Projet round robin

Maël Bourdon[1] and Mathias Cabezas[1]

[1]Université de Technologie de Belfort Montbéliard

November 29, 2019

Modèle général

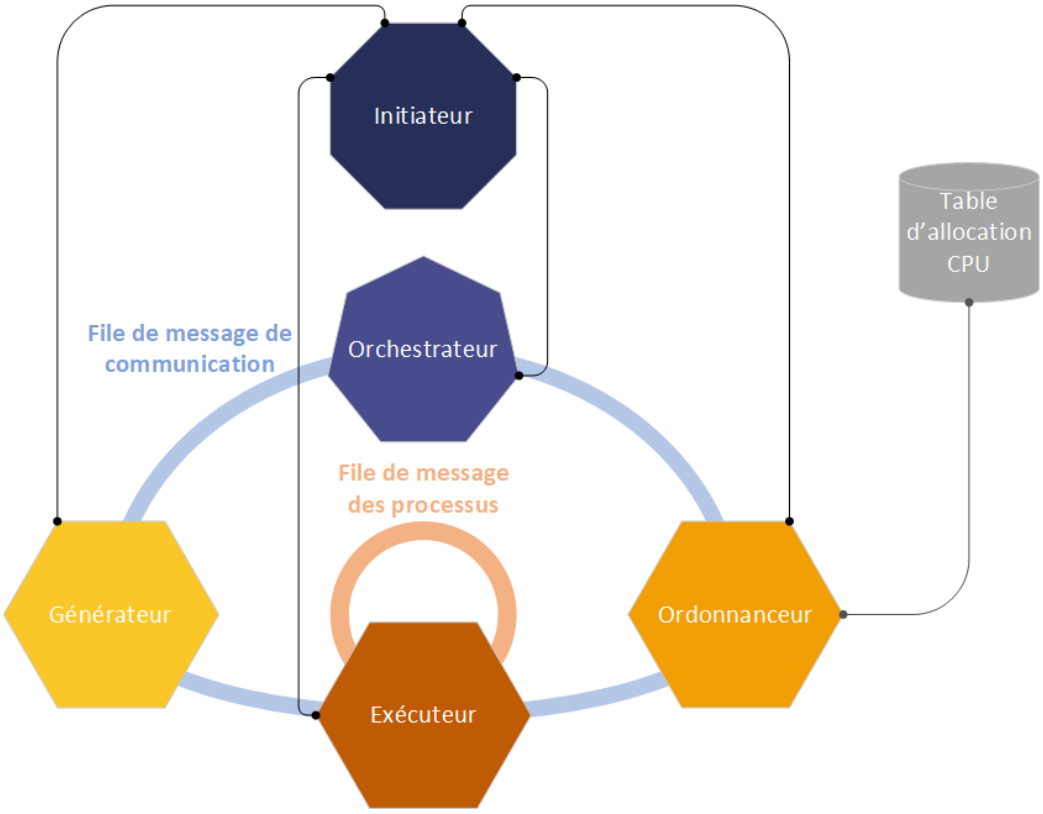

Figure 1: modèle général

Notre conceptualisation du projet utilise 5 processus :

- Le processus “Initiateur” :

Ce processus est en charge du lancement du projet. Afin de répondre à ce besoin le processus créer 4 threads qui sont “Orchestrateur”, “Générateur”, “Exécuteur” et “Ordonnanceur” (les 4 processus cardinaux). La philosophie derrière cette répartition est d’avoir un processus maître qui va fixer les bases de l’architecture des cardinaux et ainsi nous permet d’avoir un projet modulaire. On pourrait y fixer des variables “globales” ou rajouter des nouveaux processus.

- Le processus “Générateur” :

Ce cardinal à pour rôle de créer les processus du round robin. Pour ce faire il possèdent une méthode de création de structures qui représentent nos processus. L’ensembles des processus sont créés au lancement de l’application est sont stockés dans une table. Générateur est alors en attente d’une demande de Orchestrateur et lui fournit alors sur demande le processus réquisitionné.

- Le processus “Exécuteur” :

Ce cardinal endosse le rôle de traitement des données liés aux processus. Il récupère les processus qui lui sont donnés par Orchestrateur. Une fois qu’un processus est récupéré, il est stocké dans la file de message des processus (nous évoquerons les spécificités des files de messages par la suite).

- Le processus “Ordonnanceur” :

Ce cardinal est le régent de l’ordre d’exécution. En effet il est mandaté par l’orchestrateur pour la recherche de la file à un quantum donné. Pour répondre à cette fonction il attend une requête de l’orchestrateur contenant le quantum puis recherche la file à exécuter dans la table d’allocation CPU et renvoi cette file en réponse à l’Orchestrateur.

- Le processus “Orchestrateur” :

C’est le cardinal principal. En effet il endosse la synchronisation du système. Il est donc l’intermédiaire de tous les cardinaux de “bas niveau” (Générateur, Exécuteur et Ordonnanceur).

Modèle de dialogue

Pour créer notre modèle nos esprits se sont implicitement tournés vers du dialogue client serveur puisque en effet les processus cardinaux de "bas niveau" (Générateur, Exécuteur et Ordonnanceur) sont majoritairement en écoute d'une requête venant de notre client l'Orchestrateur d'où le parallèle avec le modèle client serveur.

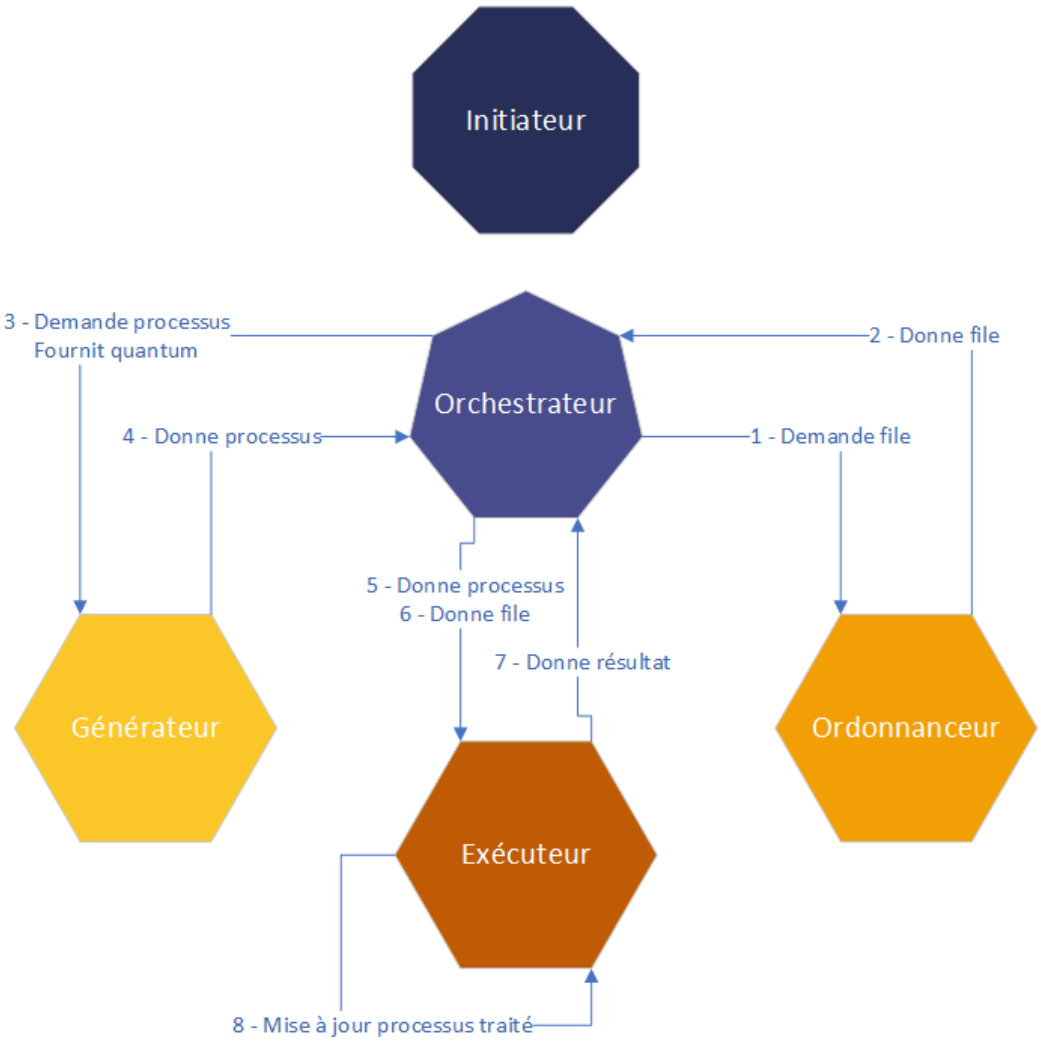

Figure 2: modèle de dialogue

Modèle de structure

Afin que nos processus puisse dialoguer entre eux par l'intermédiaire des files de messages il nous faut définir des structures représentants alors nos messages :

- Processus :

```
typedef struct {
long type;
long prio;
int pidProcess;
int tempsExec;
int tempsArrivee;
}process;
```

- Message à donnée entière

```
typedef struct {
long type;
int msg;
}tMessage;
```

Modèle de table d'allocation CPU

Concernant la table d'allocation CPU il était complexe de trouver une bonne répartition des processus qui soit "correcte" puisque par définition celle ci est arbitraire.

La premier étape de raisonnement consistait à fixer nos "bases" :

- Nombres de files : 10 (0 - 9)
- Plus une file d'attente est de numéro faible plus elle est prioritaire
- On doit consacrer plus de quantum à une file d'attente de numéro faible
- On se limite à 100 quantum maximum

On as donc tracé un gabarit de courbe simple :

$f(n) = -n + 11$

$f(n)$: nombre de quantum dédié à la file d'attente

n : numéro de file

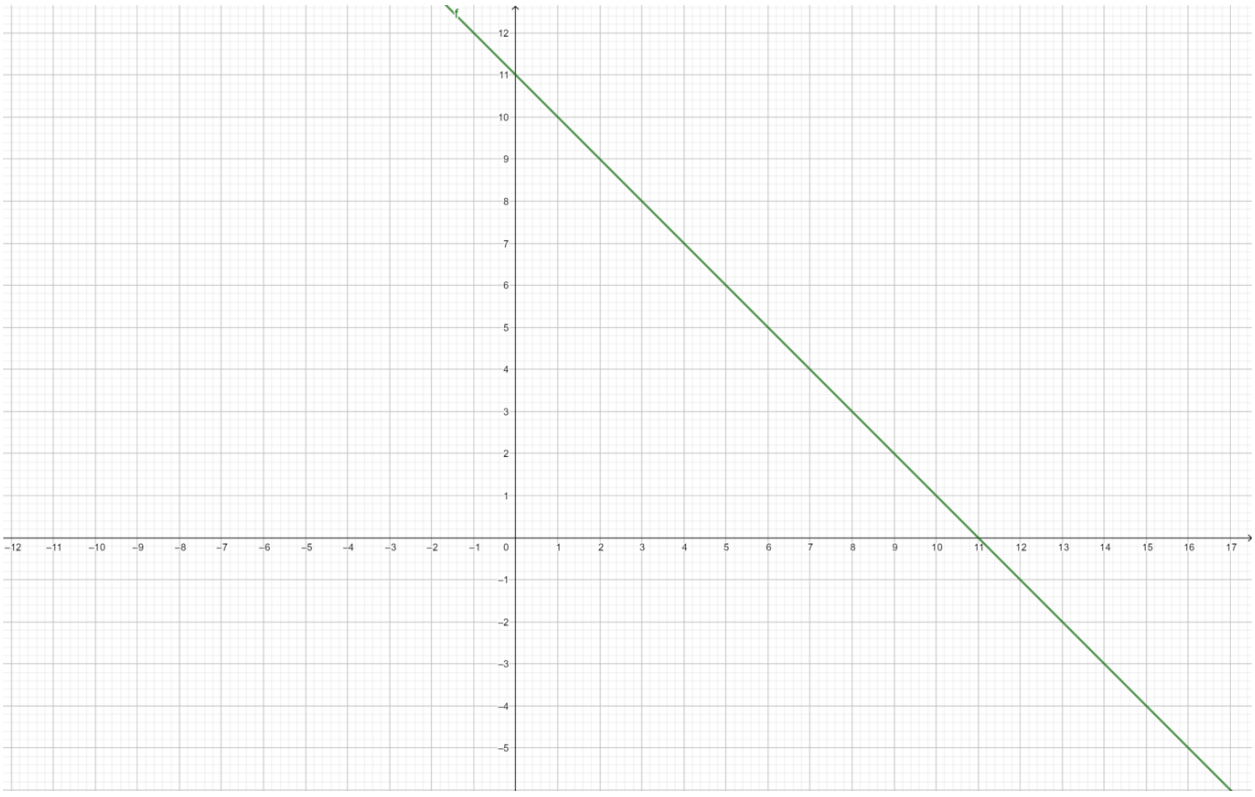

Figure 3: Courbe d'allocation CPU

Hosted file table$_a$*llocation*$_c$*pu.xlsxavailableathttps* : *//authorea.com/users/*285959*/articles/*409883*−projet−round − robin*

Modèle des files de messages

Nous avons utilisés deux files de messages dans notre projet. Une d'entre elle est utilisée pour permettre à nos processus cardinaux de communiquer tandis que l'autre est utilisée pour stocker l'ensemble de nos processus simulés.

La file de message de communication :

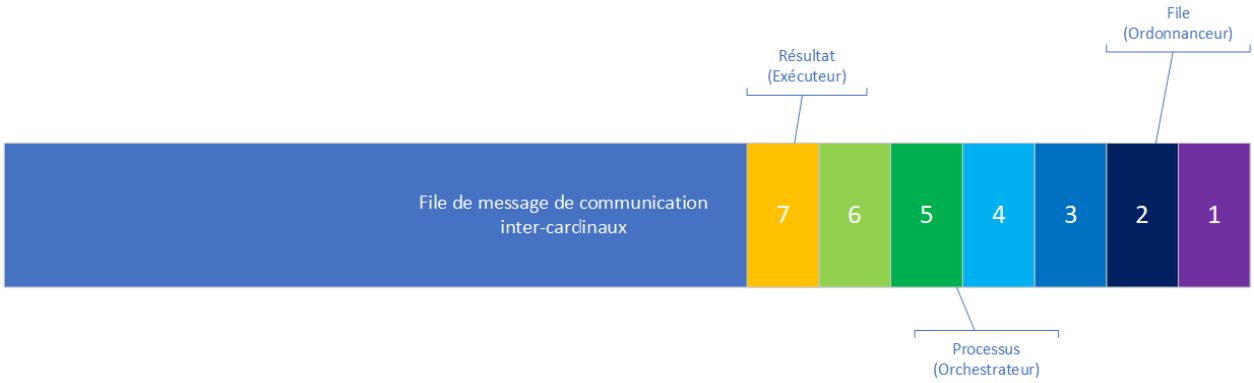

Figure 4: File de communication inter-cardinaux

Nous avons fait le choix d'utiliser un type de message par échange possible (référence au modèle de dialogue). En effet dans le cas d'une requête-réponse dont les messages serait de même type un processus cardinal aurait pu être capable de se répondre à lui même.

La file de message de processus :

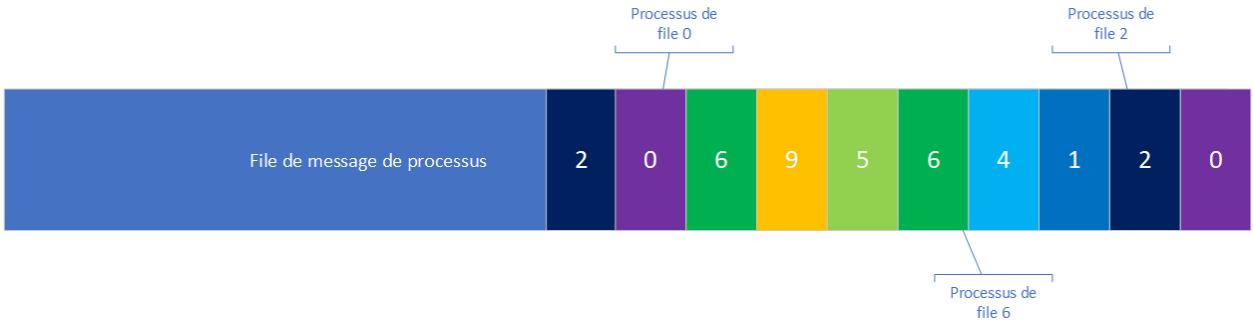

Figure 5: File de message de processus

Concernant la file de message de processus chaque type de message correspond à la priorité (donc à la file d'attente) d'un processus.

Teste

Pour vérifier que notre application produit des résultats en accords avec la théorie du round-robin et notre table d'allocation CPU, on récupère les conditions initiales que notre algorithme à utilisé pour arriver à ce résultat et on met en pratique la théorie du round-robin sur ces conditions initiales.

Voici les conditions initiale utilisées par notre algorithme :

```
PID : 887 | Priorité : 4 | Temps Exécution : 8 | Temps Arrivée : 0
PID : 336 | Priorité : 4 | Temps Exécution : 7 | Temps Arrivée : 2
```

Voici le résultat de notre algorithme :

```
****************************
Quantum Courant = 0
----------------------
Processus = 887
Arrivee = 0
Priorité = 5
Temps Execution Restant = 7
****************************

****************************
Quantum Courant = 1
----------------------
Processus = 887
Arrivee = 0
Priorité = 6
Temps Execution Restant = 6
****************************

****************************
Quantum Courant = 2
----------------------
Processus = 336
Arrivee = 2
Priorité = 5
Temps Execution Restant = 6
****************************

****************************
Quantum Courant = 3
----------------------
Processus = 336
Arrivee = 2
Priorité = 6
Temps Execution Restant = 5
****************************

****************************
Quantum Courant = 4
----------------------
Processus = 887
```

```
Arrivee = 0
Priorité = 7
Temps Execution Restant = 5
****************************

****************************
Quantum Courant = 5
----------------------
Processus = 336
Arrivee = 2
Priorité = 7
Temps Execution Restant = 4
****************************

****************************
Quantum Courant = 6
----------------------
Processus = 887
Arrivee = 0
Priorité = 8
Temps Execution Restant = 4
****************************

****************************
Quantum Courant = 7
----------------------
Processus = 336
Arrivee = 2
Priorité = 8
Temps Execution Restant = 3
****************************

****************************
Quantum Courant = 8
----------------------
Processus = 887
Arrivee = 0
Priorité = 9
Temps Execution Restant = 3
****************************

****************************
Quantum Courant = 9
----------------------
Processus = 336
Arrivee = 2
Priorité = 9
Temps Execution Restant = 2
****************************

****************************
```

```
Quantum Courant = 10
----------------------
Processus = 887
Arrivee = 0
Priorité = 10
Temps Execution Restant = 2
*****************************

*****************************
Quantum Courant = 11
----------------------
Processus = 336
Arrivee = 2
Priorité = 10
Temps Execution Restant = 1
*****************************

*****************************
Quantum Courant = 12
----------------------
Processus = 887
Arrivee = 0
Priorité = 10
Temps Execution Restant = 1
*****************************

*****************************
Quantum Courant = 13
----------------------
Processus = 336
Arrivee = 2
Priorité = 10
Temps Execution Restant = 0
*****************************

*****************************
Quantum Courant = 14
----------------------
Processus = 887
Arrivee = 0
Priorité = 10
Temps Execution Restant = 0
*****************************

*****************************
Quantum Courant = 15
----------------------
RIEN A EXECUTER POUR CE QUANTUM
*****************************

Fin de l'execution
```

Voici le résultat de notre mise en pratique de la théorie du round-robin en partant de ces mêmes conditions initiales :

	File d'attente															
Quantum		0	1	2	3	4	5	6	7	8	9	10	11	12	13	1
	1															
	2															
	3															
	4															
	5	887(7)		336(6)												
	6		887(6)		336(5)											
	7			887(6)	887(6)	336(5);887(5)	336(4)									
	8						887(5)	336(4);887(4)	336(3)							
	9								887(4)	336(3);887(3)	336(2)					
	10										887(3)	336(2);887(2)	887(2);336(1)	336(1);887(1)	887(1);336(0)	887(0)

PID	Priority	Exec_time	Arrival_time
887	4	8	0
336	4	7	2

Figure 6: Résultat de la théorie

Conclusion

Pour conclure ce projet, l'objectif est atteint cependant certaines fonctionnalités sont manquantes, notamment la possibilité de changer le quantum ou de pouvoir modifier les processus par action humaine. Le projet manque aussi d'une fin d'exécution.