de piste et de quartier

  - un **cluster** : un ensemble de secteurs

  - parfois (sur PC), 1 bloc (= unité de transfert) correspond à un secteur (angle du quartier adapté)

  - Un **Cylindre** : ensemble de pistes sur plusieurs faces qui peuvent être atteints à un instant donné ;



- Le nombre de pistes dépend de la capacité

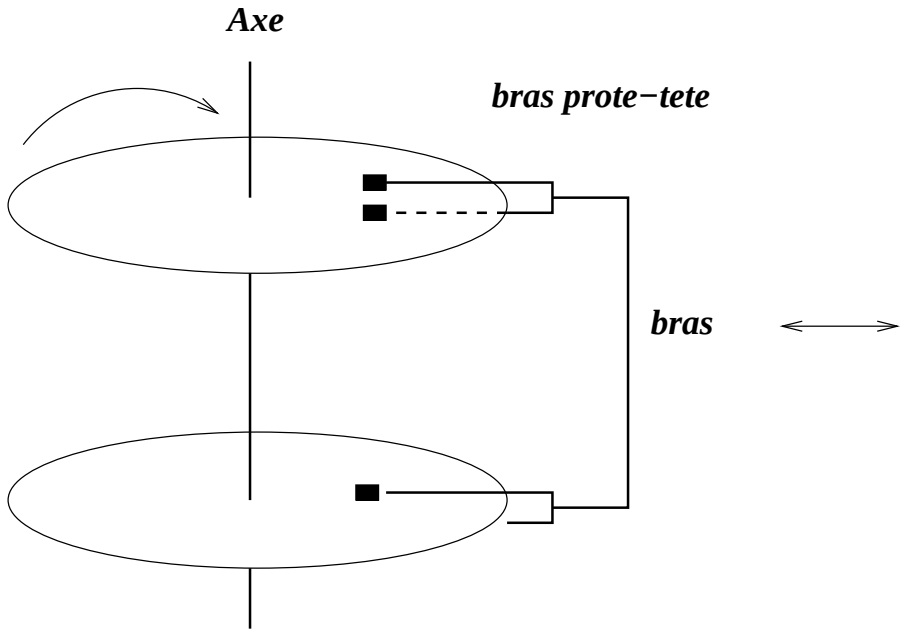
- La division en pistes est faite lors du formatage ;

- Le formatage définit aussi les *octets utiles* (user) et *octets de contrôle* (Fat, n° de secteur , bit de parité, etc.)

- Un bloc/secteur = unité de transfert MC $\leftrightarrow$ MS

⊡ Accès

- bras, bras porte tête, une tête par face, mouvement horizontal du bras, moteur ; ...



- Vitesses de rotation actuelles : 3600 à 7200 tours/min (=120 tours/sec !) ;

- Sur un disque souple (floppy), le film magnétique peut supporter le contact de la tête ;

- Sur le disque dur : la tête est à quelques microns de la surface sur un coussin d'air provoqué par la rotation ;

- contact ↔ catastrophe !

- Matériel associé à un disque :

  - un **dispositif de R/W** pour exécuter les mouvements mécaniques + R/W

  - un **contrôleur**

- Le contrôleur (IDE sur PC) peut gérer plusieurs disques ;

- Autre contrôleur : SCSI ;

- L'adresse d'un secteur (4 éléments) :

  - N° du contrôleur (IDE0/IDE1) ;

  - N° du disque (0/1 → master/slave)

  - N° Cylindre (ou piste)

  - N° surface

7.4.2 Eléments d'ordonnancements

⊡ Plusieurs étapes pour faire une E/S :

1. attente dans la file du dispositif ;
2. déplacement du bras sur le bon cylindre/piste ;

   → **temps de recherche**.

   Certains algorithmes regroupent les requêtes dans - d'autres files (→ attente supplémentaire)
3. la rotation amène le bon secteur sous la tête

   → **temps de latence**.
4. transfert depuis/vers MC : → **temps de transfert**

⇒ Le temps *nominal* (pub !) = la somme de ces 3 temps

⇒ Il ne tient pas compte du temps d'attente dans les files.

- Le temps de transfert (MC ↔ MS) est indépendant de l'algorithme d'ordonnancement ;

- Le temps de recherche (bras → cylindre) est + élevé et les algorithmes d'ordonnancement tentent de les minimiser ;

- Le temps de recherche est proportionnel à la différence entre 2 pistes $|i - j|$ dans 2 requêtes successives .

7.5 Algorithmes d'ordonnancement

Algorithmes implantés dans bios/controleur (chipset) ; le driver (pilote) est propre au périphérique.

Ordonnancement dans l'ordre d'arrivée (FIFO)

- Simple, équitable mais très peu performant ;

⊡ Exemple :

- Un disque avec 20 pistes (0 .. 19)
- La tête sur la piste 14
- La file d'attente (selon les N° de pistes) contenant 17, 18, 4, 11, 2, 12
- La tête fait : 14 → 17 → 18 ...
- Le **déplacement** total (nb. pistes) de la tête :

d=(17-14)+(18-17)+(18-4)+(11-4)+(11-2)+(12-2)= 44

⇒ Idée : regrouper les requêtes selon les pistes proches

→ on traite 4, 2, 11, 12, ... → d=30

Ou mieux : 2, 4, 11, 12, ...

7.5.1 Ordonnancement suivant le plus court temps de recherche (PCTR)

- Déplacement bras → cylindre, proportionnel à $|i - j|$

- Une seule file pour les requêtes

- Requêtes regroupées par les pistes proches

- La prochaine requête traitée est celle qui minimise le déplacement de la tête ;

- La file est réorganisée en permanance

→ problème de **famine** si des requêtes concernant des pistes éloignées restent en attente .

7.5.2 Ordonnancement par Balayage

- Evite le problème de famine précédent

- Adapté aux accès fréquents

- On parcourt les pistes dans une direction données et on traite les requêtes rencontrées (e.g. vers l'intérieur)

- Ensuite, la tête change de direction et balaie les pistes vers l'extérieur

- Les requêtes arrivées pendant ce temps attendent le retour de la tête.

⊡ La version de base de cet algorithme : **Scan**

- Variante : **Look** : la tête ne va pas au bout s'il n'y a pas de requête (file circulaire)

- Autres variantes : retour dans une direction en balayant (ou non)

- Version **C-Scan** : file circulaire, retour à une extrémité ;

- Version **C-Look** : file circulaire, retour si pas de requête dans ce sens ;

- Autres versions : tenir (ou non) compte des requêtes arrivées pendant ...

7.6 Ordonnancement selon le temps de Latence

- Rappel : secteur sous la tête

- Si les demandes sont trop fréquentes, on peut traiter en premier des requêtes pour la même piste (cylindre).

- Les files ordonnées selon les secteurs pour réduire le temps de latence.

7.6.1 Algorithme PCTL : le plus court temps de latence

- Sélectionnér les requêtes concernant les secteurs les plus proches de le tête, compte tenu du sens de la rotation

- On associe **une file** à chaque secteur du même cylindre

- La tête sur une pise (cylindre), on traite les requêtes de cette piste dans le sens de la rotation sans tenir compte de l'ordre d'arrivée

⇒ Beaucoup de files : une par secteur du même **cylindre**

⇒ Stratégie facile à implanter ; performances proches de l'optimum ;