GAVALDA Jason
SCHOENBERGER Maxime

TP2

Objectif du TP :

Le but du TP est d'implémenter l'algorithme d'exclusion mutuelle de Lamport pour gérer la section critique entre trois sites qui échangent des messages. Il faudra pour cela implémenter tout ce qui est nécessaire à l'idée derrière cet algorithme, à savoir la gestion des files d'attentes des messages, les estampilles/horloges logiques, la structure des dits messages et enfin, implémenter l'algorithme lui-même.

La distribution des messages par la file d'attente doit respecter l'ordonnancement des estampilles des dits messages en suivant la structure FIFO : lorsque un message est stocké dans celle-ci, il est placé selon son estampille. Les messages les plus anciens sont placés en début de file (et donc en sortent en premier) tandis que les plus récents sont placés en fin de file.

L'algorithme de Lamport a lui pour but de réguler l'entrée en Section Critique (SC) de chaque site :

Si un site Si veut entrer en SC :

- Il place sa requête dans une file FILEi et envoie un message *requête* à tous les autres sites.
- Le site Si entre en SC quand il reçoit la *réponse* du site Sj ET quand sa requête est en tête de FILEi.

Quand le site SJ reçoit *requête* de Si :

- Il met à jour son Horloge Logique HLj tel que HLj = max(HLj, ELm) + 1, où ELm est l'estampille du message reçu.
- place *requête* dans FILEj et envoie sa réponse *réponse* à Si.

Quand Si sort de SC il envoie un message *libération* à tous les autres sites et retire sa *requête* de FILEi.

Chaque site Sj qui reçoit un message *libération* de Si met à jour son horloge comme indiqué précédemment et enlève *requête* de FILEj.

GAVALDA Jason
SCHOENBERGER Maxime

TP2

Structure du programme :

Traitement des messages :

La structure des messages que nous échangeons entre les sites est la suivante :

```
struct msg{
    bool flag;
    char Type[20];
    int ELm;
    int Site;
};
```

- La chaîne de caractères *Type* permet de distinguer les types de messages (Requêtes, Réponses, Libération).
- L'entier *ELm* est l'estampille du message, c'est-à-dire la valeur de l'horloge logique du site d'envoi au moment où le message a été envoyé.
- L'entier *Site* correspond au site de provenance du message.
- Le booléen *flag* permet de différencier si les messages des files ont été correctement initialisé avec des requêtes, les files étant initialisées avec des messages "faux".

La fonction suivante permet de remplir une chaîne de caractères buffer à partir d'un message, un espace séparant chaque élément du message :

```
void buildMessage(struct msg message, char * buffer){
    sprintf(buffer, "%s %d %d.", message.Type, message.ELm, message.Site);
}
```

La fonction suivante permet de créer une structure de message à partir d'une chaîne de caractères *readRequest* et d'une taille entière *size*.

```c
struct msg readMessage(char * readRequest, int size){
    int i = 0;
    int j = 0;
    char * buffer = (char*)malloc(size * sizeof(char));
    struct msg messageRequest;

    /*------------Récupération du Type-----------------*/
    while(readRequest[i] != ' '){
        buffer[i] = readRequest[i];
        i++;
    }
    strcpy(messageRequest.Type, buffer);
    /*-------------------------------------------------*/
    i++;
    j=0;
    /*------------Récupération de l'estampille---------*/
    while(readRequest[i] != ' '){
        buffer[j] = readRequest[i];
        i++;
        j++;
    }
    buffer[j] = '\0';
    messageRequest.ELm = atoi(buffer);
    /*-------------------------------------------------*/
    i++;
    j=0;
    /*------------Récupération du Site-----------------*/
    while(readRequest[i] != '.'){
        buffer[j] = readRequest[i];
        i++;
        j++;
    }
    buffer[j] = '\0';
    messageRequest.Site =atoi(buffer);
    /*-------------------------------------------------*/
    free(buffer);
    messageRequest.flag = true; //Passage du flag à true, pour valider le message.
    return messageRequest;
}
```

Comme les différentes caractéristiques de nos messages sont séparées par des espaces, nous pouvons exploiter cela avec des boucles while pour récupérer depuis la string en paramètre un à un des attributs d'un message. Ceci oblige à faire attention à bien respecter la syntaxe "Requete 2 3." lorsque l'on envoie un message.

Il ne faut pas oublier à la fin de passer le flag à true pour rendre le message valide dans les files.

La fonction suivante permet l'envoi de messages vers un site donné en paramètre.

```c
void sendMessage(struct msg message, int PortBase, int Site){
    char buffer[40];
    buildMessage(message, buffer); //Construction de la chaine de caractères à partir de message.
    printf("Message envoye : %s to site %d\n", buffer, Site);
    SendSync("localhost", PortBase, buffer); //Envoi d'un message de synchro au Site.
}
```

La fonction suivante (utilisée dans SendMessage), permet physiquement l'envoi du message par l'utilisation de sockets. Elle était déjà fournie, à la seule différence que SendSync requiert un string personnalisé pour pouvoir être utilisé.

```c
/*Envoi d'un msg de synchro à la machine Site/Port*/
void SendSync(char *Site, int Port, char * stringMsg) {
  struct hostent* hp;
  int s_emis;
  int longtxt;
  struct sockaddr_in sock_add_emis;
  int size_sock;

  /*-------------Création socket-----------------------------------------*/
  if ( (s_emis=socket(AF_INET, SOCK_STREAM,0))==-1) {
    perror("SendSync : Creation socket");
    exit(-1);
  }

  /*-------------Récupération des caractéristiques de l'hôte--------------*/
  hp = gethostbyname(Site);
  if (hp == NULL) {
    perror("SendSync: Gethostbyname");
    exit(-1);
  }

  size_sock=sizeof(struct sockaddr_in);
  sock_add_emis.sin_family = AF_INET;
  sock_add_emis.sin_port = htons(Port);
  memcpy(&sock_add_emis.sin_addr.s_addr, hp->h_addr, hp->h_length);

  /*------------Connexion à l'hôte----------------------------------------*/
  if (connect(s_emis, (struct sockaddr*) &sock_add_emis,size_sock)==-1) {
    perror("SendSync : Connect");
    exit(-1);
  }

  longtxt =strlen(stringMsg);
  /*Emission d'un message de synchro*/
  if(write(s_emis,stringMsg,longtxt) == -1)
  {
    perror("write error");
    exit(-1);
  }
  close (s_emis);
}
```

La fonction suivante envoie un message à tous les autres sites via la fonction sendMessage.

TP2

```
void sendMessageToAll(struct msg newRequest, int PortZero, int NbSites, int id){
        for(int l = 0; l < NbSites; l++){
            if(l != id)
                sendMessage(newRequest, PortZero + l, id);
        }
}
```

La fonction suivante était déjà fournie et sert justement à l'attente d'un message de synchronisation.

```
/*Attente bloquante d'un msg de synchro sur la socket donnée*/
void WaitSync(int s_ecoute) {
  char texte[40];
  int l;
  int s_service;
  struct sockaddr_in sock_add_dist;
  unsigned int size_sock;
  size_sock=sizeof(struct sockaddr_in);
  printf("WaitSync : ");fflush(0);
  s_service=accept(s_ecoute,(struct sockaddr*) &sock_add_dist, &size_sock);
  l=read(s_service,texte,39);
  texte[l] ='\0';
  printf("%s\n",texte); fflush(0);
  close (s_service);
}
```

Implémentation de la file de message:

Nous avons vu comment notre structure de message fonctionnait, nous allons maintenant voir comment nous avons implémenté la file.

```
struct msg file[NSites];
initFile(file, NSites);
```

Notre file est tout simplement stockée en mémoire dans un tableau de taille NSites, NSites bornant la possibilité de messages stockés par l'algorithme de Lamport. Seulement stockée ainsi, les flags des messages non initialisés n'ayant pas un flag false, il nous faut initialiser notre file de struct msg, ayant un flag initialisé a false. C'est le but de initFile.

TP2

```
void initFile(struct msg* tabRequests, int size){
    struct msg falseRequest = (struct msg){.flag = false, .Type = "falseRequest", .ELm = -1, .Site = -1};
    for(int i = 0; i < size; i++){
        tabRequests[i] = falseRequest;
    }
}
```

Ensuite il y a 3 opérations que l'on peut faire sur notre file:

-RemoveFirstMsg(), qui enlève le tab[0] de la file, et la réorganise en fonction.

```
void removeFirstMsg(struct msg * tabRequests, int NbSites){
    int i;
    for(i = 0; i < NbSites; i++)
    {
        if(tabRequests[i].flag){
            tabRequests[i] = tabRequests[i + 1]; // on décale tout les messages sur la gauche
        }
    }
    struct msg falseRequest = (struct msg){.flag = false, .Type = "falseRequest", .ELm = -1, .Site = -1};
    tabRequests[NbSites - 1] = falseRequest; // Le dernier message à droite est remplacer par une fausse requete
}
```

-PeekSite(), qui renvoie le site du premier message de la file. (utile pour rentrer en section critique).

```
int peekSite(struct msg * tabRequests){
    if(tabRequests[0].flag){
        return tabRequests[0].Site;
    }
    else return -1;
}
```

- Insert(), qui s'occupe d'insérer à la bonne place le msg dans la file.

```
void insert(struct msg * tabRequests, int NbSites, struct msg * newRequest){
    //printFile(tabRequests, NbSites);
    //printf ("insert req = %s, %d, %d\n", newRequest->Type, newRequest->ELm, newRequest->Site);
    for(int i = NbSites - 1; i>=0; i--) // boucle decroissante
    {
        if(tabRequests[i].flag){ //Si la requete existe
            if(tabRequests[i].ELm == newRequest->ELm && tabRequests[i].Site < newRequest->Site){ //Si les horloges sont identiques, on trie par le site
                tabRequests[i+1] = *newRequest; // le site de la requete est plus grande, on peut donc inserer la requete ici
                //printFile(tabRequests, NbSites);
                return;
            }
            else if(tabRequests[i].ELm < newRequest->ELm){ //Si l'horloge de la requete inserée est plus grande
                tabRequests[i+1] = *newRequest; // l'horloge de la requete est plus grande, on peut donc inserer la requete ici
                //printFile(tabRequests, NbSites);
                return;
            }
            else
                tabRequests[i+1] = tabRequests[i]; //dans tout les autres cas, on décale les valeurs sur la droite
        }
    }
    tabRequests[0] = *newRequest;
    //printFile(tabRequests, NbSites);
}
```

TP2

En insérant à chaque fois notre requête à la bonne place, notre file est tout le temps correctement triée, et il n'y a pas besoin de s'assurer qu'elle le soit.

Il y a aussi la fonction printFile(), qui permet d'afficher la file, pratique pour déboguer.

Programme principal / Algorithme de Lamport :

Ceci était déjà fourni et sert à initialiser les ports ainsi que les sockets d'écoute.

```
int main (int argc, char* argv[]) {
  struct sockaddr_in sock_add, sock_add_dist;
  unsigned int size_sock;
  int s_ecoute, s_service;
  char texte[40];
  int i,l;
  float t;
  srand( time( NULL ) );

  int PortBase=-1; /*Numero du port de la socket a` creer*/
  int NSites=-1; /*Nb total de sites*/


  if (argc!=4) {
    printf("Il faut donner le numero de port du site et le nombre total de sites");
    exit(-1);
  }

  /*----Nombre de sites (adresses de machines)---- */
  //NSites=argc-2;
  NSites=atoi(argv[3]); //-2;
  struct msg file[NSites];
  initFile(file, NSites);


  /*CREATION&BINDING DE LA SOCKET DE CE SITE*/
  int PortZero=atoi(argv[1]);
  PortBase=atoi(argv[2]) ;//+GetSitePos(NSites, argv);
  printf("Numero de port de ce site %d\n",PortBase);

  sock_add.sin_family = AF_INET;
  sock_add.sin_addr.s_addr= htons(INADDR_ANY);
  sock_add.sin_port = htons(PortBase);

  if ( (s_ecoute=socket(AF_INET, SOCK_STREAM,0))==-1) {
    perror("Creation socket");
    exit(-1);
  }

  if ( bind(s_ecoute,(struct sockaddr*) &sock_add, \
        sizeof(struct sockaddr_in))==-1) {
    perror("Bind socket");
    exit(-1);
  }
```

```
listen(s_ecoute,30);
/*----La socket est maintenant créée, bindée et listen----*/

if (PortZero == PortBase){
  /*Le site 0 attend une connexion de chaque site : */
  for(i=0;i<NSites-1;i++) {
        WaitSync(s_ecoute);
  }

  printf("Site 0 : toutes les synchros des autres sites recues \n");fflush(0);
  /*et envoie un msg a chaque autre site pour les synchroniser */
  for(i=0;i<NSites-1;i++)
    SendSync("localhost", atoi(argv[1])+i+1, "** SYNCHRO **");
  } else {
    /* Chaque autre site envoie un message au site0
   (1er  dans la liste) pour dire qu'il est lance'*/
    SendSync("localhost", atoi(argv[1]), "** SYNCHRO **");
    /*et attend un message du Site 0 envoye' quand tous seront lance's*/
    printf("Wait Synchro du Site 0\n");fflush(0);
    WaitSync(s_ecoute);
    printf("Synchro recue de  Site 0\n");fflush(0);
}


/* Passage en mode non bloquant du accept pour tous*/
/*-------------------------------------*/
fcntl(s_ecoute,F_SETFL,O_NONBLOCK);
size_sock=sizeof(struct sockaddr_in);

int id = GetSitePos(NSites, argv);
int nbSitesAnswered = 0;
bool waitAnswer;
bool requestInFile;
int horloge = 1;
```

GAVALDA Jason
SCHOENBERGER Maxime

TP2

Voici la boucle principale du programme.
Les lignes sont commentées pour en faciliter la compréhension.

Ici, on s'occupe de la lecture et du traitement des messages reçus par les autres terminaux.

```
  while(1) {
    /* On commence par tester l'arrivee d'un message */
    s_service=accept(s_ecoute,(struct sockaddr*) &sock_add_dist,&size_sock);
    if(s_service>0) {

    /*Extraction et affichage du message */
        l=read(s_service,texte,39);

        printf("Message recu : %s\n",texte); fflush(0);
        struct msg newRequest = readMessage(texte, 39); // Création du message.
        horloge = max(horloge, newRequest.ELm) + 1; // Mise à jour de l'horloge du site.
        if(strcmp(newRequest.Type, "Requete") == 0){
            insert(file, NSites, &newRequest); //Si le message est une requête, il est placé
dans la file d'attente.

            struct msg answerRequest = (struct msg){.flag = true, .Type = "Reponse", .ELm =
horloge, .Site = id }; // Création du message de réponse.

            sendMessage(answerRequest, PortZero + newRequest.Site, id); // Envoi de la réponse.
        }
        else if(strcmp(newRequest.Type, "Liberation") == 0){
            if(!(newRequest.Site == file[0].Site)){
                //removeFirstMsg(file, NSites);
                printf("Error liberation and first message sites are different %d != %d\n",
file[0].Site, newRequest.Site);
            }
            else{
                removeFirstMsg(file, NSites); // On retire le premier message de la file.
            }


        }
        else if(strcmp(newRequest.Type, "Reponse") == 0){
            nbSitesAnswered ++;
        }
        else
            printf("Requete pas comprise : %s, %d, %d\n", newRequest.Type, newRequest.ELm,
newRequest.Site);

      close (s_service);
    }
```

Ici, l'on s'occupe d'envoyer de façon aléatoire des requêtes aux autres sites, et de rentrer en section critique si l'on a eu une réponse de chacun d'entre eux.

```
    /* Petite boucle d'attente : c'est ici que l'on peut faire des choses*/
    for(l=0;l<1000000;l++) {
      t=t*3;
      t=t/3;
    }

    float frand = (float)rand()/(float)RAND_MAX; // Création d'un flottant aléatoire.
    if(frand > 0.6 && frand < 0.7){
        horloge++; // Si la condition est celle-ci, on incrémente l'horloge du site.

    }
    else if(frand > 0.7 && frand < 0.8 && !requestInFile){
        struct msg newRequest = (struct msg){.flag = true, .Type = "Requete", .ELm =
horloge, .Site = id }; // Si la condition est respectée, on créé une requête.
        insert(file, NSites, &newRequest); // On insère cette requête dans la file d'attente.

        sendMessageToAll(newRequest, PortZero, NSites, id); // On l'envoie à tous les sites.
        waitAnswer = true;
        requestInFile = true; // On signifie que la requête est placée et qu'on attend une
réponse.
    }


    if(nbSitesAnswered == NSites - 1) // on verifie si on a toutes les reponses.
        waitAnswer = false;


    if(requestInFile && !waitAnswer && peekSite(file) == id){
        //Section Critique
        printf("Section critique : %d\n", horloge);
        removeFirstMsg(file, NSites); // On retire le message en tête de file.
        struct msg newRequest = (struct msg){.flag = true, .Type = "Liberation", .ELm =
horloge, .Site = id }; // On créé un message de Libération de la Section Critique.
        sendMessageToAll(newRequest, PortZero, NSites, id); // On l'envoie à tous les sites.
        requestInFile = false;
        nbSitesAnswered = 0;
    }
    else if(requestInFile && !waitAnswer){
        //printf("wrong id, cant critic");
    }
    printf(".");
    fflush(0); /* pour montrer que le serveur est actif*/
  }


  close (s_ecoute);
  return 0;
```

GAVALDA Jason
SCHOENBERGER Maxime

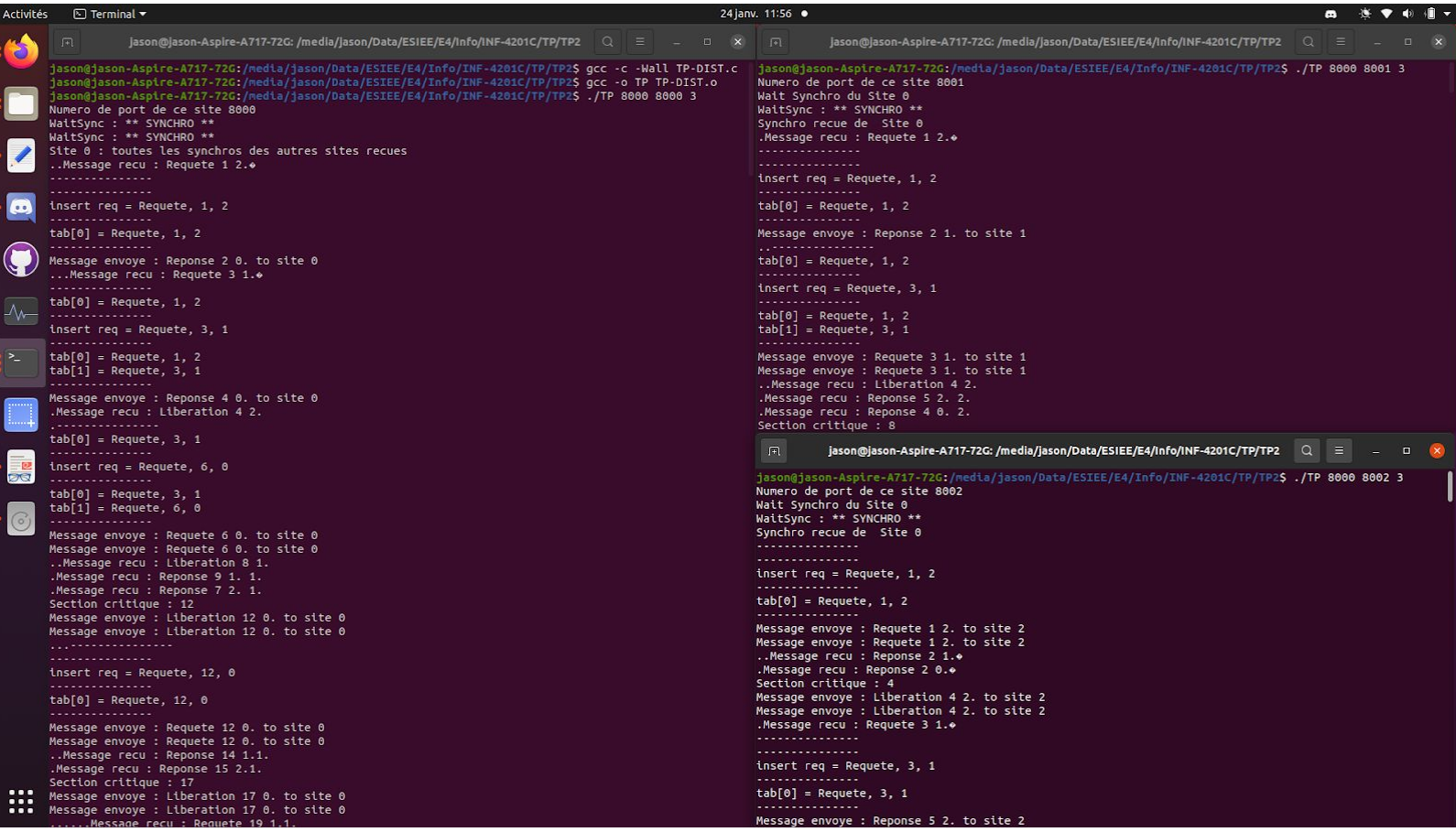
TP2

Exécution :

Pour compiler le programme on peut simplement utiliser: gcc -Wall TP-DIST.c -o TP
et comme on peut le voir, nous n'avons pas d'erreur à la compilation :

```
jason@jason-Aspire-A717-72G:/media/jason/Data/ESIEE/E4/Info/INF-4201C/TP/TP2$ gcc -c -Wall TP-DIST.c
jason@jason-Aspire-A717-72G:/media/jason/Data/ESIEE/E4/Info/INF-4201C/TP/TP2$ gcc -o TP TP-DIST.o
jason@jason-Aspire-A717-72G:/media/jason/Data/ESIEE/E4/Info/INF-4201C/TP/TP2$
```

Nous exécutons le programme dans 3 fenêtres de terminal distinctes :



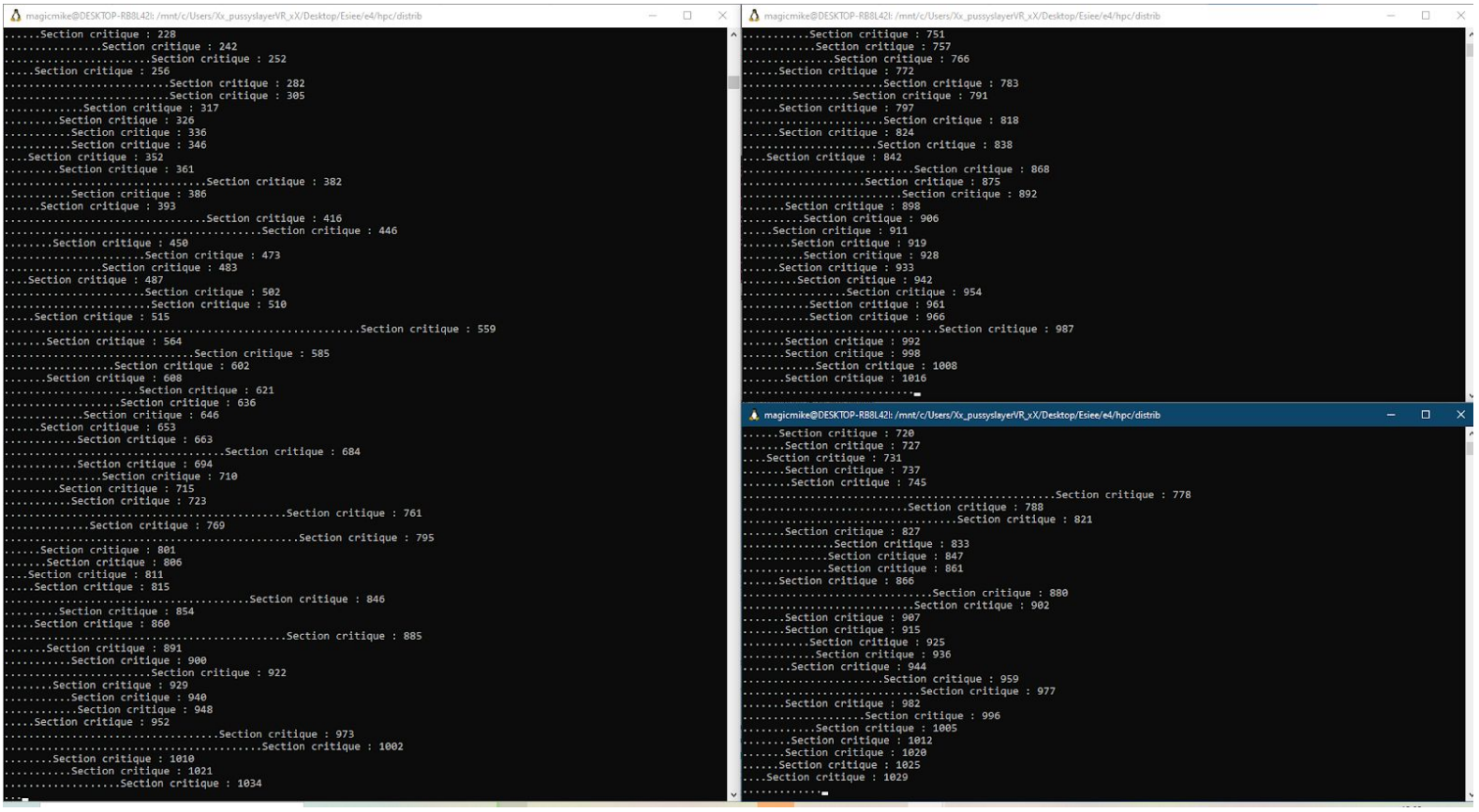
On peut constater que le programme entre bien en section critique comme au bon moment, chaque requête envoyée amène la réception de deux réponses.

On remarque que l'entrée en section critique entraîne bien, une fois terminée, la création et l'envoi d'un message de libération pour chaque site.

GAVALDA Jason
SCHOENBERGER Maxime

TP2

Voici une autre exécution en enlevant les print de messages et en ne gardant que ceux des sections critiques :



Principales difficultés rencontrées :

Nous avons rencontré plusieurs difficultés lors de la réalisation de ce programme.

- Implémentation complète mais imparfaite d'une liste double chaînée pour la file de msg, avant de se rendre compte que la borne max NSites nous permettait tout simplement de faire un tableau.

- Oublie de remettre la variable Nb Sites Answered à 0, ce qui causait les processus de ne plus s'envoyer de messages après quelque temps.

- Mise à jour de l'horloge pas automatique.

- Fonction insert() ne gérait pas l'insertion des requêtes en fonction de leur sites

- readMessage() (qui prend un char * en entrée et renvoie un struct msg) utilisant le même buffer pour les 3 lectures dans notre newRequest, pouvait parfois donner d'importantes valeurs aux site (entre 10 et 39). En effet en utilisant le même buffer sur lequel on applique Atoi(), il y avait des valeurs résiduelles de l'horloge dans le buffer qui n'étaient pas effacées. Plutôt que d'utiliser 3 buffers différents, nous avons tout simplement ajouté le caractère '\O' à la fin du buffer après chaque mot lu par ce 'parser'.

- Des comportements différents en fonction des machines sur lesquelles étaient exécutés le code (Avec le même compilateur, sur plusieurs essaies). La seule différence est l'OS.

Cela peut sembler comme peu de problèmes, mais le programme étant mis dans un seul fichier.c, les petits problèmes simples peuvent rapidement être très difficiles à détecter. Particulièrement le insert(). Or ces problèmes ne sont pas du tout difficiles à résoudre. Cela nous amène à tirer deux leçons de codes: - Bien réfléchir à s'il n'existe pas une façon plus simple de résoudre le problème.

- Mieux compartimenter le code, utiliser plusieurs fichiers notamment lorsque l'on créer beaucoup de fonctions utilitaires. Cela permet aussi de les tester indépendamment du reste du programme, et de tester les modifications rapidement. Ce au final ce qui à été fait pour déboguer plus rapidement.

Déclaration anti-plagiat :

Le code ne contient pas de segments pris sur internet ou chez des camarades.

Par rapport à l'unité :

Pas de difficultés avec l'unité., elle était bien construite, les cours sont plus dur a suivre en distanciel mais les projets permettent de compenser et facilitent énormément l'apprentissage.

En revanche la décision de mettre le partiel en présentiel est une décision qui se comprend, mais en période de pandémie, quand pour certains nous sommes confinés depuis longtemps avec les mêmes personnes, cela nous semble dangereux alors qu'il existe des alternatives ou certes la triche est possible, mais qui permet la sécurité des élèves et de leurs familles.