TP PRÉPARÉ PAR :

BENLOULOU Sarra NEKKAA Yousra

Université Clermont Auvergne

Institut d'Informatique

Département de l'informatique

Système d'exploitation: Processus

Encadré par :

TAYSSIER DAMIEN

Année Universitaire : 2018-2019

Module processus :

NB : vous pouvez voir le dossier processus

- **Les variables globale :**
  - Liste_processus , une liste chainée de processus :
    - Cette liste rassemble tous les processus présents dans le système
  -
- **Structure Processus :**
  - Notre structure processus contient :
  - Le numéro du processus
  - Le numéro du processus père
  - Le nombre du processus fils qu'il possède (il est à 0 au départ)
  - Un tableau de numéros des processus fils (nous avons définit un nombre maximum de processus fils pour chaque processus)
  - Un entier pt_pile
  - Un entier qui indiquera l'état du processus
  -

INDICE	ETAT
0	En execution
1	En attente
2	Prêt à s'exécuter
3	Zombie

  - Le nombre d'instructions
  - Le programme (tableau dynamique d'instructions)
  - Un entier qui désigne l'état du registre (sera utilisé dans le cas d'un appel de Bloquante_ressource , 1 : ressource prête et 0 ressource pas encore prête)
  - Un pointeur vers le processus suivant

- **Structure Instruction :**
  - Drapeau valide (prendra la valeur 0 ou 1. valide/pas valide)
  - Drapeau Bloquante (prendra la valeur 0 ou 1)
  - Drapeau Bloquante_processus (prendra la valeur 0 ou 1)
  - Drapeau Bloquante_ressource (prendra la valeur 0 ou 1)

- Drapeau Bloquante_processus cible (si l'instruction est bloquante_processus ou Bloquante_ressource nous lui affectons un processus cible)

- **Structure Interruption :**
  - Numero du processus origine (celui qui à déclenché l'interruption)
  - Numéro du processus cible (sera utilisé dans le cas des interruptions ATTENTE_FIN_PROCESSUS ou ATTENTE_UNE_RESSOURCE)
  - Entier informant le type d'interruption

Type d'Interruption	Indice	Traitement
FIN_PROCESSUS	0	Supprimer le processus origine
FIN_APPEL_BLOQUNT	1	Reveiller le processus origine
ATTENTE_FIN_PROCESSUS	2	Réveiller le processus cible
ATTENTE_UNE_RESSOURCE	3	Réveiller le processus cible et changer l'état de registre de processus origine, et le placer dans la liste des processus prêt à s'exécuter

- **Structure Machine :**
  - Un compteur ordinale initialisé à 0 au départ
  - Un registre (utilisé dans le cas des instructions bloquante ressource)
  - Un processus à traiter

- **Les fonctions utilisées :**

NB : Vous pouvez découvrir toutes les fonctions citées ci dessous dans le module processus.c

➢ **Creation de processus** :

(*int creation_processus(int num_pere)*)

Cette fonction prend un numéro du processus père (dans la simulation, nous faisons bien attention de donner un numéro du processus père qui existe déjà).

Nous donnons un numéro disponible à notre processus, nous avons un tableau (*tab_processus*) qui enregistre l'état des processus, nous prenons de ce tableau le premier numéro du processus disponible

Num proc	0	1	2	3	4

Etat	-2	-1	-1	-1	-1

Dans cette exemple c'est le numéro 1 que nous devons prendre, nous ne prenons jamais le processus 0 (init), le processus init a un état spéciale

Num proc	0	1	2	3	4
Etat	-2	2	-1	-1	-1

Nous changeons l'état de ce processus dans le tableau. Si le nombre max de processus est atteint alors nous arrêtons la fonction création.

Nous donnons un nombre d'instruction aléatoirement (nous avons défini le nombre maximum d'instructions que le processus peut avoir), nous remplissons chaque instruction aléatoirement, valide, bloquante, Bloquante_processus , Bloquante_ressource. Si l'instruction est Bloquante_processus ou Bloquante_ressource on doit remplir le champs du processus cible.

affecte_fils(int num_pere, int num_fils), cette fonction affecte au processus père le processus fils que nous venons de créer

Puis nous l'ajoutons à la fin de la liste des processus grâce à la fonction *ajoute_en_fin()*

➢ **Gestion des Instructions :**

'int gestion_instruction(struct liste_processus *listeProc, struct processus *processus_en_cours)'

Cette fonction prend un processus et exécute son instruction. Elle retourne 2 si elle se termine par une interruption, 1 si l'instruction traitée ne génère pas d'interruption , -1 si elle termine par une erreur.

Différents cas sont gérés dans cette fonctions :

- Si l'instruction courante dépasse le nombre d'instruction de ce processus, c'est à dire que ce processus a fini le traitement de toutes ses instructions,nous changeons son état à EN_ATTENTE et nous générons une interruption de FIN_PROCESSUS grâce à la fonction *genere_interruption_2*
- Si l'instruction n'est pas valide, nous changeons son état à EN_ATTENTE et nous générons une interruption de FIN_PROCESSUS.

- Si l'instruction est valide et n'est pas bloquante "ni processus ni ressource" , nous affichons un message disant que l'instruction est valide, nous incrémentons le pt_pille et on retourne 1.
- Si l'instruction est Bloquante_processus nous regardons si le processus cible est valide (le processus cible doit être différent de -1, 0, ou processus courant, celui qui a appelé la fonction)
    - Si le processus cible n'est pas valide, nous déclenchons une interruption de FIN_PROCESSUS. (on le considère comme instruction pas valide)
    - Sinon, on incrémente le pt_pile, on change son état à EN_ATTENTE, on déclenche une interruption de ATTENTE_FIN_PROCESSUS et on retourne 2
- Si l'instruction est Bloquante_ressource, on change l'état à EN_ATTENTE, on déclenche une interruption de type ATTENTE_UNE_RESSOURCE

➢ **Gestion des Interruption :**

Cette fonction prend en paramètre le numéro du processus en cours et prend la premiers Interruption dans la file des interruption (FIFO) , si cette dernière concerne le processus en cours alors elle affiche un message pour informer que l'interruption le concerne, sinon un message pour dire qu'elle ne le concerne pas. A la fin de chaque traitement d'interruption on supprime celle qui a été traitée de la file (pour éviter son exécution à chaque fois) , quand la fonction de gestion d'interruption se termine par une génération d'une nouvelle interruption nous retournons 2, nous retournons 1 dans le cas contraire et -1 dans le cas d'erreur.

A chaque traitement d'interruption nous effectuons le traitement suivant:

- Si l'interruption est égale à NULL c'est à dire qu'il n'y a pas d'interruption à traiter nous retournons 1
- Si le type d'interruption est FIN_PROCESSUS, nous supprimons ce processus de la liste à laquelle il appartient et nous adoptons ses fils grâce à la fonction *supprimer_proc_adopte_fils ,et* nous retournons 1
- Si le type d'interruption est FIN_APPEL_BLOQUNT, c'est à dire que nous avons besoin de réveiller ce processus , nous changeons son état à prêt à s'exécuter, et nous incrémentons le pt_pile et nous returnons 1
- Si l'interruption est de type ATTENTE_FIN_PROCESSUS : nous déclenchons cette interruption quand nous avons besoin d'une ressource qui vient d'un processus qui est en état attente (le processus cible),
    - Si le processus cible est égale à 0, -1 ou il a le même numéro que le processus d'origine (celui qui a déclenché l'interruption) alors nous considérons cela comme une interruption non valide, et nous

déclenchons une interruption de fin du processus d'origine, nous retournons 2 (pour dire qu'une interruption a généré une autre interruption)
    - Sinon, nous générons une interruption de FIN_APPEL_BLOQUNT, pour réveiller le processus cible nous plaçons le processus d'origine dans la liste des prêt à s'exécuter , nous incrémentant le pt_pile et nous retournons 2
- Si l'interruption est de type ATTENTE_UNE_RESSOURCE, nous générons aléatoirement un nombre (0 ou 1) qui désigne l'état du registre :
    - Si le nombre aléatoire est égale à 1, c'est à dire que la ressource est prête, nous copions donc le contenu du registre de la machine dans le registre du processus , nous changeons l'état du processus à prêt à s'exécuter et nous le plaçons à la fin de la liste des processus, et nous retournons 1.
    - Si le nombre aléatoire est égale à 0, c'est a dire sa ressource n'est pas encore prête, nous générons une nouvelle interruption qui concerne le même processus d'origine, et nous retournons 2 (le processus gardera son état 'en attente')

➢ **Ordonnancement :**

- **Ordonnancement FIFO**

Pour cette fonction nous définissons un nombre de cycle maximale que l'ordonnanceur peut faire, tant que le compteur ordinale de la machine n'a pas atteint ce nombre alors nous faisons le traitement suivant (nous pouvons changer la boucle "tantque" à tant que nous avons traité tous les processus et qu'il y n'a aucune interruption dans la file des interruptions) :

- Nous donnons à la machine le premier processus prêt à s'exécuter
    - Si nous avons pas de processus prêt à s'exécuter et que nous avons des interruptions à traiter alors nous les traitons toutes
    - Si nous avons traité toutes les interruptions et qu'il n'y a pas de processus à traiter alors nous affichons un message disant que nous avons fini l'exécution de tous les processus
    - Sinon, nous changeons l'état du processus à l'etat en exécution (indice 0), nous donnons à ce dernier un temps CPU aléatoire (nous avons définit un temps cpu maximum)
    - Tant que le temps CPU du processus n'est pas terminé :
        - Nous exécutons la première interruption de la file (s'il y'en a une) à l'aide de la fonction *gestion_interruption*

        - Nous exécutons une instruction du processus en cours d'exécution à l'aide de la fonction *gestion_instruction*
        - Si la valeur de retoure de la fonction *gestion_instruction* est égale à **2** c'est à dire que le processus est passé en état en attente et qu'il a déclenché une interruption, nous sortons de la boucle et nous passons au processus suivant prêt à s'exécuter
        - Sinon, nous continuons à traiter les instructions du processus en cours
    - Quand le temps CPU du processus est terminé alors, nous incrémentons le compteur ordinale et nous prenons à nouveau un processus prêt à s'exécuter et nous refaisons le même traitement.

        - **Ordonnancement Moins de temps processus ordonnancement_moins_CPU**

Dans cette stratégie d'ordonnancement, nous favorisons les processus qui nécessitent moins de temps processeur, ceux qui ont presque terminé leur programme, la fonction premier_proc_moins_CPU nous retourne le processus qui nécessite le moins de temps cpu parmi les processus prêt à s'exécuter, cette fonction parcourt tous les processus et parmi ceux qui sont prêt à s'exécuter elle cherche le minimum d'instruction restantes (le nombre d'instruction - le pointeur de pile)

- **Ordonnancement Moins appel bloquante**

Nous avons voulu aussi implémenter un algorithme qui favorise les processus qui font moins d'appels bloquants, nous pourrions rajouter dans la structure processus le nombre d'appel bloquant, qui sera initialisé à 0 au départ et s'incrémente à l'exécution de chaque instruction bloquante. A chaque tour de cycle d'horloge, nous choisissons parmi les processus prêt à s'exécuter le processus qui a fait le moins d'appel bloquant. (cette fonctionnalité n'a pas été implémenté suite au manque de temps)

<u>Ressources et Communication :</u>

- **Structure Disque :**
    - Type de disque (0 type caractere 1 type bloc)
    - Etat (0 en action, pas disponible 1 en arret, disponible)
    - Le nombre de plateaux
    - Le nombre de secteur par plateau
    - La taille d'un secteur

- Temps de passage par secteur
- Temps de passage par plateau
- Temps de transfert
- Type d'operation (0 lecture 1 écriture)

- **Éléments rajoutés dans la structure Instruction :**
  - Un entier lecture
    - **0** pour dire ce n'est pas une opération de lecture
    - **1** pour dire que c'est une opération de lecture asynchrone
    - **2** pour dire que c'est une opération de lecture synchrone
  - Un entier écriture
    - **0** pour dire ce n'est pas une opération d'écriture
    - **1** pour dire que c'est une opération d'écriture asynchrone
    - **2** pour dire que c'est une opération d'écriture synchrone
  - Un tableau pour désigner la zone mémoire lu

- **les type d'interruption ajouter :**
  - FIN_OPERATION_DISQUE indice 4
  - FIN_OPERATION_ASYNCHRONE indice 5

Dans l'initialisation aléatoire des instruction nous faisons en sorte qu'une instruction est soit une opération de lecture soit d'écriture mais pas les deux en même temps.

- **<u>Les fonctions utilisées :</u>**

  ➢ **<u>Gestion Instruction :</u>**
  - A chaque instruction de lecture ou d'écriture nous regardons l'état du disque, si ce dernier n'est pas disponible alors nous affichons une erreur, et nous plaçons le processus à la fin des processus prêt à s'exécuter
  - Si l'instruction est une instruction de lecture synchrone, nous bloquons le traitement par une boucle, nous générons à chaque fois un nombre aléatoire, et s'il est inférieure à 50 alors nous sommes toujours en train de lire ce qui nous pousse à attendre, si nous sommes supérieur ça implique que la lecture est finie, nous incrémentons le pt_pile, nous remettons le disque à l'état disponible, et nous retournons 1

- Si l’instruction est en mode lecture asynchrone, nous changeons l'état du disque à ‘non disponible’, nous incrémentons le pt_pile du processus courant, nous remettons l’etat disponible du disque et nous retournons 1
- Si l’instruction est en mode écriture synchrone, nous bloquons le traitement par une boucle, nous générons à chaque fois un nombre aléatoire et s’il est inférieure à 50 alors nous sommes toujours en train d'écrire donc nous procédons à une courte attente, si le nombre est supérieure à 50 alors l’instruction d’écriture est finie, nous incrémentons le pt_pile, nous remettons le disque à l'état disponible, et nous retournons 1
- Si l’instruction est en mode écriture asynchrone, nous changeons l'état du disque à ‘non disponible’, nous incrémentons le pt_pile du processus courant, nous remettons l’état disponible du disque et nous retournons 1

➢ **Gestion interruption :**

Dans la fonction de gestion interruption nous avons ajouté le traitement suivant:

- Si l’interruption est de type FIN_OPERATION_DISQUE nous changeons l'état du disque à disponible