Ordonnancement

Samy Jaziri

Sujet de khôlle inspiré du sujet du Concours Centrale Option Informatique 2017
et du TD de Frédéric Sur (LORIA)

Préambule

Tous les algorithmes de ce sujet devront être implémentés en ***C****.*

Pour vous mettre dans les conditions du concours, vous n'avez pas le droit d'accéder aux ressources en ligne.

Vous indiquerez vos réponses sur la fiche réponse qui vous a été fournie en utilisant, en entrée de vos programmes, le numéro u_0 inscrit sur cette fiche. Vous remettrez cette fiche à l'examinateur en fin de séance. Une fiche réponse est mise à disposition à la fin du sujet en tant qu'exemple des réponses attendues pour un $\widetilde{u_0}$ particulier.
En ce qui concerne les questions orales de la khôlle, lorsque la description d'un algorithme est demandée, vous devez présenter son fonctionnement de fa'c'on schématique, courte et précise. Vous ne devez pas expliquer votre code ligne par ligne! Quand la complexité d'un algorithme est demandée en temps ou en mémoire en fonction d'un paramètre n, on demande l'ordre de grandeur en fonction du paramètre, donné en notation de Landau ($\mathcal{O}(n), \mathcal{O}(\log(n)), \dots$). Prenez des notes lorsque vous préparez une question orale pour retrouver plus rapidement les grandes lignes de votre explication lorsque l'examinateur passe vous voir.
Il est recommandé de **tester vos programmes sur des petits exemples** que vous aurez résolus préalablement à la main ou bien à l'aide de la fiche réponse type fournie en annexe.
Il vous est demandé d'aborder les questions dans l'ordre et de noter vos difficultés à répondre à une question avant de passer à la suivante. Vous pourrez alors les aborder avec l'examinateur.
Attention, une attention particulière sera portée aux problèmes de fuite de mémoire lorsque vous utiliserez l'allocation dynamique
Les deux parties sont indépendantes.

1 Introduction

Motivation

« En informatique, l'ordonnancement est le fait d'ordonner des tâches à exécuter selon certaines contraintes. Ces tâches peuvent être l'exécution d'opération sur les entrées-sorties ou le traitement de processus. Les contraintes peuvent être temporelles ou dimensionnelles.» – *Wikipedia*
On considérera dans ce sujet deux processeurs sur lesquelles sont exécutées des tâches représentées par :

- un identifiant
- les ressources nécessaires à l'exécution de la tâche.

On souhaite implémenter une file d'attente pour chaque processeur dans lesquelles seront ajoutés les processus à exécuter. Puis on développera un algorithme permettant de répartir au mieux les tâches entre les deux processeurs.

Préliminaires

Un processus sera représenté par le type suivant :

```
typedef unsigned int uint;
typedef struct processus {
    uint id;
    uint cout;
} processus;
```

Question 1

Implémentez une fonction `processus* creer_processus(uint,uint);` renvoyant un pointeur vers un processus alloué dynamiquement et initialisé avec l'identifiant et le coût passé en paramètre.

2 Files d'attente

Structure de file avec tableau redimmensionnable

On veut implémenter une file d'attente à l'aide d'un vecteur circulaire. On définit pour cela une structure de *file* :

```c
#include <stdbool.h>
typedef struct file {
    processus** contenu;
    uint capacite;
    uint debut;
    uint fin;
    uint charge;
    bool vide;
} file;
```

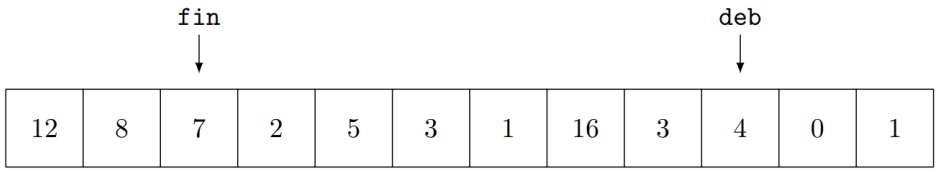

Figure 1: File `[4,0,1,12,8]`

`debut` indique l'indice du premier élément dans la file et `fin` l'indice qui suit celui du dernier élément de la file. `vide` indique si la pile est vide. Les éléments sont rangés depuis la case `debut` jusqu'à la case précédant `fin` en repartant à la case `0` quand on arrive au bout du vecteur. Ainsi, on peut très bien avoir l'indice `fin` plus petit que l'indice `debut`. `contenu` est un tableau **redimensionnable** de pointeurs vers des processus de taille égale à `capacite`. `charge` correspond à la somme des coûts des processus contenus dans la file.

Question 2

Implémenter les opérations suivantes pour la structure de file défini ci-dessus :

- `file* creer_file_vide()` renvoie un pointeur vers une file vide allouée dynamiquement.
- `void detruire_file(file*)` libère la mémoire utilisée par une file.
- `processus* retire(file*)` retire un processus de la file ou `NULL` si elle est vide.
- `void ajoute(file*, processus*)` ajoute un processus dans la file. Le tableau devra être redimensionné si nécessaire. On fera en sorte d'avoir une complexité amortie en $\mathcal{O}(1)$ pour cette fonction.

Processus aléatoires

Considérons $(u_k)_{k\geq 0}$ la suite d'entiers définie par :

$$u_k = \begin{cases} u_0 \text{ (sur votre fiche réponse)} & \text{si } k = 0 \\ 15091 \times u_{k-1} \mod 64007 & \text{si } k > 0 \end{cases}$$

On définit pour tout $n \in \mathbb{N}$, la suite $(p_k^n)_{k\geq 0}$ de processus par :

$$p_k^n = (u_k, r_k^n) \text{ avec } r_k = (u_k \mod 10) * (n \mod 10 + 1)$$

Où u_k est l'identifiant du processus p_k et r_k son coût en ressources. On définit ensuite la suite d'opérations $(o_k)_{k\geq 0}$ par

$$o_k = \begin{cases} \texttt{0} & \text{si } u_k \equiv 0[21] \\ \texttt{1} & \text{sinon} \end{cases}$$

On définit la file d'attente $\mathcal{F}_k$ pour $k \geq 0$ comme la file dans laquelle on a réalisé, dans l'ordre pour chaque i allant de 0 à k inclus, les opérations suivantes :

- Si $o_i = 0$ alors on retire un élément de la file *si cela est possible.*
- Si $o_i = 1$ alors on ajoute le processus p_i^k dans la file.

Question 3

Implémenter une fonction qui construit $\mathcal{F}_k$ en $\mathcal{O}(k)$. On considérera pour cette fonction que l'insertion dans une file est une opération en temps constant.

Question 4

Pour chaque $k \in \mathbb{N}$ ci-dessous donnez la capacité de la file $\mathcal{F}_k$, le nombre d'éléments de la file $\mathcal{F}_k$ et l'identifiant du processus situé au milieu de la file.
a) 1212 **b)** 1000 **c)** 1256

3 Equilibrage de charge.

On souhaite répartir une liste de processus $p_1, \ldots, p_n$ sur deux processeurs de manière à minimiser l'écart entre les ressources utilisées par chaque processeur. Formellement soit $\mathcal{P}$ un ensemble de processus. On définit $C(\mathcal{P})$ la somme du coût en ressource de chaque processus de $\mathcal{P}$. Soit A_1 et A_2 une partition de $\mathcal{P}$: A_1 est l'ensemble des processus qui seront exécutés par le premier processeur et A_2 les processus exécutés par le second. On cherche A_1 et A_2 tel que $|C(A_1) - C(A_2)|$ soit minimal. Une telle partition sera nommée *partition optimale*. La valeur $|C(A_1) - C(A_2)|$ pour une partition A_1, A_2 (pas forcément optimale) serra nommé *différence de charge*.

> *Préparer une réponse à donner à l'oral*
>
> Montrez qu'une partition A_1, A_2 minimise la différence de charge si et seulement si la partition minimise le maximum des charge, *i.e* la quantité $|\max(C(B_1), C(B_2))|$ pour toute B_1, B_2 une partition.

On propose d'utiliser une technique de programmation dynamique pour trouver la partition optimale d'un ensemble de processus $\mathcal{P} = \{p_1, \ldots, p_n\}$.
On définit le sous-problème suivant : Soit $0 \leq k \leq n$, étant donné une répartition B_1, B_2, de $p_1, \ldots p_k$ (pas forcément optimale) telle que $\Delta = |C(B_1) - C(B_2)|$, $\mu(k, \Delta)$ est la différence de charge minimale atteignable en répartissant les $n - k$ derniers processus.

> *Préparer une réponse à donner à l'oral*
>
> Que vaut $\mu(0, 0)$? Que vaut $\mu(n, \Delta)$?

Préparer une réponse à donner à l'oral

Etablir une relation de récurrence (ascendante en k) sur μ.

Préparer une réponse à donner à l'oral

Imaginer un algorithme récursif (sans l'implémenter) permettant de calculer la partition optimale d'un ensemble de processus. Quel serait sa complexité temporelle au pire.

Question 5

Implémenter une fonction `uint difference_optimale(file* p);` qui prend en paramètre un pointeurs vers des files de processus `p` et calcul $\mu(0,0)$ pour `p`.

Préparer une réponse à donner à l'oral

Quelle est la complexité spatiale et temporelle au pire de votre algorithme. On considérera pour ce calcul que l'insertion dans une file est une opération en temps constant.

Question 6

Pour chaque $k \in \mathbb{N}$ ci-dessous donnez la différence de charge optimale obtenue en répartissant la file $\mathcal{F}_k$ dans deux processeurs n'exécutant initialement aucun processus.
a) 1212 **b)** 1000 **c)** 1256

Question 7

Implémenter une fonction `void equilibrage(file* p, file* a1, file* a2);` qui prend en paramètre trois pointeurs vers des files de processus `p`, `a1` et `a2`. La fonction répartira les processus de `p` dans les files `a1` et `a2` de manière à minimiser la différence de charge.

Question 8

Pour chaque $k \in \mathbb{N}$ ci-dessous donnez pour chaque processeur n'exécutant initialement aucun processus, la somme modulo 12000 des identifiants de chaque processus alloués à ce processeur dans la repartition optimale de la file $\mathcal{F}_k$.
a) 1212 **b)** 1000 **c)** 1256

Fiche réponse : Ordonnancement

Nom, prénom : DOUZ Nadine $\widetilde{u_0}$: 12

Question 4:

a) (2048, 1101, 29742)

b) (1024, 917, 16175)

c) (2048, 1141, 15934)

Question 6:

a) 3

b) 0

7

Question 8:

c) (7251, 7248)

d) (2005, 2005)

e) (5570, 5563)