Système d'Exploitation Avancé
Travaux Pratiques (2), Licence 2 Informatique
Ordonnancement de Tâches

Fonctions utiles au TP :
Celles définies dans la bibliothèque libsched présentée en annexe.

1. Tester la bibliothèque

1.1

Créez un répertoire `TP_Ordo`. Dans ce répertoire, copiez la bibliothèque libsched.

Note : la bibliothèque `libsched_m64.zip` est disponible sur Celene sur la page du cours à la section « Outils pour les Travaux Pratiques » sous le nom « Ordonnancement des Tâches ».

Sauvegarder la bibliothèque dans votre répertoire `TP_Ordo` et décompactez-la :

```
unzip libsched_m64.zip
```

1.2

Compilez l'exemple et testez

```
$ cd libsched_m64/demo
$ make main
$ ./main
```

2. Écriture d'un nouvel algorithme d'ordonnancement SJF

2.1

Éditez le fichier `scen.c` et complétez la nouvelle fonction d'élection `SJFElect` (sur le modèle de l'exemple de l'élection aléatoire) implémentant l'ordonnancement `SJF`.

2.2

Compilez et testez votre programme:

```
$ make scen
$ ./scen
```

3. Approximation de SJF en temps partagé

L'algorithme `SJF` suppose de connaître à l'avance le temps d'exécution d'une tâche (ce qui est rarement le cas dans les systèmes). On veut privilégier les tâches courtes sans connaître le temps estimé des tâches.

3.1

Écrivez la fonction `ApproxSJF` qui implante un algorithme privilégiant les tâches courtes en temps partagé. Indication : vous serez amenés à utiliser le champ `ncpu` qui indique le nombre de quantums de temps consommés par chaque tâche.

3.2

Testez votre programme avec un quantum d'une seconde. Comparez vos résultats avec l'algorithme aléatoire.

3.3

Votre algorithme peut-il provoquer une famine (c'est-à-dire une situation où une tâche prête n'est jamais élue) ? Si oui modifiez-le pour éviter ce problème.

Libsched : Mode d'emploi

La bibliothèque d'ordonnancement libsched permet de tester des algorithmes d'ordonnancement de fonctions utilisateur. L'utilisateur a l'illusion que ses fonctions s'exécutent en parallèle.
Grâce à libsched, on peut définir et paramétrer de nouveaux algorithmes d'ordonnancement.
Deux fichiers sont fournis par la bibliothèque : libsched.a et le fichier d'inclusion sched.h

1. Fonctions de la librairie

```
#include <sched.h>
```

int CreateProc(function_t func, void *arg, int duration);

Cette fonction permet de créer une nouvelle fonction (que l'on appelle processus léger) qui pourra s'exécuter en parallèle. Le paramètre func est le nom de la fonction, arg est un pointeur vers les arguments de la fonction et duration est la durée estimée de la fonction. Par défaut le paramètre duration n'est pas utilisé mais il peut être utile pour des algorithmes d'ordonnancement du type SJF (Shortest Job First).

CreateProc retourne l'identifiant du processus léger créé (pid).

void SchedParam (int type, int quantum, int (*felect)(void));

Cette fonction permet de régler les paramètres de l'ordonnanceur. type indique le type d'ordonnancement. 3 types sont possibles (définis dans sched.h) :

BATCH indique un ordonnancement sans temps partagé de type FIFO. Dans ce cas, les paramètres quantum et felect sont ignorés.

PREMPT indique un ordonnancement préemptif de type "tourniquet". C'est l'ordonnancement par défaut. Dans ce cas, le paramètre quantum fixe en seconde la valeur du quantum de temps.

NEW indique une nouvelle stratégie d'ordonnancement (définie par l'utilisateur). Dans ce cas, le paramètre quantum fixe en seconde la valeur du quantum de temps. Si quantum est égal à 0, l'ordonnancement devient non préemptif (sans temps partagé). Le paramètre felect est le nom de la fonction d'élection qui sera appelée automatiquement par la librairie avec une période de "quantum" secondes (si quantum est différent de 0).

La fonction d'élection felect doit avoir la forme suivante :

```
int Mon_election(void) {

/* Choix du nouveau processus élu */
    return elu;
}
```

La fonction d'élection choisit le nouveau processus élu (à l'état RUN) en fonction des

informations regroupées dans la table `Tproc` définie dans `sched.h` :

```
struct proc {
    int flag;          // Etat de la tâche : RUN, IDLE ou ZOMB
    int prio;          // Priorité entre MINPRIO (0) et MAXPRIO (100)
    int pid;           // Pid
    ...
    struct timeval end_time;           // date de fin
    struct timeval start_time;         // date de création
    struct timeval realstart_time;    // date de lancement
    double ncpu;                       // temps "cpu" consommé
    double duration;                   // temps estimé de la tâche
} Tproc[MAXPROC];
```

La priorité d'un processus varie entre les deux constantes `MINPRIO` et `MAXPRIO`. Plus la valeur de la priorité est élevée, plus le processus est prioritaire.

Une fois le processus choisi, `felect` doit retourner l'indice dans `Tproc` du processus élu.

`int GetElecProc(void);`

Fonction qui retourne l'indice dans `Tproc` du processus élu.

`void sched(int printmode);`

Cette fonction lance l'ordonnanceur. L'ordonnancement effectif des processus ne commence qu'à partir de l'appel à cette fonction. Par défaut l'ordonnanceur exécute un algorithme similaire à Unix à base de priorité dynamique. Le paramètre `printmode` permet de lancer l'ordonnanceur en mode "verbeux". Si `printmode` est différent de 0, l'ordonnanceur affichera à chaque commutation la liste des tâches prêtes. Cette fonction se termine lorsqu'il n'existe plus de tâche à l'état prêt (`RUN`).

`void PrintStat(void);`

Cette fonction affiche les statistiques sur les tâches exécutées (temps réel d'exécution, temps processeur consommé, temps d'attente).

2. Exemple

L'exemple suivant illustre l'utilisation des primitives.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <malloc.h>
#include <sched.h>

// Fonction utilisateur

void MonProc(int *pid) {
    long i;
    for (i=0;i<8E7;i++)
        if (i%(long)4E6 == 0)
            printf("%d - %ld\n",*pid, i);

    printf("############ FIN PROC %d\n\n", *pid );
}

// Exemple de primitive d'election definie par l'utilisateur
// Remarques : les primitives d'election sont appelées directement
// depuis la librairie. Elles ne sont appelées que si au
// moins un processus est à l'etat pret (RUN)
// Ces primitives manipulent la table globale des processus
// définie dans sched.h

// Election aléatoire

int RandomElect(void) {
    int i;

    printf("RANDOM Election !\n");

    do {
        i = (int) ((float)MAXPROC*rand()/(RAND_MAX+1.0));
    } while (Tproc[i].flag != RUN);

    return i;
}

int main (int argc, char *argv[]) {
    int i; int *j;

    // Créer 3 processus

    for (i = 0; i < 3; i++) {
        j = (int *) malloc(sizeof(int));
        *j= i;
        CreateProc((function_t)MonProc,(void *)j, 0);
```

```
    }
     // Exemples de changement de paramètres
     // Définir une nouvelle primitive d'election avec un quantum
     // de 2 secondes

     SchedParam(NEW, 2, RandomElect);

     // Redéfinir le quantum par defaut

     SchedParam(PREMPT, 2, NULL);

     // Passer en mode batch

     SchedParam(BATCH, 0, NULL);

     // Lancer l'ordonnanceur en mode non "verbeux"

     sched(0);

     // Imprimer les statistiques

     PrintStat();

     return EXIT_SUCCESS;
}
```

3. Utilisation de la bibliothèque au département d'Informatique

La `libsched` est disponible sur Celene dans la page du cours à la section « Outils pour les Travaux Pratiques » sous le nom « Ordonnancement de Tâches ».

Le répertoire `/libsched/demo` contient l'exemple précédent avec un `Makefile` permettant de générer directement l'exécutable.

Le répertoire `/libsched/src` contient le source de `libsched`. Il contient également la librairie "`libelf`" utilisée en interne par l'ordonnanceur.

Pour générer un exécutable, le plus simple est de copier le `Makefile` de l'exemple :

```
$ cp /libsched/demo/Makefile .
```

Modifiez ensuite éventuellement le `Makefile`, en remplaçant le nom des fichiers exemples par celui de votre fichier, puis générez un exécutable :

```
$ make
```

4. Pour installer la librairie chez soi

La librairie est disponible sous forme d'archive compressée sur Celene dans la page du cours à la section « Outils pour les Travaux Pratiques » sous le nom « Ordonnancement de Tâches ».

Pour la décompresser, il suffit de taper sur votre machine :

```
$ unzip libsched_m64.zip
```

Un répertoire `libsched` est créé contenant les deux répertoires `demo` et `src`. Il suffit de compiler la librairie :

```
$ cd libsched/src
$ make
```

Vous pouvez alors compiler l'exemple et l'exécuter :

```
$ cd ../demo
$ make
$ ./main
```

Pour tout commentaire, ou rapport de "bug" : Pierre.Sens@lip6.fr