INF3172 : TP2

AZIZ SALAH

1. Objectif pédagogique du travail

Algorithmes d'ordonnancement.

2. Description du problème

```
dispatch -- Simule un algorithme d'ordonnancement de la famile des
algorithmes d'ordonnancement feedback

SYNOPSIS
--------
       dispatch <quantum_F0> <quantum_F1> <fichier>

DESCRIPTION
-----------

fichier <fichier> décrit les processus sur lesquels l'algorithme
feedback (décrit ci-dessous) va opérer. Les quantums des files F0 et F1
sont donnés respectivement par <quantum_F0> et <quantum_F1> qui doivent
être des entiers strictement positifs.

<fichier> est un fichier (texte) d'entiers séparés avec des blancs où
chaque ligne  décrit un processus selon l'interprétation suivante:
a- Le premier entier de la ligne représente le PID du processus. Ce PID
doit être strictement positif car 0 est le PID du processus IDLE qui
occupe la CPU lorsqu'aucun autre processus n'est ready.
b- Le deuxième entier de la ligne représente le temps d'arrivée du
processus. Celui-ci doit être positif ou nul.
c- Les entiers restants dans la ligne représentent, s'ils sont
strictement positifs, des durées de temps CPU et, s'ils sont strictement
négatifs, des durées de blocage pour E/S. Des durées de temps CPU
successives se regroupent en leur somme.

L'expression régulière suivante décrit la  forme générale d'une ligne
de <fichier>:

            <pid> <temps_arrivée> <cpu> ( <cpu> | (-<e_s> <cpu>) )*

  où tous les jetons sont des entiers ; et où * veut dire 0, une ou plusieurs occurrences
de l'expression concernée

Voici quelques exemples de lignes respectant la forme ci-dessus
12 0 12 -12 12
```

```
9  2 3 4
7  2 1 1 -3 4 -4 1 1
5  2 1 -3 4 -4 1
```

Selon ce qui précède (que les temps cpu s'additionnent), les trois lignes suivantes sont équivalentes:

```
7  5 1 1 -3 6 -4 1 1
7  5 2   -3 4 2 -4 2
7  5 2   -3 6 -4 2
```

Ces lignes signifient que le processus de pid 7 arrive à l'instant 5 et passe à l'état ready. Après un temps cpu de 2 unités de temps, il sera bloqué pendant 3 unités de temps avant passer à ready. Ensuite, après un temps cpu de 6 unités de temps, il sera bloqué pendant 4 unités de temps avant passer à ready. Enfin, il termine après un temps cpu de 2 unités de temps.

L'algorithme feedback à simuler est celui vu dans le cours. Il utilise trois files F0, F1 et F2. Nous rappelons que les files sont gérées selon la politique que le premier à entrer dans la file est le premier à sortir de la file. L'algorithme feedback est décrit par l'ensemble des exigences suiavantes:

1- La file F0 a un quantum q0 et la file F1 un quantum q1
2- La file F0 est plus prioritaire que F1 et F2
3- De même, F1 est plus prioritaire de F2
4- Les processus de F1 n'ont accès à la CPU que si F0 est vide
5- Les processus de F2 n'ont accès à la CPU que si F0 et F1 sont vides
6- Dès qu'un processus passe à l'état ready, il entre dans la file F0
7- Le processus en tête de la file F0 qui a complété un quantum q0 dans la cpu, quitte F0 pour entrer dans la file F1 s'il a encore besoin de la cpu
8- Un processus en tête de F1 qui a complété quantum q1 dans la cpu, sort de F1 pour entrer dans F2 s'il a encore besoin de la cpu
9- Le processus en tête de la file F2 peut prendre la cpu jusqu'à son prochain changement d'état
10- On suppose qu'on a une priorité avec préemption : l'arrivé d'un processus plus prioritaire que celui qui a la CPU cause sa préemption
11- Tout processus préempté reste à la tête de sa file pour ne pas perdre sa priorité
12- Un processus de F1 qui été préhempté avant la fin de son quantum aura le droit de compléter son quantum lors de son prochain accès à la CPU
13-Un processus qui passe à l'état bloqué quitte la CPU (et sa file) mais, par la suite, il rejoint F0 dès qu'il passe à l'état ready
14- Un processus ne quitte pas forcément la CPU lorsqu'il passe d'une file à l'autre
15- Si au même moment deux processus ou plus doivent entrer dans une même file, on favorise celui qui le plus petit PID

3. Exemple d'exécution

Voici un exemple de fichier des processus qui va servir pour donner un exemple d'exécution :

```
16 10 4 -2 23
2 5 11 10
```

Le programme `dispatch` calcule l'ordonnancement de la population des processus spécifiée dans le fichier reçu comme paramètre selon l'algorithme décrit. L'affichage des résultats du programme `dispatch` doit ressembler à l'exemple suivant :

```
java > cat  exemple.txt
16 10 4 -2 23
2 5 11 10
java > wc -l exemple.txt
       2      exemple.txt
java > # wc montre bien que le fichier contient deux lignes
java > ./dispatch 5 10 exemple.txt
PID           0 |           0 |           5
-----------------------------------------------
PID           2 |           5 |           5
-----------------------------------------------
PID          16 |          10 |           4
-----------------------------------------------
PID           2 |          14 |           2
-----------------------------------------------
PID          16 |          16 |           5
-----------------------------------------------
PID           2 |          21 |          8
-----------------------------------------------
PID          16 |          29 |          10
-----------------------------------------------
PID           2 |          39 |           6
-----------------------------------------------
PID          16 |          45 |           8
-----------------------------------------------
```

Le processus de `PID 0` (IDLE) occupe la CPU lorsque aucun processus prêts n'est disponible. De plus, par exemple «`PID 16 | 10 | 4`» veut dire le processus de PID 16 entre dans la CPU à l'instant 10 pour une duré de 4 unités. Le programme doit effectuer l'affichage des résultats en utilisant strictement les fonctions d'affichage fournies.

Voici un deuxième exemple qui est très simple :

```
java > cat  exempleSimple.txt
2 0 10 20
java > ./dispatch 2 10 exempleSimple.txt
PID           2 |           0 |          30
-----------------------------------------------
```

Voici maintenant quelques cas d'erreurs :

```
java > ls exempleXYZ.txt
ls: exempleXYZ.txt: No such file or directory
java > ./dispatch 5 10 exempleXYZ.txt
```

```
Erreur : erreur ouverture fichier
java >  ./dispatch 5a 10 exemple.txt
Erreur : parametres incorrects
java >  ./dispatch 10 exemple.txt
Erreur : parametres incorrects
java > cat  exemple2.txt
16 10 4 -2 -23 1
2 5 34 11 10
java > ./dispatch 5 10 exemple2.txt
Erreur : fichier corrompu
```

On ne peut pas avoir deux nombres négatifs de suite ! Cette erreur trace les cas où le fichier ne respecte pas la forme des données spécifiée.

```
java > cat  exemple3.txt
16 10 4 2 -23 6
2 5 34k 11 10
java > ./dispatch 5 10 exemple3.txt
Erreur : fichier corrompu
```

`34k` n'est pas un entier.

4. Exigences non fonctionnelles

4.1. **Gestion des erreurs.** Seules les erreurs considérées par la fonction `print_erreur` sont à traiter dans votre programme.

4.2. **Gestion de l'allocation dynamique.** Le programme doit s'adapter dynamiquement avec la taille des données en utilisant l'allocation dynamique. Tout espace alloué dynamiquement doit être libéré dès qu'il n'est plus requis.

4.3. **Composition du programme.**

(1) Code source
- `«utils.h»` : définit les prototypes de fonctions utilitaires à utiliser pour l'affichage. Le fichier `«utils.h»` ne doit pas être modifié.
- `«utils.c»` : est une implémentation complète qui vous fournit les fonctions utilitaires pour l'affichage. Les fonctions fournies dans ce fichier ne doivent pas être modifiées.
- `«dispatch.c»` : comporte la fonction `main` et vos fonctions selon votre conception. C'est le seul fichier à remettre dans moodle.
- Lors de son évaluation par le correcteur, la compilation de votre programme `«dispatch.c»` se fera par la commande :
  `gcc -Wall -std=c99 dispatch.c utils.c -o dispatch`

(2) La compilation de votre programme ne devrait donner aucun avertissement (warnning).

(3) Votre fichier `«dispatch.c»` doit contenir les noms des auteurs et leurs codes permanents en commentaire selon le gabarit suivant :

```
/*

NOM, PRENON : un_nom, un_prenom
C.P. : AAAA99999999

NOM, PRENON : un_nom, un_prenom
C.P. : AAAA99999999
```

```
*/
```

(4) Votre programme n'accepte que les arguments tels que spécifiés.

(5) L'affichage doit respecter les exemples selon les cas d'utilisation fournis.

(6) Si votre programme «`dispatch.c`» rencontre une erreur, il doit afficher seulement un message d'erreur avec `print_erreur` et quitter avec le code d'erreur.

(7) Tout affichage doit se faire seulement avec les fonctions fournies :
- `print_element`
- `print_erreur`

4.4. **Portabilité du programme.** Afin de s'assurer de la probabilité de votre programme, celui-çi devrait être compilé et testé sur le serveur java (zéta2 est en panne maintenant).

5. Ce que vous devez remettre

En équipe d'au plus deux personnes, un membre de l'équipe seulement remettra votre fichier «`dispatch.c`» électriquement dans Moodle en suivant le lien approprié. Pas de remise en double svp.

6. Pondération

- Tests de fonctionnement : 70%
- Exigences non fonctionnelles : 15%
- Structure du programme, commentaires, indentation ... : 15%

Remarques importantes :

- Un programme ne compilant pas se verra attribuer la note 0.
- Aucun programme reçu par courriel ne sera accepté. En cas de panne des serveurs, un délai supplémentaire vous sera accordé sans pénalité de retard.
- Les règlements sur le plagiat seront strictement appliqués.
- 10% comme pénalité de retard par journée entamée. Après cinq jours, le travail ne sera pas accepté.
- La remise d'un fichier "zippé" donne lieu à 10% de pénalité.
- Un manquement aux exigences pourrait vous exposer à des pénalités. Votre collaboration est fortement appréciée.