LO41 A06
Projet 7

Montre digitale

Binôme : Haffner Alexandre
Monneret Nicolas

Enoncé :

Il s’agit de simuler le fonctionnement d’une montre digitale comprenant une horloge interne, mise à jour périodiquement par une horloge externe (universelle !), un afficheur et une interface utilisateur permettant de changer de fuseau horaire, de programmer des alarmes.
Les composants distribués de la montre seront traduits par des processus. Ceux-ci se synchroniseront pour adopter un comportement cohérent. L'horloge externe communiquera avec l'horloge interne à l'aide d'un tube ou d'une file de message (au choix). Les fonctions offertes à l'utilisateur seront prioritaires.
Analyse et démarche

1. Considérations générales

Dans un premier temps, nous avons réfléchi sur les différentes fonctionnalités de notre horloge :

- le changement d'alarme
- le changement de fuseau horaire
- le changement d'heure avec synchronisation un serveur externe.

A partir de ces fonctionnalités, nous en avons déduit les différents processus qui vont interagir entre eux :

- la montre digitale principale autonome
- un serveur autonome sur lequel la montre pourra se synchroniser
- un processus menu chargé de lancer l'un des trois processus suivants
- un processus de changement d'alarme
- un processus de changement de fuseau horaire
- un processus de réglage de l'heure, soit par synchronisation, soit manuellement.

La communication entre ces processus se fera grâce à deux files de messages :

- une file pour la communication avec l'horloge principale
- une file pour la communication avec le serveur

2. Définition des types

Nous avons ensuite choisi les différents types pour représenter les variables. Nous choisissons d'inclure les fichiers time.h et types.h pour pouvoir utiliser le type time_t afin de décrire l'horloge.

A partir de là nous avons créé le fichier mqueue_types.h qui défini les différents types de variables et de messages utilisés par la suite. On défini ainsi le type time_zone, afin de décrire le fuseau horaire.
On défini également trois structures pour les types de messages :

- msg_time, pour envoyer un temps de type time_t
- msg_byte, pour les messages de type demande ou accusé de réception
- msg_time_zone pour envoyer un fuseau horaire

Enfin, nous créons une énumération afin de différencier les messages envoyés. L'ensemble de ces messages correspondent en fait aux messages M0...M9 du réseau de Pétri.

3. L'horloge principale

Nous pouvons créer maintenant l'horloge principale et ces quelques fonctions. L'horloge principale peut être divisée en deux partie : la partie initialisation, qui créé la file de message grâce à la primitive msgget(), et qui, à l'aide de la fonction initAll(), initialise l'horloge, l'alarme et le fuseau en les demandant à l'utilisateur.

La deuxième partie est une boucle infinie while(true). Celle-ci est composée de trois fonctions qui scrutent la file de message pour savoir si l'alarme, le fuseau ou l'horloge sont à modifier : updateTimeZone(), updateAlarm() et synchronise().

Par la suite nous mettons à jour l'horloge grâce à la fonction updateClock() et nous affichons l'heure avec la fonction displayTime(). Avec la fonction ring(), on regarde si l'alarme doit agir, c'est-à-dire si elle est comprise dans la minute qui suit l'heure souhaitée et on affiche « Beep » si c'est le cas (avec la fonction beep()). On écrit également l'id de la file de message sur l'écran afin que l'utilisateur puisse l'utiliser dans le menu pour modifier les fonctionnalités.

Les fonctions updateTimeZone() et updateAlarm(), sont quasiment identiques. On scrute la file de message de l'horloge pour savoir si elle contient une nouvelle alarme (ou un nouveau fuseau) avec la primitive msgrcv(). Si c'est le cas, on prend en compte la nouvelle donnée et on envoi un nouveau message pour confirmer la prise en compte du changement avec la primitive msgsnd().

La fonction synchronise() est plus compliquée car elle nécessite la synchronisation de l'horloge. On scrute la file de message pour savoir si elle contient une demande de synchronisation. Si c'est le cas, on envoie un nouveau message sur la file pour confirmer la synchronisation, puis la fonction se met en attente du message contenant la nouvelle heure, change l'horloge et renvoie un message de confirmation de changement.

On a aussi créé un fichier clock_functions.c qui contient les fonctions displayTime(), updateClock() et setTime().
La première affiche l'heure sous la forme hh : mm : ss à partir d'une variable de type time_t.
La seconde, updateClock(), met à jour l'horloge. On utilise 2 variable statiques previous et current. Avec la fonction time(), on récupère un temps en seconde. A chaque fois, on obtient ainsi l'intervalle de temps entre 2 appels et on peut incrémenter l'horloge en conséquence.
Enfin, la fonction setTime() permet de convertir un temps de la forme hh : mm : ss en un temps en secondes de type time_t.

Nous avons donc, à cette étape du projet, le processus horloge principal ainsi que la première file de messages.

4. Les processus de modifications

Nous passons à l'étape suivante : la création des processus qui vont interagir avec l'horloge. Les 3 processus vont être contenus dans 3 fichiers :change_alarm.c, change_timezone.c et synchronise.c.

L'utilisateur devra appeler ces processus en passant l'id de la file de message de l'horloge en paramètre afin que ceux-ci puissent interagir avec l'horloge.

4.1. Changer l'alarme

Nous créons tout d'abord le fichier change_alarm.c, processus qui sert à modifier l'alarme.
On commence par vérifier que l'id de la file de message a bien été passé en paramètre. Ensuite, on demande à l'utilisateur la nouvelle alarme, on créée le message à envoyer qui contient cette nouvelle alarme et on l'envoie à l'horloge grâce à msgsnd. Avant de terminer, on attend la réception du message de confirmation par l'horloge avec msgrcv.

4.2. Changer le fuseau horaire

Nous créons ensuite le fichier change_timezone.c, processus qui modifie le fuseau horaire. Celui-ci se comporte de la même manière que change_alarm.c. On vérifie que l'id de la file de messages est passé en paramètre, puis on demande à l'utilisateur le nouveau fuseau. Enfin, on envoie ce nouveau fuseau par message à destination de l'horloge. Si celle-ci reçoit bien la donnée, elle va renvoyer un message de confirmation que change_timezone va récupérer.

4.3. Changer l'heure

Enfin, la dernière fonctionnalité est créée dans le fichier synchronise.c, qui va modifier l'horloge, soit manuellement, soit en se synchronisant au serveur.
Comme les deux autres processus, on vérifie que l'id de la file de message est passé en paramètre.
On envoie un message de demande de synchronisation à l'horloge et on attend la réception du message de confirmation de synchronisation.
On demande à l'utilisateur si l'heure doit être changée manuellement ou par synchronisation.
Si on modifie l'heure manuellement, la nouvelle heure est simplement demandée à l'utilisateur.
Si on veut se synchroniser au serveur, on demande l'id de la file de messages du serveur à l'utilisateur.
Grâce à cet id, on envoie un message au serveur pour demander l'heure. Celui-ci va alors renvoyer un message contenant la nouvelle heure.
Le message est alors récupéré ainsi que la nouvelle heure qu'il contient.
Cette nouvelle heure est ensuite envoyée à l'horloge via sa file de message propre.
Pour finir, comme dans les deux précédents processus, on attend le message de confirmation de l'horloge avant de quitter.

Note : nous avons décidé de mettre en place deux files de messages pour bien différencier les deux différentes strates lors de la synchronisation : un travail en interne (horloge principale – processus de synchronisation) et un travail en externe (processus de synchronisation – serveur).

4.4. Le menu

Ces 3 processus de modifications peuvent être lancés individuellement ou via un processus menu que nous créons maintenant.

On demande toujours l'id de la file de messages de l'horloge en argument. Ensuite, dans une boucle infinie, on demande quel processus l'utilisateur veut lancer. Suivant ce choix, on créée un processus fils grâce à la fonction fork() que l'on recouvre par le processus choisi (change_alarm, change_timezone ou synchronise) avec la fonction execl().
L'utilisateur a également le choix de modifier l'id de la file de message passée en argument (s'il s'est trompé lors de la saisie par exemple) ou simplement de quitter le menu.

A ce stade, nous disposons donc de l'horloge, du menu et des processus qui permettent de changer les données. Il ne nous reste plus qu'à créer le serveur qui aura pour tâche de donner l'heure lorsque l'horloge lui demande.

5. Le serveur

Au départ, on initialise toutes les données du serveur. On créée la file de message propre à celui-ci et on demande à l'utilisateur d'initialiser l'horloge via la fonction initClock().
Ensuite, dans une boucle infinie, on met à jour l'horloge du serveur à l'aide de la fonction updateClock() contenu dans clock_functions.c, on affiche l'heure grâce à displayTime() et on scrute la file de message pour savoir si une horloge quelconque veut se synchroniser (primitive msgrcv()). Si c'est le cas, on envoie un message contenant l'heure avec la fonction sendTime() dans la file du serveur.

6. Compléments

Afin de gerer proprement les files de messages IPC dans les processus tournant indéfiniments (while(TRUE)), nous avons décidés d'utiliser les signaux (primitive signal()) afin d'intercepter CTRL-C. Au lieu d'être imédiatement interrompu, le processus va passer par une fonction chargée de détruire la file de message qui lui est associée avant de se terminer (quitProperly()).

Vous trouverez en annexes les sources sous deux formes : les pdf des sources commentées avec la coloration syntaxique, ansi que les source commentées brutes pour une éventuelle compilation. Nous avons mis en place un petit script « compile.sh » qui se chargera de compiler le projet complet avec l'option -Wall.

```
#ifndef BOOL_H
#define BOOL_H

typedef
enum
{
    true = 1, false = 0, TRUE = true, FALSE = false
}
bool;

#endif
```

```c
#ifndef MQUEUE_TYPE_H
#define MQUEUE_TYPE_H

#include <time.h>

#define MSG_TIME_SIZE sizeof(time_t)    /* Taille des message du type : msg_time */
#define MSG_BYTE_SIZE sizeof(char)     /* Taille des message du type : msg_byte */
#define MSG_TIME_ZONE_SIZE sizeof(time_zone) /* Taille des message du type : msg_byte */

typedef int time_zone;

struct msg_time
{
    long  mtype;      /* Type de message ( > 0 ) */
    time_t  mtime;    /* Contenu du message */
};

struct msg_byte
{
    long  mtype;      /* Type de message ( > 0 ) */
    char  mbyte;    /* Contenu du message */
};

struct msg_time_zone
{
    long  mtype;      /* Type de message ( > 0 ) */
    time_zone  mtz;   /* Contenu du message */
};

typedef
enum
{
    /* Concerne la file de messages 1 */

    SND_TZONE = 1,   /* Message envoyé par le processus fuseau
        horaire : décalage */

    OK_TZONE,         /* Message répondu par l'horloge principale
        quand la modification est effectuée */

    SND_ALARM,        /* Message envoyé par le processus alarme :
        nombre de secondes depuis 00:00:00 */

    OK_ALARM,         /* Message répondu par l'horloge principale
        quand la modification est effectuée */

    REQ_SYNCHRO,      /* Message envoyé par le processus synchronisation
        demandant à l'horloge principale de se mettre en
        attente pour la synchronisation */

    AN_SYNCHRO,       /* Message répondu par l'horloge quand celle-ci
        est prete pour la synchronisation */

    SND_SYNCHRO,      /* Message envoyé par le processus synchronisation
        à l'horloge principale : la nouvelle heure */

    OK_SYNCHRO,       /* Message répondu par l'horloge principale quand
```

```
        la modification est effectuée */



    /* Concerne la file de messages 2 */

    REQ_TIME,           /* Message envoyé par le processus synchornisation
        au serveur externe pour lui demander l'heure */

    SND_TIME            /* Message envoyé par le serveur externe au
        processus synchronisation : la nouvelle heure */
}
mqueue_type;

#endif
```

clock_functions.h

```c
#ifndef CLOCK_FUNCTIONS_H
#define CLOCK_FUNCTIONS_H
#include "bool.h"
#include "mqueue_types.h"

/* Formate la valeur clock puis l'affiche en hh:mm:ss */
void displayTime(time_t clock, time_zone shift);

/* Retourne TRUE si clk a été mise à jour, FALSE sinon. La veleur raz à TRUE permet de
réinitialiser l'incrémentation si l'horloge a été mise à jour */
bool updateClock(time_t* clk, bool raz);

/* Donne à clk la valeur en secondes de hh:mm:ss depuis 00:00:00 */
void setTime(time_t* clk, int h, int m, int s);

/* Fonction modulo N, marche pour les valeurs de val négatives supperieurs à -N*/
int mod_N(int val, int N);

#endif
```

```c
#include "clock_functions.h"
#include <stdio.h>

void displayTime(time_t clock, time_zone shift)
{
    int hours;
    int minuts;
    int seconds;

    /* Formatage */
    hours = ((clock / 3600));
    minuts = ((clock - (hours * 3600)) / 60) % 60;
    seconds = ((clock - (hours * 3600) - (minuts * 60))) % 60;

    /* Affichage */
    printf("%02d : %02d : %02d\n", mod_N(hours + shift, 24), minuts, seconds);
}


/* Retourne TRUE si clk a été mise à jour, FALSE sinon */
bool updateClock(time_t* clk, bool raz)
{
    static time_t previous = 0;
    static time_t current;
    time_t delay;
    if(previous == 0 || raz == TRUE)
    {
        previous = time(NULL);
    }

    current = time(NULL);

    if(current == -1)
    {
        printf("Erreur *time*...\n");
        return FALSE;
    }

    if((delay = current - previous) > 0)
    {
        *clk = *clk + delay;
        previous = current;
        return TRUE;
    }
    else
    {
        return FALSE;
    }
}


void setTime(time_t* clk, int h, int m, int s)
{
    *clk = (mod_N(h, 24) * 3600) + (m * 60) + s;
}
```

```c
int mod_N(int val, int N)
{
    return (val + N) % N;
}
```

```c
#ifndef HORLOGE_PRINCIPALE_H
#define HORLOGE_PRINCIPALE_H
#include "bool.h"
#include "mqueue_types.h"



/* Retourne TRUE si l'horloge doit sonner, FALSE sinon */
bool ring(time_t clk, time_t alrm, time_zone sh);

/* Afficher "Beep" */
void beepBeep();

/* Interface avec l'utilisateur pour initialiser une horloge une alarme et un fuseau horaire */
void initAll(time_t* clk, time_t* alrm, time_zone* shift);

/* Verifie dans la file de messages id si le fuseau horaire doit etre modifié
et met sh à jour dans le cas ou il doit l'etre. La fonction renvoie TRUE si sh
a été mis à jour, FALSE sinon */
bool updateTimeZone(time_zone* sh, int id);

/* Verifie dans la file de messages id si l'alarme doit etre modifié
et met alrm à jour dans le cas ou elle doit l'etre. La fonction renvoie TRUE si
alrm a été mis à jour, FALSE sinon */
bool updateAlarm(time_t* alrm, int id);

/* Verifie dans la file de messages id si l'horloge doit etre synchronisée
et met clk à jour dans le cas ou elle doit l'etre. La fonction renvoie TRUE si
clk a été mis à jour, FALSE sinon */
bool synchronise(time_t* clk, int id);

#endif
```

```c
#include "clock_functions.h"
#include "horloge_principale.h"
#include <stdio.h>
#include <stdlib.h>
#include <sys/msg.h>
#include <sys/ipc.h>
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>

/* Utilisé pour détruire la file de message avant de quitter */
int* g_id = NULL;
void quitProperly(int v)
{
    msgctl(*g_id, IPC_RMID, NULL);
    exit(EXIT_SUCCESS);
}

int main(int argc, char** argv)
{
    /* Intercepter ctrl-c */
    signal(SIGINT, quitProperly);

    time_t clock = 0;    /* Horloge GMT */
    time_t alarm = 0;    /* Alarme GMT */
    time_zone shift = 0;   /* Stockage du décalage */

    /* Création de la file de message horloge principale <-> menu (fuseau, alarme, synchronisation) */
    key_t key = ftok(argv[0], 0);
    int mqueue1_id = msgget(key, IPC_CREAT | 0666);
    g_id = &mqueue1_id;

    /* Initialisation de l'horloge, de l'alarme et du fuseau (interface avec l'utilisateur) */
    initAll(&clock, &alarm, &shift);

    while(TRUE)
    {
        /* Scruter la file de message à la recherche de modifications à effectuer */
        updateTimeZone(&shift, mqueue1_id);
        updateAlarm(&alarm, mqueue1_id);
        synchronise(&clock, mqueue1_id);

        /* Mise à jour de l'horloge sans réinitialisation */
        if(updateClock(&clock, FALSE))
        {
            /* Si l'horloge à été mise à jour on affiche l'heure et on test si l'alarme doit s'activer */
            system("clear");
            displayTime(clock, shift);
            if(ring(clock, alarm, shift))
            {
                beepBeep();
                printf("Horloge principale - id de la file de messages : %d\n", mqueue1_id);
            }
            else
            {
                printf("\nHorloge principale - id de la file de messages : %d\n", mqueue1_id);
            }
```

```
            printf("Alarme reglee a : ");
            displayTime(alarm, 0);
        }

        usleep(100000); /* Endormir 1/10 seconde pour baisser la charge processeur */
    }

    return EXIT_SUCCESS;
}


/* Retourne TRUE si l'horloge doit sonner, FALSE sinon */
bool ring(time_t clk, time_t alrm, time_zone sh)
{
    return (
               (clk % 86400) >= mod_N((alrm % 86400) - (sh * 3600), 86400)
               &&
               (clk % 86400) <= mod_N(((alrm+60) % 86400) - (sh * 3600), 86400)
           );
}


void beepBeep()
{
    printf("Beep\n");
}


void initAll(time_t* clk, time_t* alrm, time_zone* shift)
{
    int h;  /* Heures horloge GMT */
    int m;  /* Minutes horloge GMT */
    int s;  /* Secondes horloge GMT */
    int ha; /* Heures alarme */
    int ma; /* Minutes alarme */
    int sa; /* Secondes alarme */
    int sh; /* Décalage pour l'affichage (fuseau horaire) */

    printf("\nVeuillez initialiser l'horloge : (heure fuseau alarme)\n");
    printf("Exemple : 15:08:32 +1 06:30:00\n\nInitilialiser a : ");
    scanf("%d:%d:%d %d %d:%d:%d", &h, &m, &s, &sh, &ha, &ma, &sa);

    *shift = sh;      /* Initialisation du décalage correspondant au fuseau specifié */
    setTime(clk, h-sh, m, s);    /* Initialisation de l'horloge à l'horaire GMT */
    setTime(alrm, ha, ma, sa);   /* Initialisation de l'alarme */
}


bool updateTimeZone(time_zone* sh, int id)
{
    struct msg_time_zone msg;
    if(msgrcv(id, &msg, MSG_TIME_ZONE_SIZE, SND_TZONE, IPC_NOWAIT) == -1)
    {
        return FALSE;
    }
    else
    {
```

```
        /* Préparation de la réponse */
        struct msg_byte answer;
        answer.mtype = OK_TZONE;
        answer.mbyte = 'O';

        /* Stockage du nouveau fuseau */
        *sh = msg.mtz;

        /* Retour signalant que la modification a été effectuée */
        if(msgsnd(id, &answer, MSG_BYTE_SIZE, 0) == -1)
        {
            printf("Erreur snd\n");
        }
        return TRUE;
    }
}


bool updateAlarm(time_t* alrm, int id)
{
    struct msg_time msg;
    if(msgrcv(id, &msg, MSG_TIME_SIZE, SND_ALARM, IPC_NOWAIT) == -1)
    {
        return FALSE;
    }
    else
    {
        /* Préparation de la réponse */
        struct msg_byte answer;
        answer.mtype = OK_ALARM;
        answer.mbyte = 'O';

        /* Stockage du nouveau fuseau */
        *alrm = msg.mtime;

        /* Retour signalant que la modification a été effectuée */
        if(msgsnd(id, &answer, MSG_BYTE_SIZE, 0) == -1)
        {
            printf("Erreur snd\n");
        }
        return TRUE;
    }
}


bool synchronise(time_t* clk, int id)
{
    struct msg_byte req;
    if(msgrcv(id, &req, MSG_BYTE_SIZE, REQ_SYNCHRO, IPC_NOWAIT) == -1)
    {
        return FALSE;
    }
    else
    {
        /* Préparation de la réponse */
        struct msg_byte answer1;
        answer1.mtype = AN_SYNCHRO;
```

```
        answer1.mbyte = 'O';

        /* Retour signalant que l'horloge est prete pour se synchroniser */
        if(msgsnd(id, &answer1, MSG_BYTE_SIZE, 0) == -1)
        {
            printf("Erreur snd\n");
        }

        /* Mise en attente de la nouvelle heure (pas de IPC_NOWAIT pour se bloquer en attente) */
        system("clear");
        printf("En attente de message de synchronisation dans la file %d.\n", id);
        struct msg_time msg;
        if(msgrcv(id, &msg, MSG_TIME_SIZE, SND_SYNCHRO, 0) == -1)
        {
            printf("Erreur lors de la synchronisation.\n");
            return FALSE;
        }
        else
        {
            /* Préparation de la réponse */
            struct msg_byte answer2;
            answer2.mtype = OK_SYNCHRO;
            answer2.mbyte = 'O';

            /* Stockage de la nouvelle heure et mise à jour de l'horloge avec réinitialisation */
            *clk = msg.mtime;
            updateClock(clk, TRUE);

            /* Retour signalant que la modification a été effectuée */
            if(msgsnd(id, &answer2, MSG_BYTE_SIZE, 0) == -1)
            {
                printf("Erreur snd\n");
            }

            printf("Horloge synchronisée.\n");
            return TRUE;
        }
    }
}
```

server.h

```
#ifndef SERVER_H
#define SERVER_H

#include "bool.h"
#include "mqueue_types.h"
#include "time.h"

void sendTime(int id, time_t* clk);
bool lookForRequest(int id);
void initClock(time_t* clk);

#endif
```

```c
#include "server.h"
#include <sys/msg.h>
#include <sys/ipc.h>
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "clock_functions.h"
#include <signal.h>

/* Utilisé pour détruire la file de message */
int* g_id = NULL;
void quitProperly(int v)
{
    msgctl(*g_id, IPC_RMID, NULL);
    exit(EXIT_SUCCESS);
}

int main(int argc, char** argv)
{
    /* Intercepter ctrl-c */
    signal(SIGINT, quitProperly);

    time_t clock = 0; /* Horloge GMT */
    bool snd = FALSE;

    /* Création de la file de message serveur <-> menu (synchronisation) */
    key_t key = ftok(argv[0], 0);
    int mqueue2_id = msgget(key, IPC_CREAT | 0666);
    g_id = &mqueue2_id;

    /* Initialisation de l'horloge (interface utilisateur) */
    initClock(&clock);

    while(TRUE)
    {
        /* Scruter la file de message à la recherche d'informations à envoyer */
        snd = lookForRequest(mqueue2_id);

        /* Mise à jour de l'horloge sans réinitialisation */
        if(updateClock(&clock, FALSE))
        {
            /* Si l'horloge à été mise à jour on affiche l'heure GMT */
            system("clear");
            displayTime(clock, 0);
            printf("\nServeur - id de la file de messages : %d\n", mqueue2_id);
        }

        if(snd)
            sendTime(mqueue2_id, &clock);

        usleep(100000); /* Endormir 1/10 seconde pour baisser la charge processeur */
    }


    return EXIT_SUCCESS;
}
```

```c
void sendTime(int id, time_t* clk)
{
    /* Préparation de la réponse */
    struct msg_time answer;
    answer.mtype = SND_TIME;
    answer.mtime = *clk;

    /* Retour signalant que l'horloge est prete pour se synchroniser */
    if(msgsnd(id, &answer, MSG_TIME_SIZE, 0) == -1)
    {
        printf("Erreur snd\n");
    }
}

bool lookForRequest(int id)
{
    struct msg_byte req;
    if(msgrcv(id, &req, MSG_BYTE_SIZE, REQ_TIME, IPC_NOWAIT) == -1)
    {
        return FALSE;
    }
    else
    {
        return TRUE;
    }
}

void initClock(time_t* clk)
{
    int h;   /* Heures horloge GMT */
    int m;   /* Minutes horloge GMT */
    int s;   /* Secondes horloge GMT */

    printf("\nVeuillez initialiser l'horloge GMT :\n");
    printf("Exemple : 15:08:32\n\nInitilialiser a : ");
    scanf("%d:%d:%d", &h, &m, &s);

    setTime(clk, h, m, s);    /* Initialisation de l'horloge */
}
```

change_timezone.c

```c
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include "mqueue_types.h"

int main(int argc, char** argv)
{
    int id;
    time_zone shift;
    struct msg_time_zone tzone;
    struct msg_byte answer;

    /* Vérification du passage en argument de l'id de la file de message*/
    if(argc < 2)
    {
        printf("Mauvaise utilisation : %s <id file>\n", argv[0]);
        exit(EXIT_FAILURE);
    }
    id = atoi(argv[1]);

    /* Demande du nouveau fuseau horaire a l'utilisateur*/
    printf("\nChanger le fuseau horaire\n");
    printf("Exemple : +2\n\nNouveau fuseau : ");
    scanf("%d",&shift);

    /* Préparation du message : le nouveau fuseau*/
    tzone.mtype = SND_TZONE;
    tzone.mtz = shift;

    /* Envoi du message à l'horloge principale*/
    if(msgsnd(id, &tzone, MSG_TIME_ZONE_SIZE, 0) == -1)
    {
        exit(EXIT_FAILURE);
    }

    printf("\nAttente du règlage du fuseau...\n");

    /* Attente de réception du message confirmant le changement de fuseau par l'horloge*/
    if(msgrcv(id, &answer, MSG_BYTE_SIZE, OK_TZONE, 0) == -1)
    {
        exit(EXIT_FAILURE);
    }

    printf("\nFuseau programmé à %d\n", shift);

    return EXIT_SUCCESS;
}
```

```c
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include "mqueue_types.h"
#include "clock_functions.h"

int main(int argc, char** argv)
{
    int id;
    int h, m, s;
    time_t clk;
    struct msg_time alarm;
    struct msg_byte answer;

    /* Vérification du passage en argument de l'id de la file de message*/
    if(argc < 2)
    {
        printf("Mauvaise utilisation : %s <id file>\n", argv[0]);
        exit(EXIT_FAILURE);
    }
    id = atoi(argv[1]);

    /* Demande de la nouvelle alarme a l'utilisateur*/
    printf("\nChanger l'alarme\n");
    printf("Exemple : 15:08:32\n\nNouvelle alarme : ");
    scanf("%d:%d:%d",&h,&m,&s);

    setTime(&clk, h, m, s);

    /* Préparation du message : la nouvelle alarme*/
    alarm.mtype = SND_ALARM;
    alarm.mtime = clk;

    /* Envoi du message à l'horloge principale*/
    if(msgsnd(id, &alarm, MSG_TIME_SIZE, 0) == -1)
    {
        exit(EXIT_FAILURE);
    }

    printf("\nAttente du règlage de l'alarme...\n");

    /* Attente de réception du message confirmant le changement d'alarme par l'horloge*/
    if(msgrcv(id, &answer, MSG_BYTE_SIZE, OK_ALARM, 0) == -1)
    {
        exit(EXIT_FAILURE);
    }

    printf("\nAlarme programmée à %d:%d:%d\n", h, m, s);

    return EXIT_SUCCESS;
}
```

```c
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include "mqueue_types.h"
#include "clock_functions.h"

int main(int argc, char** argv)
{
    int idC, idS;
    int choix;
    int h, m, s;
    time_t newclock;
    struct msg_time clk;
    struct msg_time serv;
    struct msg_byte reqC;
    struct msg_byte reqS;
    struct msg_byte answer1;
    struct msg_byte answer2;

    /* Vérification du passage en argument de l'id de la file de message de l'horloge*/
    if(argc < 2)
    {
        printf("Mauvaise utilisation : %s <id file horloge>\n", argv[0]);
        exit(EXIT_FAILURE);
    }
    idC = atoi(argv[1]);

    /* Préparation du message de demande de synchronisation avec l'horloge*/
    reqC.mtype = REQ_SYNCHRO;
    reqC.mbyte = '0';

    /* Envoi du message à l'horloge*/
    if(msgsnd(idC, &reqC, MSG_BYTE_SIZE, 0) == -1)
    {
        exit(EXIT_FAILURE);
    }
    printf("\nSynchronisation avec l'horloge...\n");

    /* Attente de réception du message confirmant la synchronisation avec l'horloge*/
    if(msgrcv(idC, &answer1, MSG_BYTE_SIZE, AN_SYNCHRO, 0) == -1)
    {
        exit(EXIT_FAILURE);
    }

    /* Demande à l'utilisateur si l'heure doit  être changée manuellement ou automatiquement*/
    do
    {
        printf("\n1 : Changer l'horloge manuellement\n");
        printf("2 : Synchroniser l'horloge avec le serveur\n");
        scanf("%d",&choix);
    }
    while(choix != 1 && choix != 2);
```

```
    switch(choix)
    {
        /* L'horloge est changée manuellement, l'utilisateur entre la nouvelle heure*/
        case 1:
        printf("\nEntrez l'heure\n");
        printf("Exemple : 15:08:32\n\nNouvelle heure : ");
        scanf("%d:%d:%d", &h, &m, &s);

        setTime(&newclock, h, m, s);
        break;

        /* L'horloge est changée par synchronisation aec le serveur*/
        case 2:
        printf("\nEntrez l'id de la file de message du serveur\n");
        scanf("%d", &idS);

        /* Préparation du message de demande de synchronisation avec le serveur*/
        reqS.mtype = REQ_TIME;
        reqS.mbyte = '0';

        /* Envoi du message au serveur*/
        if(msgsnd(idS, &reqS, MSG_BYTE_SIZE, 0) == -1)
        {
            exit(EXIT_FAILURE);
        }

        printf("\nSynchronisation avec le serveur...\n");

        /* Attente de la réponse du serveur : la nouvelle heure*/
        if(msgrcv(idS, &serv, MSG_TIME_SIZE, SND_TIME, 0) == -1)
        {
            exit(EXIT_FAILURE);
        }

        newclock = serv.mtime;
        break;
        default:
        printf("\nErreur\n");
        break;
    }
    /* Préparation du message d'envoi de la nouvelle heure à l'horloge*/
    clk.mtype = SND_SYNCHRO;
    clk.mtime = newclock;

    /* Envoi du message à l'horloge : la nouvelle heure*/
    if(msgsnd(idC, &clk, MSG_TIME_SIZE, 0) == -1)
    {
        exit(EXIT_FAILURE);
    }

    printf("\nAttente du règlage de l'horloge...\n");

    /* Attente de la réception du message de confirmation de changement d'heure par l'horloge*/
    if(msgrcv(idC, &answer2, MSG_BYTE_SIZE, OK_SYNCHRO, 0) == -1)
    {
        exit(EXIT_FAILURE);
    }
```

```
    printf("\nhorloge synchronisée\n");

    return EXIT_SUCCESS;
}
```

menu.h

```
#ifndef MENU_H
#define MENU_H

void printMenu(int id);
void clean();

#endif
```

```c
#include <stdio.h>
#include <stdlib.h>
#include "menu.h"
#include "bool.h"
#include <unistd.h>
#include <wait.h>

int main(int argc, char** argv)
{
    bool quit = FALSE;
    int answer = 0;
    int id;

    /* vérification du passage en argment de l'id de la file de message*/
    if(argc < 2)
    {
        printf("Mauvaise utilisation : %s <id file>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    id = atoi(argv[1]);

    while(!quit)
    {
        system("clear");
        printMenu(id);
        scanf("%d", &answer);

        switch(answer)
        {
            case 1:
            /* choix de modification de l'alarme*/
            if(fork()==0)
            {
                execl("chalarm", "chalarm", argv[1], NULL);
                printf("Erreur exec\n");
                exit(EXIT_FAILURE);
            }
            else
                wait(NULL);
            break;

            /* choix de modification du fuseau*/
            case 2:
            if(fork()==0)
            {
                execl("chtz", "chtz", argv[1], NULL);
                printf("Erreur exec\n");
                exit(EXIT_FAILURE);
            }
            else
                wait(NULL);
            break;

            /* choix de modification de l'heure*/
            case 3:
            if(fork()==0)
```

```
            {
                execl("sync", "sync", argv[1], NULL);
                printf("Erreur exec\n");
                exit(EXIT_FAILURE);
            }
            else
                wait(NULL);
            break;

            /* choix de quitter*/
            case 4:
            quit = TRUE;
            break;

            /* affichage de l'id de la file de message de l'horloge*/
            case 5:
            {
                system("clear");
                printf("\n\n\tNouvelle id de l'horloge : ");
                scanf("%d", &id);
                clean();
            }
                    break;

                    default:
            printf("\n\tErreur de saisie...\n");
            sleep(1);
            clean();
        }
    }
    return EXIT_SUCCESS;
}


void printMenu(int id)
{
    printf("\t\tMENU\n\n\t1 - Modifier l'alarme\n\t2 - Modifier le fuseau horaire\n\t3 -
Synchroniser l'horloge\n\t4 - Quitter\n\n\t5 - Modifier l'id de la file de message de
l'horloge (%d)\n\n\n\t", id);
}

void clean()
{
    char c;
    while((c = getc(stdin)) != EOF && c != '\n')
        ;
}
```

compile.sh

```
#/bin/sh

gcc -Wall menu.c -o "menu" &&
gcc -Wall horloge_principale.c clock_functions.c -o "hp" &&
gcc -Wall change_alarm.c clock_functions.c -o "chalarm" &&
gcc -Wall change_timezone.c -o "chtz" &&
gcc -Wall synchronise.c clock_functions.c -o "sync" &&
gcc -Wall server.c clock_functions.c -o "server"
```