Chapitre 03:

Gestion des tâches / Ordonnancement

(Scheduling**)**

1. Définitions :

- **Tâche/Processus (Thread/Job/Task)** : Entité d'exécution. Instance dynamique d'un programme exécutable.
- **Tâches périodiques :** déclenchées régulièrement car leurs caractéristiques sont connues à l'avance.
- **Tâches apériodiques :** déclenchées par un évènement avec des caractéristiques partiellement non-connues à l'avance.
- **Tâches indépendantes** : ne présentent pas de relation de précédence et ne partagent pas des ressources critiques.
- **Tâches dépendantes** : suivant ou précédant d'autres tâches parce qu'elles se synchronisent ou communiquent entre elles.
- **Séquence valide :** si toutes les tâches respectent leur CT.

2. Paramètres de base d'une tâche :

Une tâche (généralement périodique) se caractérise par les paramètres temporels suivants (avec $0 \leq C \leq D \leq P$) :

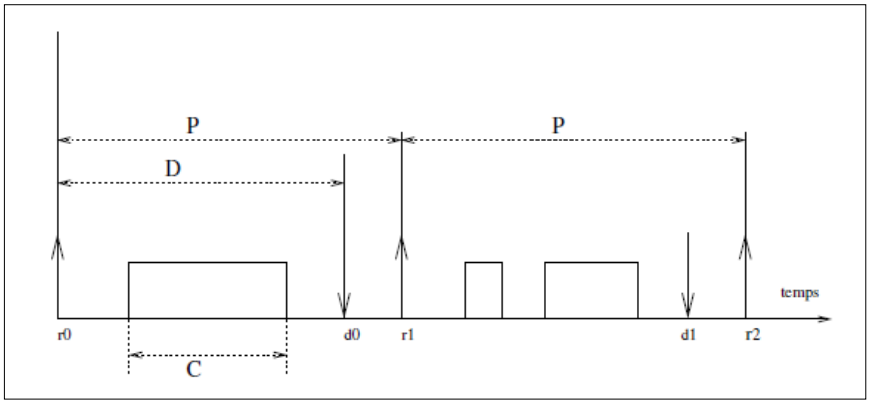


Figure 3.1. Diagramme temporel d'une tâche.

- r_0 : date de réveil, le moment de déclenchement de la 1ere requête d'exécution.
- C : durée max d'exécution,
- c_i : temps de calcul au pire cas (WCET: Worst Case Execution Time),
- D : délai critique (délai maximal acceptable pour son exécution),

- P : période,
- r_k : date de réveil de la k^{eme} requête de la tâche:
- $r_k = r_0 + k*P$, représentée par ↑
- d_k : échéance de la k^{eme} requête de la tâche (à contraintes strictes):
- $d_k = r_k + D$, représentée par ↓
- une tâche périodique à échéance sur requête ⇒ $D = P$, représentée par ↕.

3. Différents états d'une tâche:

- **Élue (courante) :** la tâche est en service, c.-à-d., en cours d'exécution,
- **Prête :** si la tâche dispose de toutes les ressources nécessaires à son exécution à l'exception du processeur,
- **Passive (en attente)** : si la tâche ne dispose pas de toutes les ressources nécessaires à son exécution, ou elle attente qu'un événement se produit pour pouvoir continuer,
- **Inexistante** : l'ordinateur ne sait pas que la tâche est existé (elle est en disque),
- **Bloquée (Existante**) : l'ordinateur sait que la tâche est existé mais elle n'a aucune priorité (hors file d'attente).

Les transitions entre les états prouvent être schématisées comme suit:

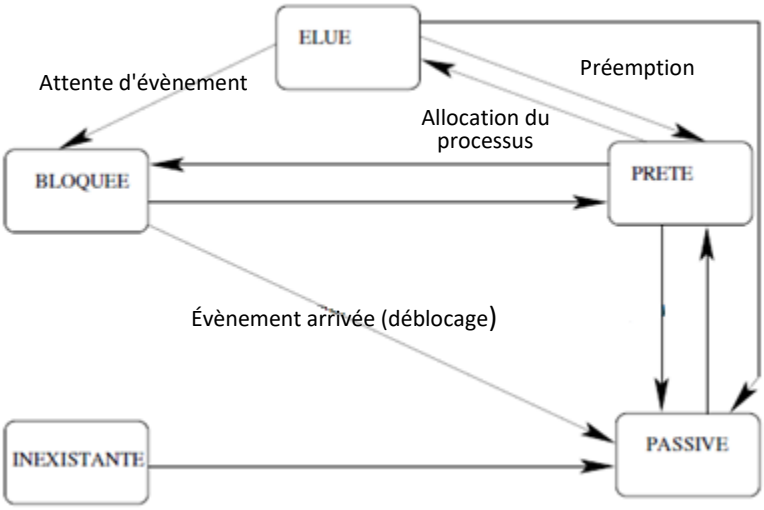


Figure 3.2. Transitions entre les états.

Remarque: Le dépassement d'une échéance est appelée faute temporelle.

4. Ordonnancement temps réel:

Planification de l'exécution des tâches/requêtes de façon à respecter les contraintes temporelles. Il détermine l'ordre d'allocation du processeur. L'ordonnancement a pour deux objectifs majeurs:

- *en fonctionnement normal* : respecter les contraintes temporelles pour toutes les requêtes.
- *en fonctionnement anormal* (*surcharge ou présence d'incidents → tâches supplémentaires suite à des anomalies : alarmes, fautes temporelles...*) : atténuer les effets des surcharges et maintenir un état cohérent et sécuritaire par l'exécution d'au moins les requêtes vitales.

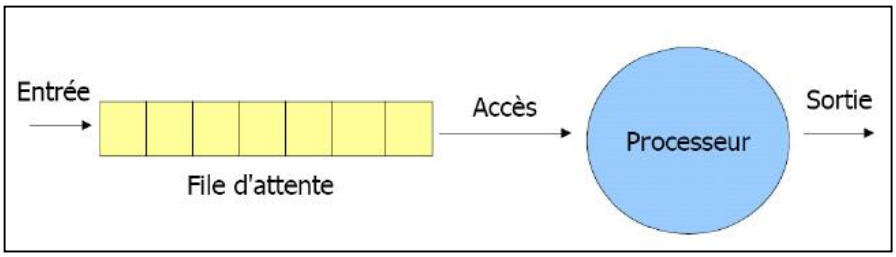


Figure 3.3. Ordonnancement des taches.

Le but de l'ordonnancement est de :

- Valider a priori la possibilité d'exécuter l'application en respectant ces contraintes.
- Permettre le respect de ces contraintes, lorsque l'exécution se produit dans un mode courant.
- En régime anormal : l'ordonnancement doit offrir une tolérance aux surcharges et permettre une exécution dégradée mais sécuritaire pour le procédé.

5. Typologie des algorithmes d'ordonnancement:

5.1. Ordonnanceur hors ligne (Offline): l'algorithme connue la séquence complète de planification des tâches avant l'exécution.

- Avantage:
  - ❖ Très efficace,
  - ❖ Il convient les tâches périodiques.

- Inconvénient:
    - ❖ statique,
    - ❖ rigide,
    - ❖ ne s'adapte pas au changement de l'environnement.

5.2. Ordonnanceur en ligne (Online): capable à tout instant de l'exécution d'une application de choisir la prochaine tâche à ordonnancer, en utilisant les informations des tâches déclenchées à cet instant.

- Avantage:
    - ❖ dynamique,
    - ❖ permet l'arrivée imprévisible des tâches,
    - ❖ autorise la construction progressive de la séquence d'ordonnancement,
    - ❖ convient les tâches apériodiques.
- Inconvénient:
    - ❖ offre des solutions moins bonnes (car elle utilise moins d'informations),
    - ❖ entraîne des surcoûts de mise en œuvre.

5.3. Ordonnanceur préemptif (preemptive): une tâche élue peut perdre le processeur au profit d'une tâche plus prioritaire (ou une interruption).

- Avantage:
    - ❖ dynamique,
    - ❖ s'adapte au surcharge ou changement de l'environnement.
- Inconvénient:
    - ❖ utilisable si toutes les tâches sont préemptibles.

5.4. Ordonnanceur non préemptif (non-preemptive): une fois élue, la tâche ne doit plus être interrompue jusqu'à la fin de la requête.

- Avantage:
    - ❖ Rigide,
    - ❖ Facile à concevoir.
- Inconvénients:
    - ❖ ne s'adapte pas au surcharge ou changement de l'environnement.
    - ❖ absence de la notion interruption.

6. Critères d'ordonnancement :

- À maximiser :
  - le pourcentage d'utilisation du processeur.
  - le débit (nombre moyen de processus traités par unité de temps).
- À minimiser :
  - *le temps de réponse* : le temps entre une demande et la réponse.
  - *le temps de rotation* : durée moyenne qu'il faut pour qu'un processus s'exécute.
  - *le temps d'attente* : durée moyenne qu'un processus passe à attendre.

7. Performances des algorithmes d'ordonnancement :

- *Temps de rotation* = temps fin d'exécution - temps d'arrivée.
- *Temps d'attente* = temps de rotation - Durée d'exécution.
- *Temps moyen d'attente* = ∑ (temps d'attente) / nombre de processus.
- *Rendement* = ∑ (temps d'exécution) / nombre de processus.

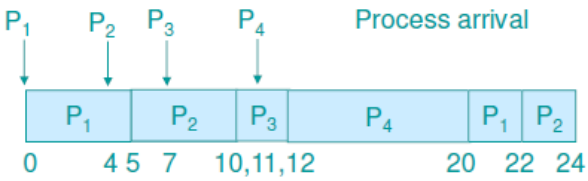
Exemple :

Utilisation de l'UCT:

- 100%

Temps de réponse (P3, P2):

- P3: 3 = (10 – 7)
- P2: 1 = (5 – 4)



Débit :

- 4/24

Temps de rotation (P3, P2):

- P3: 5 = (12 – 7)
- P2: 20 = (24 – 4)

Temps d'attente (P2):

- P2: 13 = (5 – 4) + (22 – 10)

8. Algorithmes d'ordonnancement les plus utilisés:

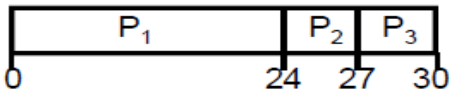
8.1. Ordonnancement FCFS (First Come First Served):

Dans cet algorithme; connu sous le nom FIFO (First In, First Out), en français (premier arrivé, premier servi); les processus sont rangés dans la file d'attente des processus prêts selon leur ordre d'arrivée.

Exemple de FCFS:

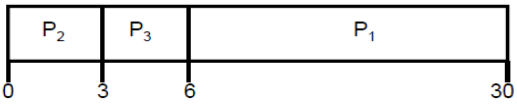
Processus	durée d'exécution
P1	24
P2	3
P3	3

Si les processus arrivent au temps 0 dans l'ordre: P1, P2, P3. Le diagramme Gantt est:



- Temps d'attente pour P1= 0, P2= 24, P3= 27
- Temps moyen d'attente: (0 + 24 + 27)/3 = 17
- Utilisation UCT = 100%
- Débit = nbre de processus / temps = 3/30 = 0,1 (3 processus complétés en 30 unités de temps)
- Temps moyen de rotation: (24+27+30)/3 = 27

Si les mêmes processus arrivent à 0 mais dans l'ordre *P2*, *P3*, *P1*. Le diagramme de Gantt est:



- Temps d'attente pour P1 = 6, P2 = 0, P3 = 3
- Temps moyen d'attente: (0 + 3 + 6)/3 = 3
- Utilisation UCT = 100%
- Débit = nbre de processus / temps = 3/30 = 0,1
- Temps moyen de rotation: (3+6+30)/3 = 13

➢ Avantages de FCFS:

❖ Simple.

➢ Inconvénients:

❖ le temps moyen d'attente peut varier grandement,

❖ les tâches de faible temps d'exécution sont pénalisées si elles sont précédées dans la file par une tâche de longue durée.

Exercice: répéter les calculs si:

◦ P2 arrive à temps 0,

◦ P1 arrive à temps 2,

◦ P3 arrive à temps 5.

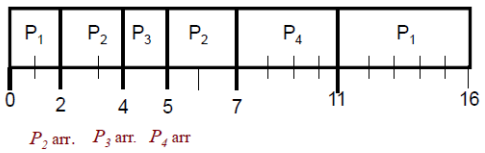
8.2. Ordonnancement SJF (Shortest Job First) (Plus Cours d'abord):

Le processus le plus court part le premier.

8.2.1. SJF avec préemption: si un processus qui dure moins que le restant du processus courant se présente plus tard, l'UCT est donnée à ce nouveau processus (**SRTF**: shortest remaining-time first).

Exemple de SJF avec préemption (SRTF):

Processus	Temps d'arrivée	durée d'exécution
P1	0	7
P2	2	4
P3	4	1
P4	5	4



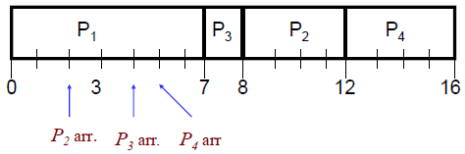
Temps moyen d`attente = (9 + 1 + 0 +2)/4 = 3

- *P1* attend de 2 à 11,
- *P2* de 4 à 5,

8.2.2. SJF sans préemption*:* on permet au processus courant de terminer son cycle.

Exemple de SJF sans préemption:

Processus	Temps d'arrivée	durée d'exécution
P1	0	7
P2	2	4
P3	4	1
P4	5	4



Temps moyen d`attente = (0 + 6 + 3 + 7)/4 = 4

- Avantages de SJF:
  - ❖ Remédie à l'inconvénient de FCFS,
  - ❖ Minimise le temps de réponse moyen.
- Inconvénients:
  - ❖ Pénalise les travaux longs,
  - ❖ Elle impose d'estimer les durées d'exécution des tâches a priori (connues difficilement).

8.3. Ordonnancement basé sur les priorités (PRIO) :

Affectation d'une priorité à chaque processus, les processus sont rangés dans la file d'attente des processus prêts par ordre de priorité.

Problème de la famine: les processus moins prioritaires n'arrivent pas à s'exécuter pendant un temps indéterminé.

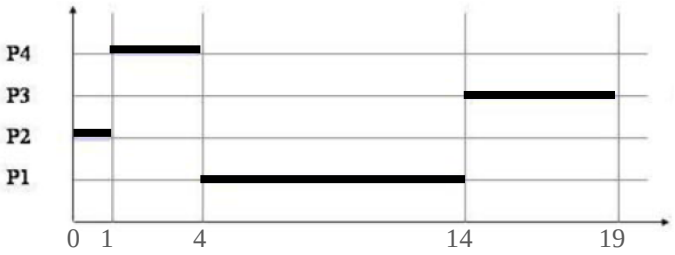
Solution : modifier/incrémenter graduellement la priorité d'un processus en fonction de son âge et de son historique d'exécution (*vieillissement*).

Pour les processus de même priorité, on peut choisir :

- FCFS,
- Tourniquet (avec la préemption).

Exemple de PRIO: supposons que les 4 processus arrivent au même temps (instant 0)

Processus	PRIO	durée d'exécution (*ms*)
P1	3	10
P2	1	1
P3	4	5
P4	2	3



Temps moyen d`attente = (4 + 0 + 14 + 1)/4 = 4,75*ms*

- Avantages de PRIO:
  - ❖ adapté aux systèmes temps réels durs (strictes),
  - ❖ répond efficacement aux interruptions et aux alertes.

- Inconvénients:
  - ❖ problème de la famine.

Remarque: l'algorithme d'ordonnancement PRIO peut avoir plusieurs variantes en fonction de la nature des priorités affectées aux tâches indépendantes périodiques. On peut citer quelques variantes:

8.3.1. Algorithme à Priorités fixes (RM : Rate Monotonic): la tâche de plus petite période est la plus prioritaire. L'utilisation de cet algorithme est valable uniquement si les tâches sont à échéance sur requête (P=D).

- Test d'acceptabilité (condition suffisante): les tâches respectent leur CT si:

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq n\left(2^{1/n} - 1\right)$$

8.3.2. Algorithme Inverse Deadline (ID) ou Deadline Monotonic (DM) : La tâche la plus prioritaire est celle de plus petit délai critique.

- Test d'acceptabilité (condition suffisante): les tâches respectent leur CT si:

$$\sum_{i=1}^{n} \frac{C_i}{D_i} \leq n\left(2^{1/n} - 1\right)$$

8.3.3. Algorithme à priorité variable ou dynamique (EDF : Earliest Deadline First) : à chaque instant (i.e. à chaque réveil de tâche), la priorité maximale est donnée à la tâche dont l'échéance est la plus proche.

- Période d'étude $= [0\,, PPCM(P_i)]$ (départ simultané des tâches)
- Test d'ordonnançabilité :

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1 \text{ (Condition nécessaire)}$$

$$\sum_{i=1}^{n} \frac{C_i}{D_i} \leq 1 \text{ (Condition suffisante)}$$

8.3.4. Algorithme à priorité variable (LLF : Least Laxity First) : la tâche à qui il reste le moins de marge s'exécute d'abord. Autrement dit, la priorité maximale est donnée à la tâche qui a la plus petite laxité résiduelle *L(t).*

- Laxité dynamique résiduelle $L(t) = D(t) - C(t) = D + r - t - C(t)$
  - retard maximum pour reprendre l'exécution d'une tâche.

Exercice:

Considérons le système temps réel suivant:

Tâche	Temps de réveil (r)	Durée d'exécution (C)	Délai critique (D)	Période (P)
T1	0	3	7	20
T2	0	2	4	5
T3	0	2	9	10

1- Calculer la période d'étude de ce système.

2- Vérifier l'acceptabilité d'ordonnancement de ce système et tracer le diagramme de Gantt sur une période d'étude en utilisant **RM** puis **DM**.

3- Vérifier l'ordonnançabilité de ce système et tracer le diagramme de Gantt correspondant en utilisant **EDF** (D_3 = 8 au lieu de 9).

4- Étudier et tracer l'ordonnancement généré par **LLF** (D_3 = 8).

Solution:

On a bien n=3

1- Période d'étude = [0, PPCM (20, 5, 10)] = [0, 20].

2- Algorithme de **RM**: ($\text{Prio}_{T2} > \text{Prio}_{T3} > \text{Prio}_{T1}$)

Remarque: ici les tâches sont à échéance sur requête ⇒ *D* = *P*.

$$\sum_{i=1}^{n} \frac{C_i}{P_i} = \frac{3}{20} + \frac{2}{5} + \frac{2}{10} = 0.15 + 0.4 + 0.2 = 0.75$$

$$n\left(2^{1/n} - 1\right) = 3\left(2^{1/3} - 1\right) = 0.779$$

Donc, les 3 tâches respectent leur CT car la condition suffisante est vérifiée $0.75 \leq 0.779$.

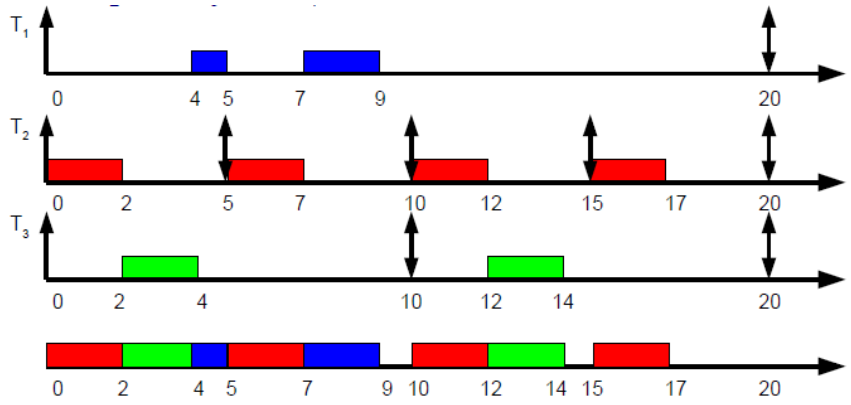


Diagramme de Gantt généré par RM

Algorithme de **DM**: ($\text{Prio}_{T2} > \text{Prio}_{T1} > \text{Prio}_{T3}$)

$$\sum_{i=1}^{n} \frac{C_i}{D_i} = \frac{3}{7} + \frac{2}{4} + \frac{2}{9} = 1.14 \geq 0.779$$

Donc, la condition suffisante n'est pas vérifiée. Cependant, le diagramme construit sur la période d'étude montre qu'elles sont ordonnançables sans faute temporelle.

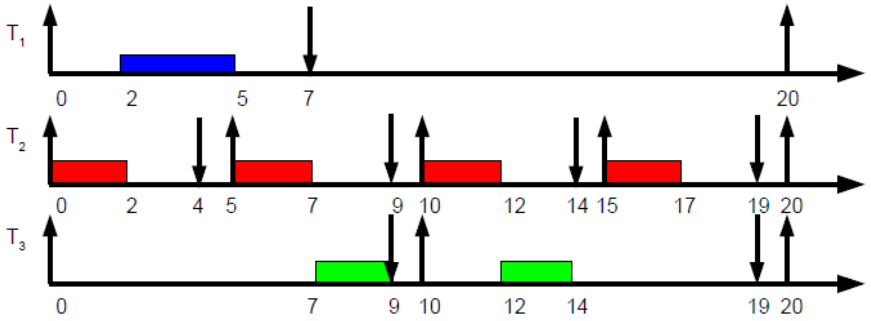


Diagramme de Gantt généré par DM

3- Algorithme **EDF**:

La condition nécessaire est vérifiée mais la condition suffisante n'est pas vérifiée.

- À t=0 : au réveil des 3 tâches, T2 prioritaire, elle s'exécute pendant 2 unités.
- À t=2, T2 termine. C'est T1 la plus prioritaire. Elle s'exécute pendant 3 unités de temps.
- À t=5, T1 se termine et T2 se réveille de nouveau (car sa période 5), mais c'est T3 qui est cette fois la plus prioritaire car son échéance est 8, elle s'exécute pendant 2 unités.

- Ici, on voit que les priorités des tâches varient dans le temps : par exemple, à t=0, c'est T2 la plus prioritaire (que T1 et T3), mais à t=5, c'est T3 la plus prioritaire.

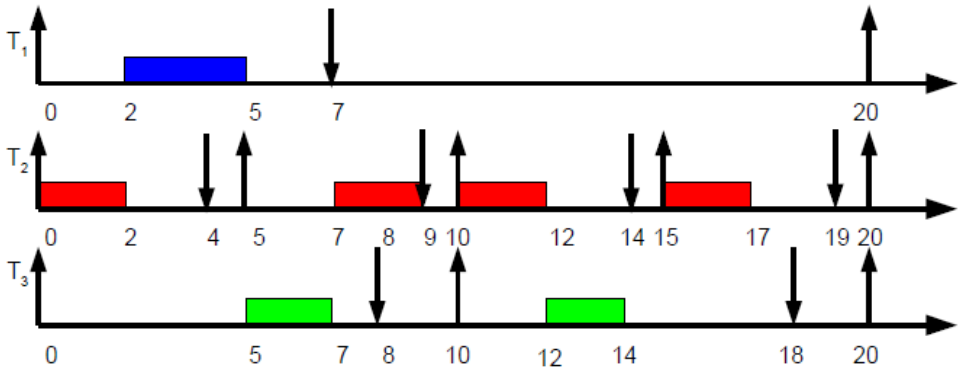


Diagramme de Gantt généré par EDF

4- Algorithme **LLF:**

- À t=0, les 3 tâches sont réveillées:

  Laxité de T1 = 7 - 3 = 4

  Laxité de T2 = 4 - 2 = 2

  Laxité de T3 = 8 - 2 = 6

  C'est donc T2 (la plus prioritaire) qui s'exécute.
- À t=1, L1 = 7-1-3= 3, L2= 4-1-1= 2, L3 = 8-1-2= 5 ⇒ T2 qui s'exécute.
- À t=2, T1 qui s'exécute (sa laxité [7-2-3] est plus petite que celle de T3 [8-2-2]).
- À t=3, L1 = 2, L3 = 3 ⇒ T1 qui s'exécute.
- À t=4, L1 = 2, L3 = 2 ⇒ T1 qui s'exécute (T1 est préférable, le retard de T3 est toléré tant quand son échéance est toujours respectée).
- À t=5, T2 se réveille de nouveau.

  L2 = 9-5-2 = 2 (2ème échéance - temps courant - durée exécution),

  L3 = 8-5-2 = 1. Donc T3 qui s'exécuter.
- À t=6, L2 = 1, L3 = 1 ⇒ T1 qui s'exécute.
- À t=7, L2 = 9-7-2 = 0 ⇒ T2 qui s'exécute.

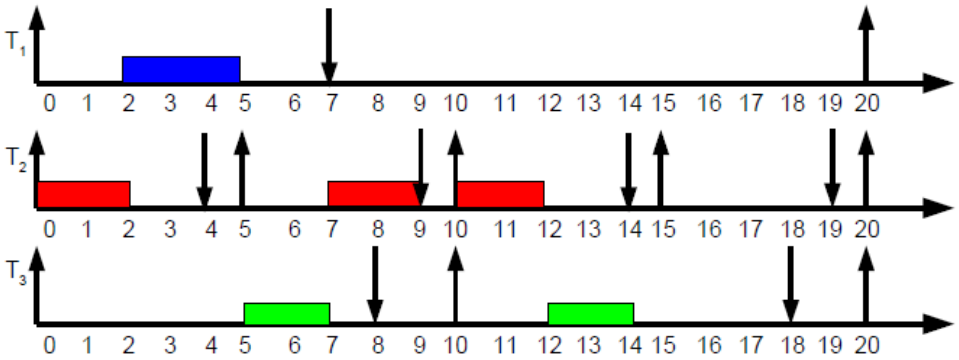


Diagramme de Gantt généré par LLF

8.4. Ordonnancement par Tourniquet/Tour de rôle (RR : Round Robin) :

Le processeur est alloué successivement aux différents processus pour une tranche de temps fixe ***Q*** appelé ***Quantum*** (tranche du temps).

L'efficacité de cet ordonnanceur dépend principalement de la valeur du quantum ***Q***:

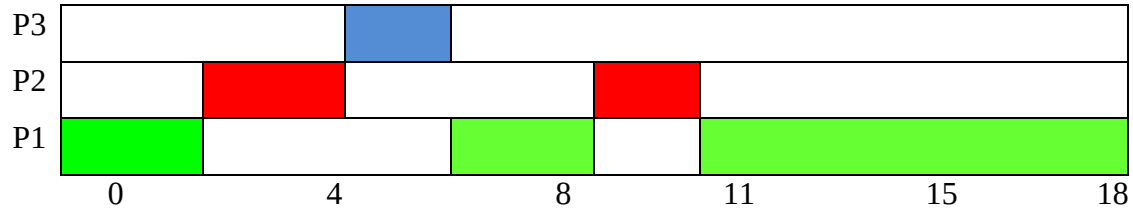
- ***Q*** assez petit augmente le nombre de commutation.
- ***Q*** assez grand augmente le temps de réponse du système → FCFS.

- Avantages de RR:
  - Adapté aux systèmes temps partagé,
  - garantit que tous processus sont servis au bout d'un temps fini,
  - méthode préemptive,
  - évite *la famine*.
- Inconvénients:
  - ***Q*** optimal est difficilement estimé.

Exemple de RR:

Processus	PRIO	Durée d'exécution (*ms*)
P1	1	20
P2	2	7
P3	3	3

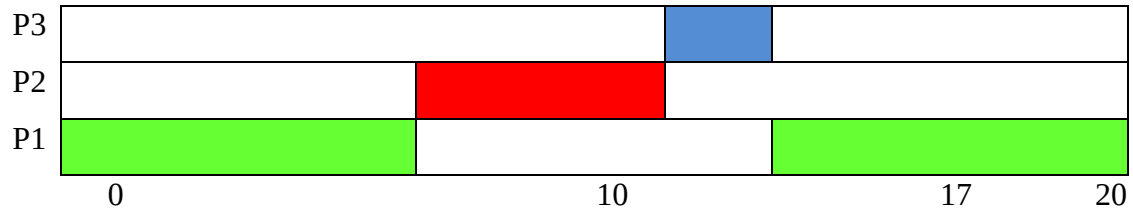
A- avec *Q*uantum = 4



30

- Temps moyen d'attente: (8 + 11 + 8)/3 = 9
- Temps moyen de rotation: (30+18+11)/3 = 19,66

A- avec *Q*uantum = 10



30

- Temps moyen d'attente: (10 + 10 + 17)/3 = 12,33
- Temps moyen de rotation: (30+17+20)/3 = 22,33