HUANG Roger, 11807555

11Devoir Système d'exploitation, Réseaux

Exercice 1 :

1) C'est l'ensemble des adresses logiques, c'est-à-dire en fait l'ensemble de la mémoire : 256 Go.

2) Une page = 8ko. 1 adresse logique = 32 bits. Or 8ko = 8000 octets = 64 000 bits / 32 bits = 2000 adresses logiques dans une page. Donc 0 <=offset <= 1999. Pour coder 2000 nombres il faut 11 bits. **Le déplacement dans une page nécessite donc 11 bits.**

3) 1 page = 2^3 ko
256 Go = $2^8 * 2^{20}$ ko / 2^3 ko = 2^{25} pages. Chaque **numéro de pages est compris entre 1 et 2^{25}, donc stocké sur 25 bits**, donc sur 4 octets. Chaque entrée de la table des pages est donc stockée sur 8 octets**. La table des pages a donc une taille de $2^{25} * 2^3 = 2^{28}$ octets.**

4) Chaque page a une taille de 8ko, c'est-à-dire $2^3 * 2^{10} = 2^{13}$ octets. Il faut donc **2^{15} pages pour représenter la table des pages.**

On a besoin alors d'une table des tables de pages, contenant 2^{15} entrées, donc de taille $2^{15} * 8 = 2^{18}$ octets. De même, puisque chaque page a une taille de 2^{13} octets, il nous **faut 2^5 pages pour représenter cette table des tables de pages.**

On a besoin d'une profondeur supplémentaire. Cette nouvelle table est de taille $2^5 * 8 = 2^8$ octets < 8ko. Elle est donc représentée sur une **seule page**, pas besoin de profondeur supplémentaire.

On a besoin d'une **table de table de table des pages.** Profondeur de pagination = 3.

Une adresse logique est composé d''une page et d'un offset.
Il y a 4 octets pour coder le numéro de la page et 3 octets pour coder l'offset.

A partir de l'adresse logique, on récupère le numéro de page, on va chercher dans la table de table de table des pages, le cadre correspondant à la bonne table de table des pages, puis de la même façon la table de table des pages nous renseigne sur le cadre ou se trouve la bonne table des pages et à partir de cette table des pages on trouve le cadre physique où se trouve ce qu'on cherche. Dans ce cadre, on a plus qu'à ajouter l'offset pour trouver l'adresse physique.

5) La zone mémoire utilisée par ce processus est égal à 90 ko + 80 ko + 2 octets = 170 ko + 2 octets. On utilise ainsi 21 pages non fragmentées = 21 * 8ko = 168 ko. Il reste 2 ko (=2048 octets) + 2 octets = 2050 octets qui vont être stocké dans la dernière page. Cette dernière page connait une fragmentation interne de 8 ko (= 8*1024 octets) – 2050 octets = 6142 octets. On a besoin de 22 cadres finalement.

Exercice 2 :

1) a. Réseau : 197.68.202 .0 / Sous réseau : 197.68.202 .136 / Id machine : 0.0.0.1
b. Réseau : 42.0.0 .0 / Sous réseau : 42.128.0 .0 / Id machine : 0.18.200.42

2) a. Classe B
b. On veut 5 sous réseaux, il nous faut 3 bits (log (5) = 2,3…). Donc on a le masque de sous réseau suivant : 255.255.224.0
c. Il reste 13 bits pour identifier les machines hôtes. On a donc $2^{13} – 2$ nombre d'hôtes possibles.
d. Le sous réseau 2 est 10110110.00000000.010 11111. 11111111 Reseau/SousReseau/Hote
On a donc 182.0.95.255.

HUANG Roger, 11807555

Exercice 3 :

La machine X ne peut transmettre si la machine Y a comme numéro IP 192.168.113.177 car il s'agit d'une adresse particulière qui est dites privées : elles ne sont pas visibles sur le réseau internet donc les paquets ne sont pas transférés par des routeurs.

1) **Réseau A :**

4	5	0	2196
185211		001	0
5	6	checksum	
120.246.34.17			
155.236.200.157			

4	5	0	1324
185211		000	272
5	6	checksum	
120.246.34.17			
155.236.200.157			

Réseau B :

4	5	0	2196
185211		001	0
4	6	checksum	
120.246.34.17			
155.236.200.157			

4	5	0	1324
185211		000	272
4	6	checksum	
120.246.34.17			
155.236.200.157			

HUANG Roger, 11807555

Réseau C

4	5	0	2196
185211		001	0
3	6	checksum	
120.246.34.17			
135.236.200.157			

4	5	0	1324
185211		000	272
3	6	checksum	
120.246.34.17			
135.236.200.157			

2) Les fragments d'un paquet peuvent prendre des chemins différents, ce qui empêche de rassembler les fragments avant la destination finale. De plus, des fragments peuvent être perdu en route, donc pour reconstituer il faudrait attendre l'arrivée d'un paquet ce qui retarderait le trafic.

Exercice 4 :

La totalité du code est donné dans le dossier envoyé : clientadminUDP.c / clientTCP.c / serveur.c. Les extraits de code respectifs à chaque question est donné dans la suite de ce document.

Pour compiler :
gcc -Wall clientadminUDP.c -o clientAdmin
gcc -Wall clientTCP.c -o client
gcc -Wall serveur.c -o serveur -lpthread

Pour exécuter :

Lancer le serveur depuis un terminal : ./serveur
Puis simuler les connexions client : ./client ou ./clientAdmin

HUANG Roger, 11807555

Exercice 4 :

1. Ecrivez le code du programme client ordinaire.

```c
//Client ordinaire TCP
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#include <arpa/inet.h>


int main(){

    int sock;
    if( (sock = socket(AF_INET, SOCK_STREAM, 0)) == -1 ) {
        perror("Erreur de creation socket client");
        exit(1);
    }

    struct sockaddr_in srv;
    socklen_t sl = sizeof(struct sockaddr_in); // taille d'une addresse IPV4

    /* Attributs du serveur */
    srv.sin_family = AF_INET;
    srv.sin_port = htons(20000);
    srv.sin_addr.s_addr = inet_addr("127.0.0.1");

    if( connect(sock, (struct sockaddr*) &srv,sl) < 0 ){
        perror("Erreur de connect");
        exit(1);
    }

    struct in_addr *ipclient = malloc(sizeof(struct in_addr) );
    ipclient->s_addr = gethostid();

    char * phrase = "mon numero IP est ";
    char* monip = inet_ntoa( *ipclient ) ;
    char * ip2 = malloc( strlen(monip) + strlen(phrase) + 1);
    strcpy( ip2, phrase);
    strcat( ip2 , monip);


    int nb_octets = write(sock, ip2, strlen(ip2));
    if(nb_octets<0){
        perror("Erreur d'ecriture");
    }

    close(sock);
    return 0;
}
```

2. Ecrivez le code du programme client administrateur.

```c
//Client admin UDP
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/socket.h>
#include <unistd.h>
#include <arpa/inet.h>


int main(){

    int sock;
    if( (sock = socket(AF_INET, SOCK_DGRAM, 0))== -1 ) {
        perror("Erreur de creation socket admin");
        exit(1);
    }

    struct sockaddr_in srv;
    socklen_t sl = sizeof(struct sockaddr_in);
```

```
23      /* Attributs du serveur */
24      srv.sin_family = AF_INET;
25      srv.sin_port = htons(30000);
26      srv.sin_addr.s_addr = inet_addr("127.0.0.1");
27
28      //Envoi
29      printf("Envoi de la requete : je veux connaitre le nombre de clients\n");
30      char* demande = "Je veux connaitre le nombre de clients";
31      sendto(sock, demande, strlen(demande), 0, (struct sockaddr *) &srv, sl);
32
33      //Reçu
34      int nbClients;
35      recvfrom(sock, &nbClients, sizeof(int), 0, NULL, NULL);
36      printf("Reçu : %d\n", nbClients);
37      close(sock);
38      return 0;
39
40  }
```

3. Ecrivez la fonction associée au thread t1.

```
//gestion du client ordinaire TCP
void* f2 (void* arg){

    printf("thread admin cree\n");

    int new_sock, sock;
    pthread_t t3;

    if( (sock = socket(AF_INET, SOCK_STREAM, 0))== -1  ){
        perror("Erreur de creation socket 2");
        exit(1);
    }

    struct sockaddr_in srv, clt;
    socklen_t sl = sizeof(struct sockaddr_in);

    /* Attributs du serveur */
    srv.sin_family = AF_INET;
    srv.sin_port = htons(20000);
    srv.sin_addr.s_addr = inet_addr("127.0.0.1");

    /* liaison socket - serveur */
    if( bind(sock, (struct sockaddr *) &srv, sl)  < 0  ){
        perror("Erreur de bind 2");
        exit(1);
    }

    if( listen(sock, 5) != 0){
        perror("Erreur de listen");
        exit(1);
    }

    while(1){

        printf("Attente client ordinaire sur le port 20000...\n");

        new_sock = accept(sock, (struct sockaddr *) &clt, &sl);
        if(new_sock < 0){
            perror("Erreur de accept");
            exit(1);
        }

        printf("Client Accepte \n");

        sem_wait(&S1) ;
        nbClients++;
        sem_post(&S1) ;

        pthread_create(&t3, NULL, newClient, &new_sock);

    }

    close(sock);
    pthread_exit(NULL);
}
```

```c
//Fonction - prise en charge d'un client ordinaire
void* newClient(void* sock) {
    int* new_sock = (int*) sock;

    char message[MAXMSGSIZE];
    bzero(message, MAXMSGSIZE);

    int nb_octets = read(*new_sock, message, MAXMSGSIZE);
    printf("Le serveur a recu %d octets du client : %s\n", nb_octets, message);

    sem_wait(&S1) ;
    nbClients--;
    sem_post(&S1) ;

    close(*new_sock);
    pthread_exit(NULL);
}
```

4. Ecrivez la fonction associée au thread t2.

```c
//gestion du client admin
void* f1 (void* arg){
    printf("thread client cree\n");

    int nb_octets;
    int sock;
    char message[MAXMSGSIZE];

    if( ( sock = socket(AF_INET, SOCK_DGRAM, 0)) == -1 ){
        perror("Erreur de creation socket 1");
        exit(1);
    }

    struct sockaddr_in srv, clt;
    socklen_t sl = sizeof(struct sockaddr_in);

    /* Attributs du serveur */
    srv.sin_family = AF_INET;
    srv.sin_port = htons(30000);
    inet_aton("127.0.0.1", &(srv.sin_addr));

    /* liaison socket - serveur */
    if( bind(sock, (struct sockaddr *) &srv, sl)  < 0  ){
        perror("Erreur de bind 1");
        exit(1);
    }

    fflush(stdout);

    while (1) {

        printf("Attente admin sur port 30000...\n") ;
        nb_octets = recvfrom(sock, message, MAXMSGSIZE, 0,(struct sockaddr *) &clt, &sl) ;
        printf("Admin accepte\n");

        message[nb_octets] ='\0';
        printf("Requete recue : %s\n", message);

        sem_wait(&S1) ;
        sendto(sock, &nbClients , sizeof(int) , 0, (struct sockaddr *) &clt, sl);
        sem_post(&S1) ;

        printf("Reponse envoyee\n");

        fflush(stdout);
    }

    close(sock);
    pthread_exit(NULL);
}
```

5. Code du serveur (avec le reste, se référer au code dans le dossier) :

```c
int main(){
    sem_init(&S1, 0, 1); // 1 = la variable est accessible et 0 = la variable est en cours d'utilisation
    pthread_t t1, t2;

    if( pthread_create(&t1, NULL, f1, NULL) == -1){
        perror("pthread_create1");
        return EXIT_FAILURE;
    }

    if( pthread_create(&t2, NULL, f2, NULL) == -1){
        perror("pthread_create2");
        return EXIT_FAILURE;
    }
    while(1);
    return 0;
}
```