DS Programmation Système

2016-2017 Semestre 4 Nom : Prénom :

Exercice 1 :

Nous considérons le code ci-dessous. Celui-ci est le code d'un serveur TCP. Il est incomplet. Son fonctionnement est le suivant :

Une fois un client connecté,

(i) Il se met en attente de donnée en réception
(ii) Affiche les données reçues (ascii)
(iii) Lit sur l'entrée standard un contenu ascii tapé par l'utilisateur
(iv) Envoi ce contenu au client
(v) Reboucle sur le point (i)

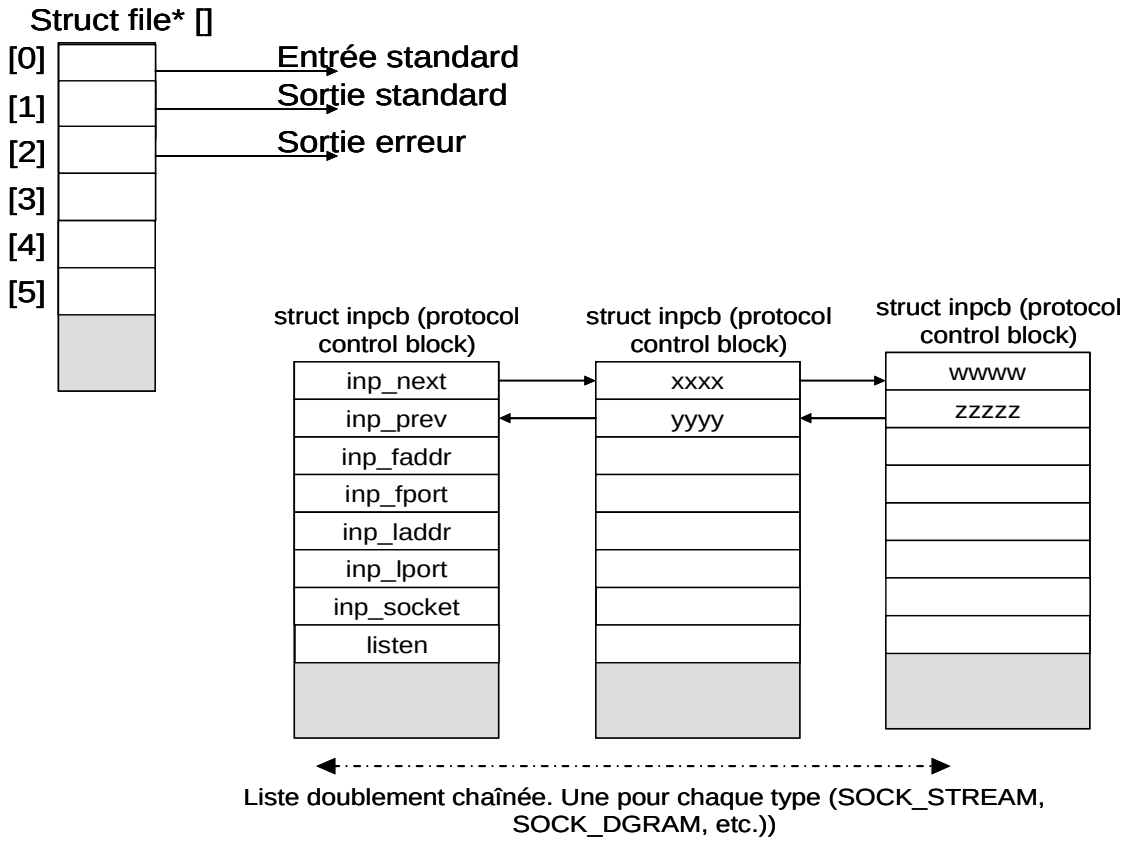
La boucle s'arrête quand il n'y a plus de données en réception (le client a fermé la connexion).

1. Complétez le code. Des commentaires « A compléter » sont placés aux différents endroits où vous devez compléter. Les prototypes se trouvent en annexe.

```
int main()

{

        int sockfd, confd ;

        char buffer[512];

        struct addrinfo hints, *res;


        memset(&hints,0,sizeof(hints));

        hints.ai_family = AF_UNSPEC;

        hints.ai_socktype = SOCK_STREAM;

        hints.ai_flags = AI_PASSIVE;


        if(getaddrinfo(NULL,"2000",&hints,&res)!=0) erreur("Erreur getaddrinfo()",1);

        //A compléter

        if((sockfd=socket(          ,          ,          ))<0) perror(« Erreur socket ») ;
```

```
	// Association de la socket à un numero de port: A compléter




	if(listen(sockfd, SOMAXCONN)<0) erreur("Erreur listen()",2);


	while((confd=accept(sockfd, NULL, NULL))>0)
	{
		//A compléter









	}
	/* On est sorti de la boucle while. C'est donc qu'il y a une erreur sur accept()     */
	perror("Erreur accept()");
	return(0);
}
```

2. Un client se connecte sur ce serveur. L'adresse IP des clients est 23.1.1.1. L'adresse du serveur est 45.2.2.2. Le port côté client est 3452. Celui du serveur est 2000. Complétez le tableau des descripteurs et des structures décrivant les connexions au niveau du système.



Exercice 2 :

Ecrire un programme qui créer trois threads. Les threads correspondent à la même fonction. La fonction thread affiche bonjour.

Exercice 3 :

Complétez le code précédent (celui de l'exercice 2) de manière à ce que après l'affichage du bonjour les 3 threads s'attendent mutuellement. Autrement dit, il y a une barrière juste après le bonjour. Après cela les threads affichent « Aurevoir ».

ANNEXE

```
/* La structure addrinfo est la suivante: */

struct addrinfo {

        int     ai_flags;
        int     ai_family;
        int       ai_socktype;
        int     ai_protocol;
        size_t  ai_addrlen;
        struct sockaddr  *ai_addr;
        char   *ai_canonname;
        struct addrinfo *ai_next;
};
```

Prototypes

int socket(int *domain***, int** *type***, int** *protocol***);**
int bind(int *sockfd***, const struct sockaddr *****addr***, socklen_t** *addrlen***);**
int connect(int *sockfd***, const struct sockaddr *****serv_addr***, socklen_t** *addrlen***);**
int accept(int *sockfd***, struct sockaddr *****adresse***, socklen_t *****longueur***);**
int listen(int *sockfd***, int** *backlog***);**
ssize_t recv(int *s***, void *****buf***, ssize_t** *len***, int** *flags***);**
ssize_t read(int *fd***, void *****buf***, size_t** *count***);**
ssize_t write(int *fd***, const void *****buf***, size_t** *count***);**
ssize_t send(int *s***, const void *****buf***, size_t** *len***, int** *flags***);**
int close(int fd);

int pthread_create(pthread_t *****thread*, const pthread_attr_t *****attr***, void *(*****start_routine***) (void *), void *****arg***);**
int pthread_join(pthread_t *thread***, void ****retval***);**

<u>Pour initialiser un mutex :</u> pthread_mutex_t *monMutex* **= PTHREAD_MUTEX_INITIALIZER;**
int pthread_mutex_lock(pthread_mutex_t *****mutex*));**
int pthread_mutex_trylock(pthread_mutex_t *****mutex*);**
int pthread_mutex_unlock(pthread_mutex_t *****mutex*);**

<u>Pour initialiser une variable de condition</u> : pthread_cond_t *cond* **= PTHREAD_COND_INITIALIZER;**
int pthread_cond_signal(pthread_cond_t *****cond*);**
int pthread_cond_broadcast(pthread_cond_t *****cond*);**
int pthread_cond_wait(pthread_cond_t *****cond*, pthread_mutex_t *****mutex***);**

sem_t monSemaphore;
int sem_init(sem_t *sem, int pshared, unsigned int value);
int sem_wait(sem_t **sem*);
int sem_trywait(sem_t **sem*);
int sem_post(sem_t *sem);
int sem_getvalue(sem_t *sem, int *sval);
int sem_destroy(sem_t *sem) ;