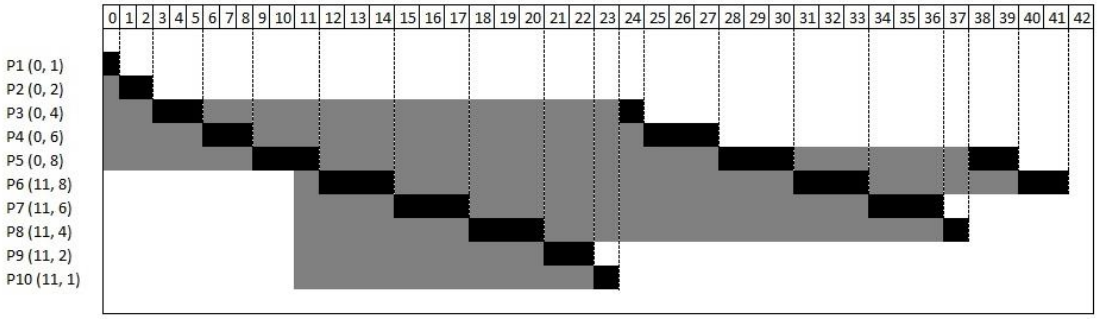
Projet LO41 : Algorithme d'ordonnancement



Corentin MENIER – Aimeric MONTANGE

LO41 – A19 – Département informatique

Date du rendu : 06/01/2020

Établissement :

Université de Technologie de Belfort-Montbéliard

Responsable UV :

Philippe DESCAMPS

Tables des matières

1 – Présentation du sujet..... 3
2 – Organisation du code..... 3
2.2 – Première partie : Allocation.....3
2.3 – Deuxieme Partie : RoundRobin.....4
2.4 – Troisième Partie : Libération de la mémoire 5
3 – Jeu de Test 6
4 – Conclusion..... 8

1 – Présentation du sujet

Le but de ce projet est de concevoir un algorithme d'ordonnancement simple s'appuyant sur le modèle du RoundRobin avec priorités. Nous devions également respecter les principes de synchronisations et de communication vus en cours.

Notre programme est en langage C, et conçu sous Linux. Dans tout le projet la priorité maximale et forte est 10.

2 – Organisation du code

Afin de réaliser ce projet nous avons segmenter notre afin de faciliter son utilisation et sa compréhension. Nous avons créé un unique fichier appelé ProjetLO41, comprenant le Main et les fonctions auxiliaires permettant au programme de fonctionner. Comme nous avons un seul ficher de code nous n'avons pas eu la nécessité de créer un Makefile afin de compiler le programme.

Notre programme se décompose en trois parties :

- La partie d'allocation de mémoire
- La partie de l'algorithme d'ordonnancement
- La partie de restitution et libération de mémoire

2.2 – Première partie : Allocation

Nous allons expliquer certaines fonctions dans le déroulement normal du programme ensuivant la trame du Main.

Tout d'abord lors de l'exécution du programme l'utilisateur devra renseigner 3 arguments qui correspondent respectivement au nombre de processus voulu, a la date de soumission maximale et au temps d'exécution maximal. Ses trois arguments sont vérifiés et l'utilisateur est alerté si ceux-ci ne sont pas conformes ou absents.

Nous créons un tableau dynamiquement rempli de pid des processus fils créer.

Nous initialisons des sémaphores correspondant à chaque niveau de priorité (10), grâce a la fonction initsem. Elle permet de les initialiser à zéro chacun.

Par la suite nous attribuons un sémaphore à chaque processus fils afin de gérer leur exécution et contrôler leur activation.

Nous devions aussi créer un quantum de temps afin de décrémenter ou d'incrémenter les processus avec une échelle de temps qui dans notre cas sera d'une unité de temps par action. Ce quantum de temps est un segment de mémoire partagé qui sera incrémenter pendant l'exécution et l'avancement du programme.

Chaque processus a quatre attribues, un identifiant, une priorité une date de soumission et un temps d'exécution. Afin de remplir la table d'allocation, il faudra créer une matrice du nombre de processus par 4 (attribues). La fonction remplir_desc permet de créer cette matrice dynamiquement et de la remplir aléatoire avec les arguments que l'utilisateur aura rempli lors de l'exécution du programme.

La fonction CreerTourniquet, elle, permet de créer dix tourniquets correspondants aux dix niveaux de priorité des processus de l'algorithme d'ordonnancement.

Un segment de mémoire partagée est créé en plus pour stocker les priorités qui seront choisit à chaque quantum de temps afin de travailler ensuite avec le bon tourniquet et le processus à exécuter pendant ce quantum. La taille de segment est égale à la date de soumission maximale plus le nombre de processus fois le temps d'exécution maximale de chacun d'eux.

2.3 – Deuxieme Partie : RoundRobin

Cette partie correspond au plus gros du programme, c'est-à-dire que c'est la principale fonction qui permet l'exécution des processus en fonction de leur priorité.

Nous allons analyser la fonction fonc_roundRobin. Celle-ci vérifie tout d'abord si dans la table des descripteurs il y a un temps d'exécution supérieur à zéro ou pas

pour savoir s'il y a des processus à exécuter ou si tous les processus sont déjà morts. C'est la fonction check_process qui permet de faire ceci.

Il faut ensuite vérifier s'il y a bien un processus disponible a la date de soumission et priorité voulu afin de ne rien faire pendant un quantum de temps (check_start_process).

Une fois le processus vérifié et identifié, il est exécuté pendant un quantum de temps. Son temps d'exécution total sera décrémenté de même pour sa priorité.

Une gestion de file s'opère ensuite c'est le principe de la queue FIFO que nous avons appliqué ici, une fois un processus exécuter, il sera mis à la fin de la file de la priorité inferieur.

La fonction sortir_int_trn permet cette gestion de file de type FIFO, elle s'applique sur les tourniquets.

Nous avons aussi implémenté une fonction permettant d'arrêter le processus du programme lui-même avec l'action CTRL+C, nous avons utilisé les signaux pour ceci. C'est la fonction traitantSIGINT, elle teste d'abord si on se trouve dans un processus fils ou un processus grparent, si nous sommes dans un processus fils un signal SIGINT est émis au parent (fils du grand parent) grâce au pid stocké précédemment. Un fois dans le processus parent, il tue ses processus fils, et se terminera lui-même permettant au processus grparent mis en attente sur la terminaison de son propre processus fils, de s'occuper lui-même de libérer la mémoire. Et ainsi arrêter le programme.

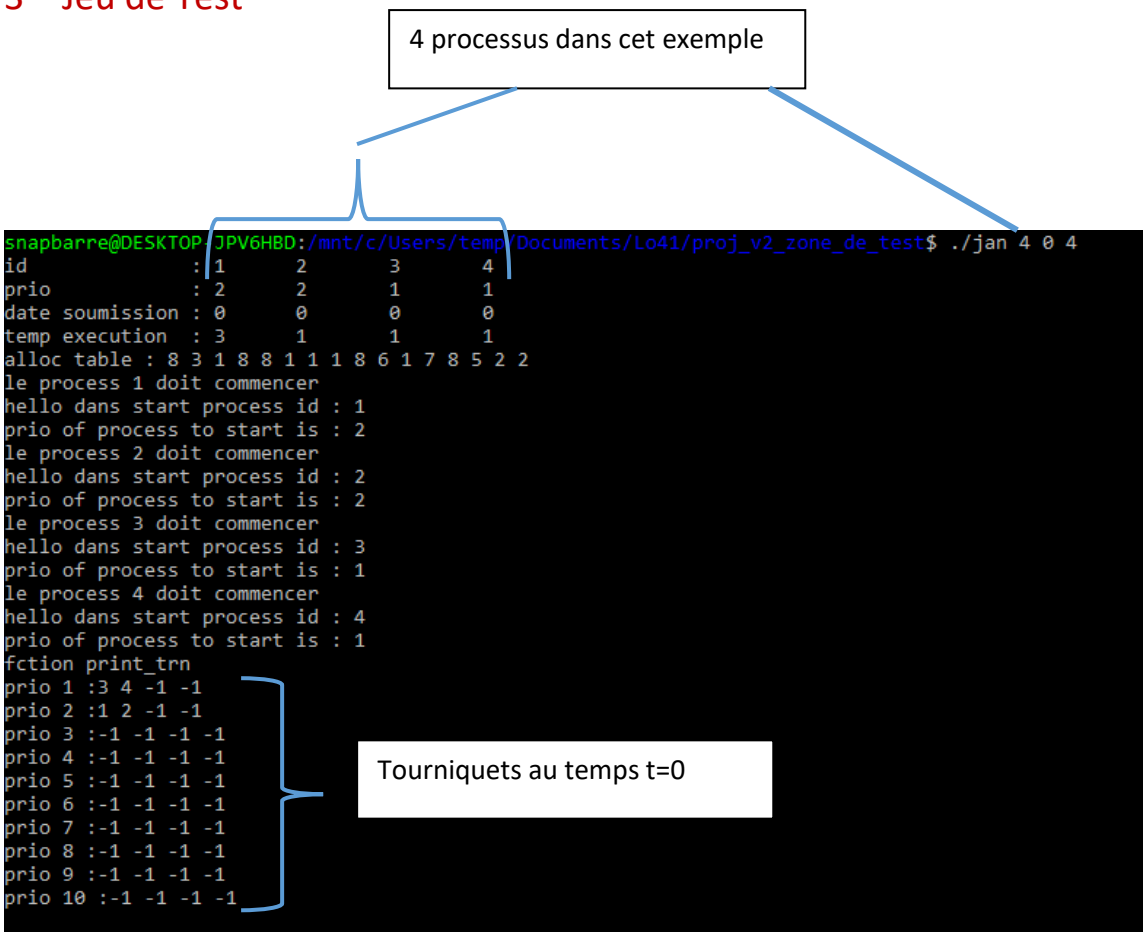
2.4 – Troisième Partie : Libération de la mémoire

De nombreuses fonctions sont implémenter afin de libérer toute la mémoire que nous avons allouer lors de l'exécution du programme.

Nous avons libéré la mémoire des différents segments : tourniquets, quantum de temps, descripteur, allocation, grâce à la fonction liberationSeg.

Une autre libération est effectuée celle des sémaphores avec la fonction liberationSem.

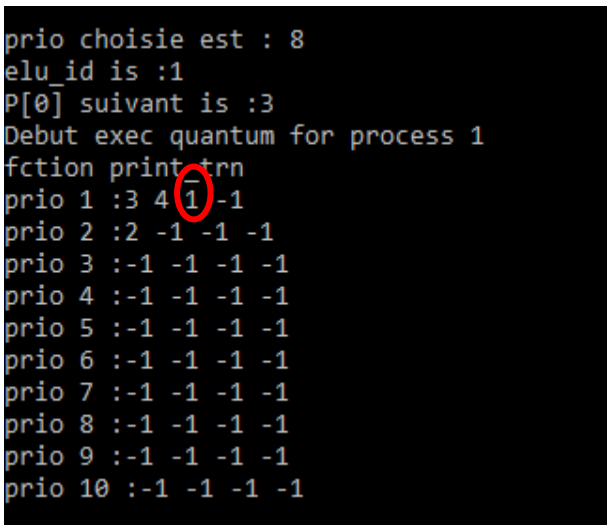
3 – Jeu de Test

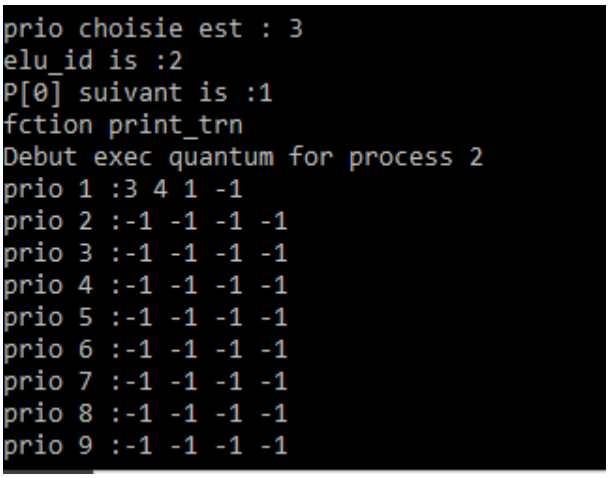


Au temps t=1

Le processus 1 est elue et est executé.

Il est ensuite placé dans le tourniquet de priorité inferieur ici priorité 1.





Ici au temps t=2 c'est la priorité 2 qui a été choisit et c'est le processus 2 qui a été élu. Il n'avait qu'un temps d'execution de 1 donc il se termine et disparait de l'algorithme.

4 – Conclusion

Ce projet nous a permis de bien mettre en application les notions vus en cours, TD et TP et de les réunir dans un même exemple. Il nous a permis de mieux appréhender ses notions et d'aller au-delà en faisant nos recherches pour savoir comment implémenter celle-ci dans certains cas précis. Les notions telles que les signaux, les segments de mémoires partagées, les sémaphores, ont été les principales notions utilisées. Ce projet nous a aussi permis de nous perfectionner dans la programmation C système.