Lycée La Fayette
Champagne-sur-Seine • Fontaineroux

Informatique et Réseaux

Rapport de Travaux Pratiques
BTS SN-IR

MTR - TP Web MT

Flavian LAXENAIRE
12 octobre 2021

Forme / 20 [coef. 1] →		Fond / 20 [coef. 2] →		Note / 20	
Qualité du rapport		Méthodologie			
Expression écrite		Respect du Cahier des Charges			
Pertinence de la rédaction		Qualité technique			
Respect des standards de codage		État d'avancement		Malus	

Sommaire

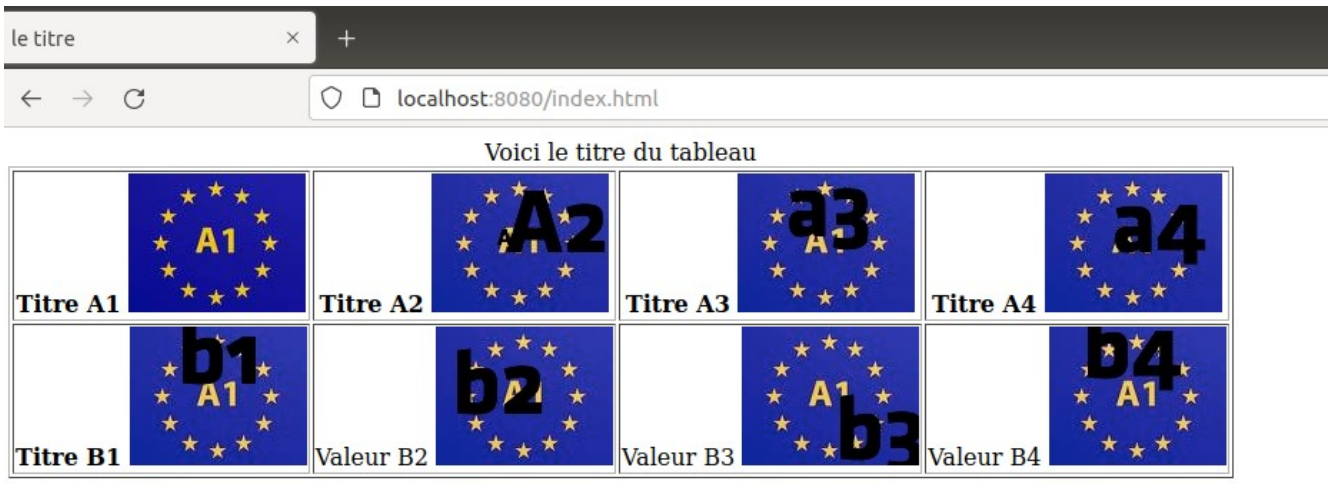
1. Etape0 :V0.1, Mise en Œuvre du serveur

Ce programme est le résultat de la compilation de deux fichiers sources.

Lors de l'exécution, il prend en argument 1 le port d'écoute du serveur. Nous choisissons par défaut le port 8080.

Les fichiers client sont stockés dans le dossier docroot, on les ouvre via un fopen, on utilise aussi la fonction stat.

L'affichage prend du temps, c'est dû à la ligne : usleep(200000) ;



2. Etape1 :V0.2, Fermeture propre du serveur

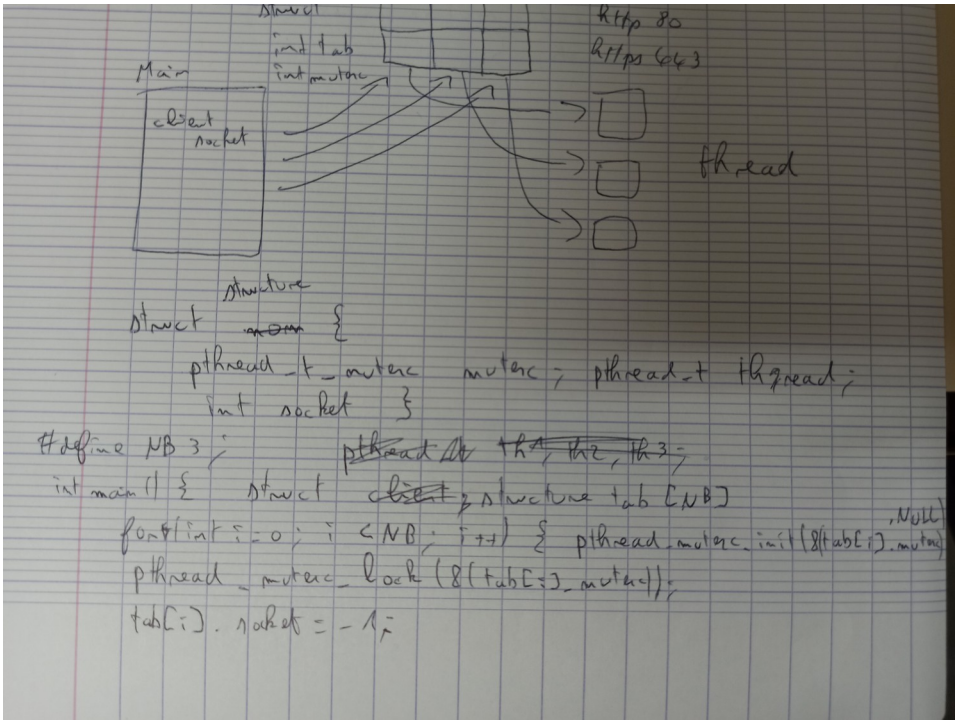
```
4) File Sent : 6298; file : ./docRoot//a4.jpg, stat = 0
Handling client 127.0.0.1
5) File Sent : 6039; file : ./docRoot//b1.jpg, stat = 0
Handling client 127.0.0.1
6) File Sent : 6406; file : ./docRoot//b2.jpg, stat = 0
Handling client 127.0.0.1
7) File Sent : 6772; file : ./docRoot//b3.jpg, stat = 0
Handling client 127.0.0.1
8) File Sent : 6596; file : ./docRoot//b4.jpg, stat = 0
^C--> recu signal 2
Fermeture des transmission
Fermeture du serveur
```

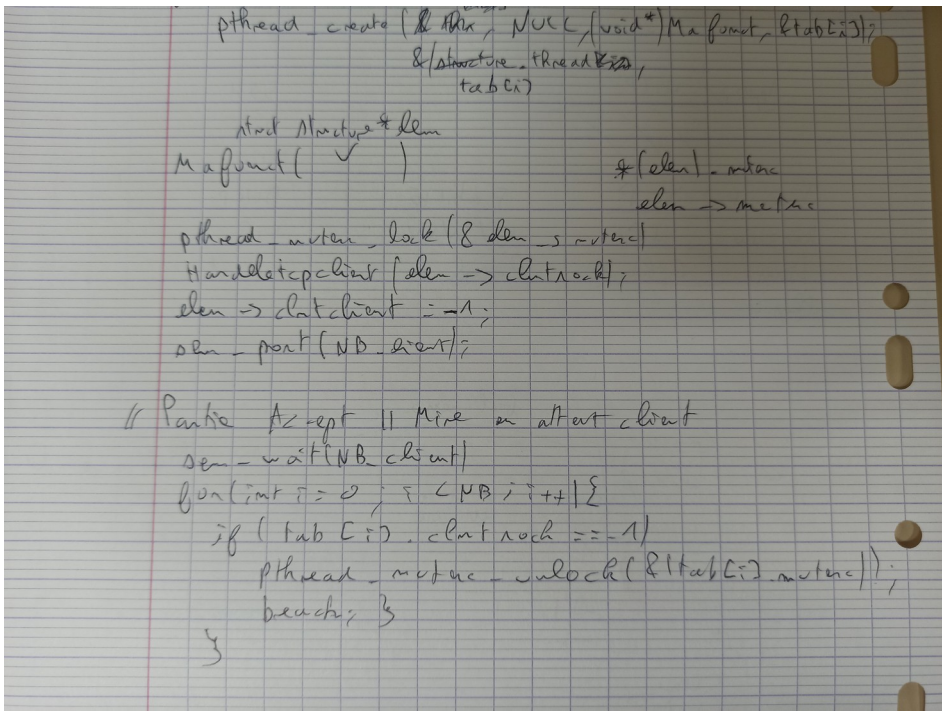
```
Int main()
{
signal(SIGINT, get_blody_signal);
signal(SIGPIPE, get_blody_signal);
}

void get_blody_signal(int s)
{
    printf("--> recu signal %d\n", s);
    die();
}
```

3. Etape2 :V0.3 Serveur « Threadé »

Voici mes notes avant de programmer la partie mutli-thread.





Voici la partie que j'ai rajouter concernant le multi-thread :

```
struct structure
{
    pthread_mutex_t mutex;
    pthread_t thread;
    int socket;
};

void Myfunct(struct structure *elem)
{
    while (1)
    {
        pthread_mutex_lock(&elem->mutex);
        HandleTCPClient(elem->socket);
        elem->socket = -1;
        sem_post(&NB_client);
    }
}

int main()
{
    for (int i = 0; i < NB; i++)
    {
        pthread_mutex_init(&(tab[i].mutex), NULL);  //creation d'un mutex pour chaque thread
        pthread_mutex_lock(&(tab[i].mutex));
        tab[i].socket = -1;
        pthread_create(&(tab[i].thread), NULL, (void *)Myfunct, &tab[i]);  //creation des threads dans la structure
    }
    sem_init(&NB_client, 1, NB);
    sem_wait(&NB_client);
        for (int i = 0; i < NB; i++)
        {
            if (tab[i].socket == -1)
            {
                tab[i].socket = clntSock;
                pthread_mutex_unlock(&(tab[i].mutex));
                break;
            }
        }
```

```
}
```

4. Etape3 :V0.4 Log

J’ai créé une fonction log, puis j’ai remplacé chaque printf par un appel à cette fonction avec en argument le texte à envoyé dans le fichier log.txt.

```
void log_funct(char *log_txt)
{
    int log = open("log.txt", O_RDWR | O_APPEND | O_CREAT, 0666); //ouvertue d’un fichier texte, et création si n’existe pas
    if (log != -1)
    {
        write(log, log_txt, strlen(log_txt)); //Ecriture dans le fichier du texte reçu en argument de la fonction
    }
    else
    {
        printf("Impossible de creer le fichier log.txt\n");
    }
}

int main()
{
   time_t timestamp = time(NULL);
    struct tm *timeInfos = localtime(&timestamp);
    sprintf(log_txt, "\nDate and local time : %04d/%02d/%02d %02d:%02d:%02d\n",       //Mettre la variable log_txt à la
            timeInfos->tm_year + 1900, timeInfos->tm_mon + 1, timeInfos→tm_mday,       valeur de la date et heure du log
            timeInfos->tm_hour, timeInfos->tm_min, timeInfos→tm_sec);

    log_funct(log_txt); Appel de la fonction pour afficher la date
}
```