DS Programmation Système

2017-2018 Semestre 4 Nom : Prénom :

Exercice 1 : Création de thread et accès à un tableau partagé

Un tableau de 100 entiers contient des entiers de 0 à 99. Une fonction lit une case du tableau et l'affiche. La fonction boucle jusqu'à que toutes les cases du tableau soient affichées. Cette fonction sera lancé comme un thread 5 fois (il y a aura donc 5 threads).

Proposez le code du main et le code de la fonction décrite ci-dessus. Attention, lors de l'exécution du programme chaque case du tableau devra ^etre affichée une fois et une seule.

Exercice 2 : Barrière

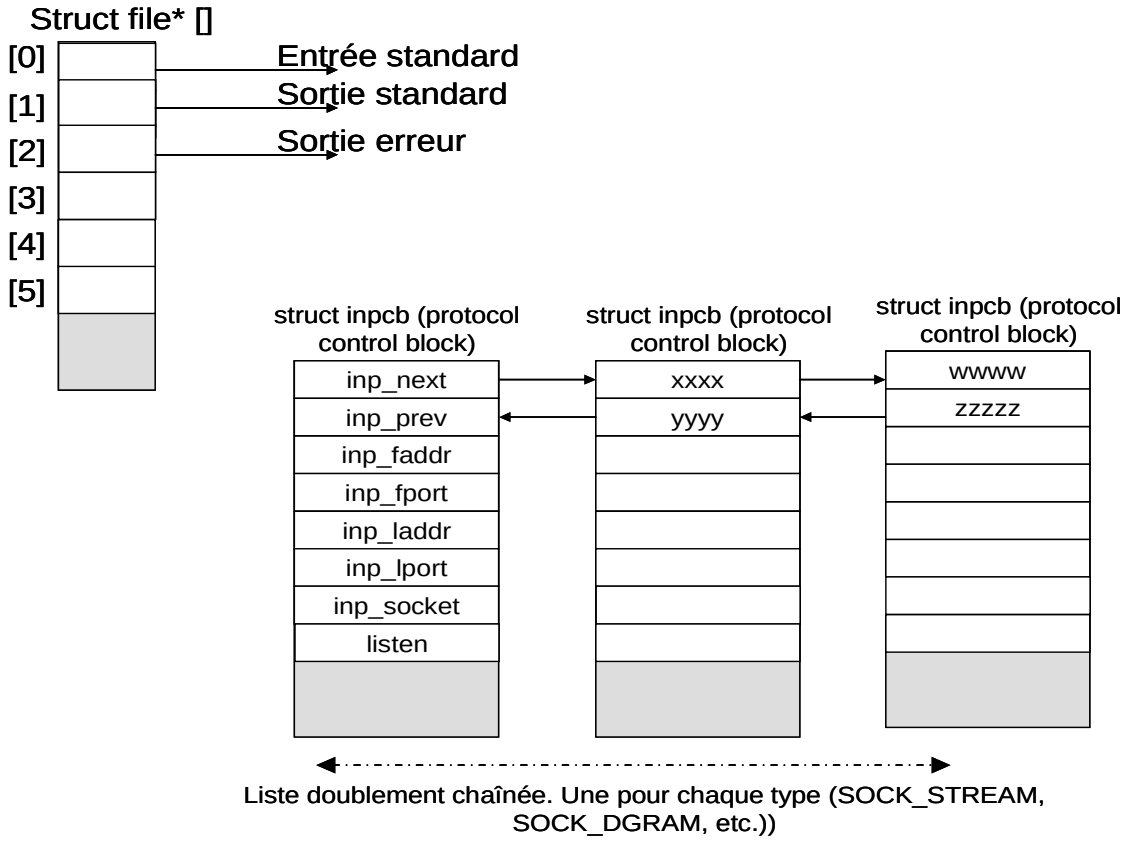
Donnez le code correspond à la fonction d'un thread (vous n'avez pas donner celle du main()). Ce thread sera lancé 5 fois (5 threads). Chaque thread affiche « Bonjour », attend un temps aléatoire (sleep(rand()%10)). Les threads doivent alors s'attendre mutuellement (barrière) avant d'afficher un dernier message.

Exercice 3 :

Nous considérons un serveur TCP. Deux clients se connectent sur ce serveur. Les adresses IP des clients sont 23.1.1.1 et 56.87.121.1. L'adresse du serveur est 45.2.2.2. Les ports côtés client sont 3452 et 6523. Celui du serveur est 2000.

1 . Complétez le tableau des descripteurs et des structures décrivant les connexions au niveau du système.

2 . Même question au niveau d'un des deux clients.



Exercice 4 :

Donnez le code d'un programme qui intercepte le signal Ctrl-C et affiche « fin du programme » quand celui-ci est saisi au clavier par l'utilisateur.

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***);**

Pour initialiser un mutex : 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*);**

Pour initialiser une variable de condition : 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) ;

sighandler_t signal(int *signum***, sighandler_t** *handler***);**