Final Examination
Practical Portion
EEE243 Applied Computer Programming
13 December 2016

Examiner: S. Givigi, PhD

Instructions

- You may begin the practical portion of the exam immediately.
- The practical portion is out of 25 marks.
- The practical portion of this examination is open book. You may refer to course notes and your code.
- Communication with any person other than the invigilators during the practical portion of the test **is forbidden**.
- You will not receive an exam booklet for the practical portion of the test; you have received some notepaper instead.
- If you wish your notepaper to be marked by the examiner, write your name and student number on it and turn it in at the end of the exam.
- Ask the invigilator for more notepaper if you require it.

Hints: Compile and execute your program **often**. You will lose marks if your program does not compile and execute as submitted, or if it generates compiler warnings. Test your code as you go. Start small and build from there.

Draw memory diagrams of all pointer manipulations before writing the code!

Examen Final
Partie Pratique
GEF243 Programmation informatique appliquée
13 décembre 2016

Examinateur: Capt Adrien Lapointe, MSc

Instructions

- Vous pouvez commencer la partie pratique de l'examen tout de suite.
- La valeur de la partie pratique est de 25 points.
- La partie pratique de l'examen est à livre ouvert. Vous pouvez faire référence aux notes du cours et à votre code.
- La communication avec toutes personnes autre que les surveillants pendant la partie pratique de l'examen **est interdite**.
- Vous ne recevrez pas de livret d'examen pour la partie pratique du test; vous avez plutôt reçu des pages de papier de notes.
- Si vous voulez que les pages de notes soient corrigées par l'examinateur, écrivez-y votre nom et numéro de collège et remettez-les à la fin de l'examen.
- Nous avons amplement de papier de notes, demandez-en plus si vous en avez besoins.

Indices : Compilez et exécutez votre code **souvent**. Vous serez pénalisé si votre programme ne compile et n'exécute pas ou s'il produit des avertissements. Testez votre code souvent. Commencez petit et ajoutez des fonctionnalités au fur et à mesure.

Dessinez des diagrammes de mémoire pour toutes les manipulations de pointeurs avant de coder.

Introduction

Open the project that you created at the beginning of the examination.

If you haven't already done so, **add your name** to the comment block at the beginning of the `main.c` file.

You must provide evidence of your design for each part of the tasks. Clear, well-structured code that works and includes appropriate comments is sufficient. However, if your code does not work or is not clear or well structured, you may receive part marks for design documentation in the form of flowcharts, pseudo-code, pointer or memory diagrams, or textual descriptions of your design. Write these on the provided notepaper.

You have been provided with three files: `main.c`, `sched.c` and `sched.h`. You are **not permitted** to make any changes to `sched.c` and `sched.h`.

Overview

The code provided is a partial implementation of an Operating System scheduler.

As the processes (`Process`) are created by the `create_process` function, they are added to one of the three queues according to their type, and they are scheduled according to each of their priority.

Your task will be to write code to complete this partial implementation of the simulation

Queues representation

The queues are implemented as three separate doubly linked lists.

Each entry in the queues is an instance of the `Process` structure that is defined in `sched.h`.

The queues are shown in figure 1 below. Note that not all the processes fields are represented in the figure.

The head of each of the three queues are kept in an array of pointers called `queues`. The queues are used to sort the processes according to their type:

Introduction

Ouvrez le projet que vous avez créé au début de l'examen.

Si vous ne l'avez pas encore fait, **ajoutez votre nom** au bloc commentaire au début du fichier `main.c`.

Vous devez fournir des traces de votre conception pour chaque partie des tâches. Du code clair et bien structuré qui fonctionne et qui inclus des commentaires appropriés est suffisant. Toutefois, si votre code ne fonctionne pas ou s'il n'est pas clair ou bien structuré, vous pourrez recevoir des points partiels pour la documentation de conception sous la forme d'organigrammes, de pseudo-code, de schémas de la mémoire ou des pointeurs, ou de descriptions textuelles de votre conception. Écrivez-les sur le papier de notes fournit.

Nous vous fournissons trois fichiers : `main.c`, `sched.c` et `sched.h`. Vous **ne devez pas** modifier les fichiers `sched.c` et `sched.h`.

Vue d'ensemble

Le code fournit est une implémentation partielle de l'ordonnanceur (*scheduler*) d'un système d'exploitation.

À mesure que des processus (`Process`) sont créés par la fonction `create_process`, ceux-ci sont ajoutés à l'une des trois files d'attente selon leur type, puis ordonnancé (*scheduled*) selon leur priorité respective.

Votre tâche est d'écrire le code pour compléter cette implémentation partielle de la simulation.

Représentation des files

Les files d'attente sont représentées trois listes doublement chaînées séparées.

Chaque entrée d'une file est une instance de la structure `Process` qui est définie dans `sched.h`.

Les files sont illustrées dans la figure 1 ci-dessous. Notez que tous les champs des processus ne sont pas illustrés.

La tête de chacune des trois files est stockée dans un tableau de pointeurs appelé `queues`. Les files permettent de trier les processus selon leur type :

BACKGROUND, INTERACTIVE and REALTIME. When a queue is empty, its corresponding pointer in queues is NULL as can be seen in the third element of queues in figure 1.

The bi-directional structure of the list is maintained with the next and prev pointers in the Process structure. The prev pointer of the first node in a queue is NULL, as is the next pointer of the last node.

New entries are added to the queues based on their type in increasing order of priority (the field priority of the structure).

BACKGROUND, INTERACTIVE et REALTIME. Lorsqu'une file est vide, le pointeur respectif dans queues est NULL comme c'est le cas pour le troisième élément de queues dans la figure 1.

La structure bi-directionnelle de la liste est maintenue grâce aux pointeurs next et prev dans la structure Process. Le pointeur prev du premier nœud d'une file est NULL, tout comme le pointeur next du dernier nœud de la file.

Les nouvelles entrées sont ajoutées à la file correspondant à leur type et en ordre croissant de priorité (le champ priority de la structure).

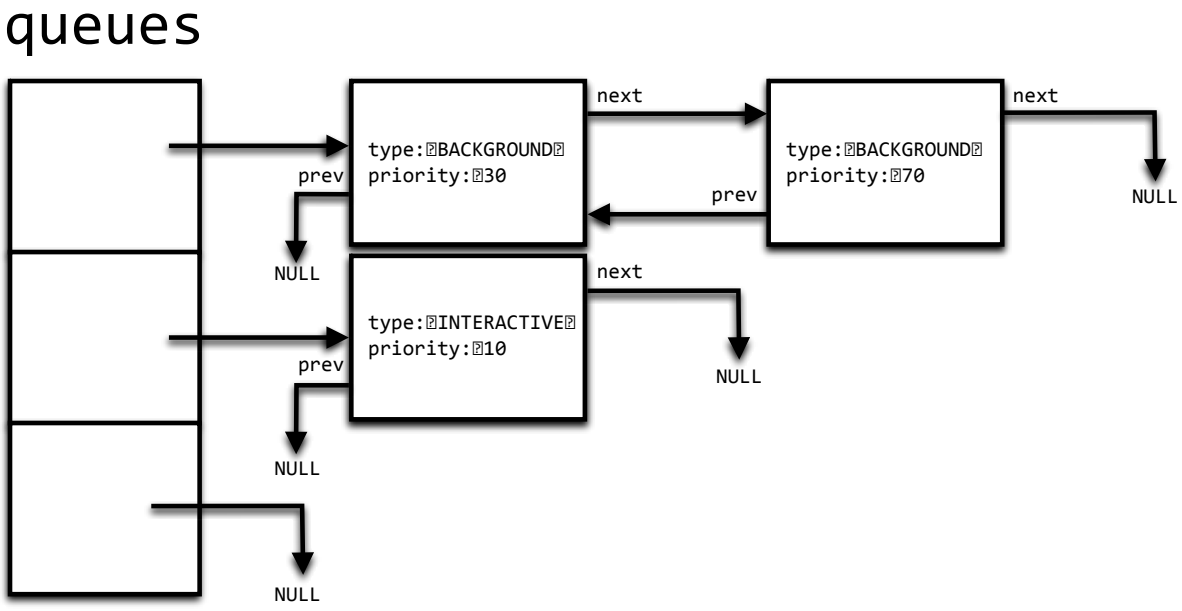

Figure 1

You must complete the following tasks. We suggest you complete them in the order listed. As you complete each task, write test code in your main function to ensure your implementations work as intended. You may comment out this test code and leave it in the main function when it is no longer required.

Note that the last task is to provide particular code in your main.

Vous devez compléter les tâches suivantes. Nous vous recommandons de les faire dans l'ordre où elles apparaissent. Écrivez du code de test dans votre fonction main à mesure que vous compléter les tâches pour vous assurer que l'implémentation fonctionne. Vous pouvez mettre ce code de test en commentaires et le laisser dans la main lorsqu'il n'est plus nécessaire.

Notez que la dernière tâche indique le code particulier à mettre dans votre main.

Task 1: Add processes to a queue [5]

Implement the add_in_order function so that it adds a new process (the parameter new) to the queue

Tâche 1 : Ajouter des processus à une file

Implémentez la fonction add_in_order afin qu'elle ajoute un nouveau processus (le paramètre new) à la

whose head is the parameter `head` (`NULL` if the queue is empty). New processes are added to the queue in increasing order of priority. Your code must correctly maintain the doubly linked structure previously illustrated.

You can generate new processes by calling the `create_process` function. Task 2 provides a hint that will be useful for testing.

Task 2: Print a queue **[2]**

Complete the implementation of the `print_queue` function so that it prints all the processes in a given queue. You can use the provided functions `print_one_process` to print an individual process entry and `print_column_headings` to print the queue heading.

Task 3: Add to the appropriate queue **[2]**

Implement the `add_to_queues` function to add processes to the appropriate queue. Each process type (`BACKGROUND`, `INTERACTIVE` and `REALTIME`) is assigned a queue. For instance, the `BACKGROUND` type is assigned to the first queue.

Task 4: Print all the queues **[2]**

Implement the `print_all_queues` function so that it prints the content of each queue. Make sure you print an indication that makes it clear which queue is displayed.

Task 5: Remove the highest priority process **[3]**

Implement the `remove_process` function so that it removes the process with the highest priority, which is the process with the **lowest** `priority` value.

Task 6: Remove all entries **[3]**

Implement the `schedule` function so that it schedules all the processes. Scheduling is done by removing a process from a queue, print a message telling which process is scheduled, and free the memory that was used by the process.

The processes with `REALTIME` type must be scheduled first, then the `INTERACTIVE`, and finally the `BACKGROUND`. If there is more than one process in a given queue, they must be scheduled in order of priority. The scheduler only goes to the next queue when one is empty. In figure 1, the first process that

file dont la tête est donnée par le paramètre `head` (`NULL` si la file est vide). Les nouveaux processus sont ajoutés à la file en ordre croissant de priorité. Votre code doit maintenir correctement la structure doublement chaînée de la liste tel qu'illustré précédemment.

Vous pouvez créer un nouveaux processus en appelant `create_process`. La tâche 2 fournit un indice utile pour vos tests.

Tâche 2 : Afficher une file **[2]**

Finissez l'implémentation de la fonction `print_queue` afin qu'elle affiche tous les processus contenus dans une file donnée. Vous pouvez utiliser les fonctions fournies `print_one_process` pour afficher un seul processus de la file et `print_column_headings` pour afficher l'en-tête de la file.

Tâche 3 : Ajouter à la bonne file **[2]**

Implémentez la fonction `add_to_queues` pour ajouter les processus à la bonne file. Chaque type de processus (`BACKGROUND`, `INTERACTIVE` et `REALTIME`) se voit assigner à sa propre file. Par exemple, le type `BACKGROUND` est assigné à la première file.

Tâche 4 : Afficher toutes les files **[2]**

Implémentez la fonction `print_all_queues` afin d'afficher le contenu de toutes les files. Assurez-vous d'afficher clairement de quelle file il s'agit.

Tâche 5: Retirer le processus prioritaire **[3]**

Implémentez la fonction `remove_process` afin qu'elle retire le processus ayant la plus haute priorité, représentée par la **valeur la plus basse** de `priority`.

Tâche 6: Ordonnancer tous les processus **[3]**

Implémentez la fonction `schedule` afin d'ordonnancer tous les processus. L'ordonnancement consiste à retirer un processus d'une des files, afficher un message indiquant quel processus est ordonnancé et libérer la mémoire occupée par ce processus.

Les processus du type `REALTIME` doivent être ordonnancés en premier, ensuite ceux du type `INTERACTIVE` et enfin ceux du type `BACKGROUND`. S'il y a plus d'un processus dans la file d'un certain type, ceux-ci doivent être ordonnancés en ordre de priorité. L'ordonnanceur ne passe à la file de type

would be scheduled is the only one in the `INTERACTIVE` queue, the one with a priority of 30 in the `BACKGROUND` queue, and finally the one with a priority of 70.

Task 7: Calculate the amount of used memory

Implement the `calculate_total_mem_GB` function so that it returns the amount of memory used by all the processes in gigabytes (1 GB = 1 048 576 kB).

Task 8: Enforce the memory size

Our system has a finite amount of memory (`MEM_SIZE_GB`). Modify your `add_to_queues` function so that it is not possible to add a new process if there is not enough memory left to store it in memory. If memory space is insufficient, the process must not be added and an error message is printed.

Task 9: main

Comment out any test code remaining in your `main` function and add code to do the following, using the functions you have implemented:

- Initialize the queues.
- Generate 20 processes, adding each one to the appropriate queue. If you have completed task 8, the used memory space should never exceed `MEM_SIZE_GB`. Print the queues, the used memory space and the remaining memory space after each process is added.
- Schedule the processes.
- Print the (now empty) queues.
- Exit cleanly.

When you are finished

Ensure your name is in the header comment of the main file and on your notes pages. Leave the computer logged in and your project open in Eclipse, and leave your notes pages on the keyboard. Leave quietly by the nearest door.

End of the examination

suivant que lorsqu'une file est complètement vide. Dans la figure 1, le premier processus qui serait ordonnancé serait le seul dans la file `INTERACTIVE`, puis celui avec une priorité de 30 dans la file `BACKGROUND` et enfin celui de priorité 70.

[2] Tâche 7: Calculer la mémoire utilisée

Implémentez la fonction `calculate_total_mem_GB` afin qu'elle retourne la quantité de mémoire utilisée par tous les processus en giga-octets (1 Go = 1 048 576 ko).

[3] Tâche 8: Faire respecter l'espace mémoire

Notre système a une quantité de mémoire finie (`MEM_SIZE_GB`). Modifiez votre fonction `add_to_queues` afin qu'il ne soit pas possible d'ajouter de nouveaux processus s'il n'y a pas suffisamment d'espace mémoire pour les contenir. Si l'espace mémoire est insuffisant, le processus n'est pas ajouté et un message d'erreur est affiché.

[3] Tâche 9: main

Mettez en commentaire tout code de test restant dans votre fonction `main` et ajoutez le code faisant ce qui suit en utilisant les fonctions que vous avez implémentées :

- Initialisez les files.
- Créez 20 processus et ajoutez-les à la file appropriée. Si vous avez complété la tâche 8, l'espace mémoire utilisé ne devrait jamais dépasser `MEM_SIZE_GB`. Affichez les files, l'espace mémoire utilisé et l'espace mémoire restant après l'ajout de chaque processus.
- Ordonnancez les processus.
- Affichez les files (maintenant vides).
- Sortez proprement.

Lorsque vous avez terminé

Assurez-vous que votre nom est dans le bloc commentaire des fichiers et sur vos pages de notes. Laissez l'ordinateur connecté et votre projet ouvert dans Eclipse. Laissez aussi vos pages de notes sur le clavier. Partez discrètement par la porte la plus proche.

Fin de l'examen