Université Paris Cité Année 2022–2023

Programmation Réseaux - Examen

Durée : 2h30
Une feuille A4 de notes manuscrites autorisée

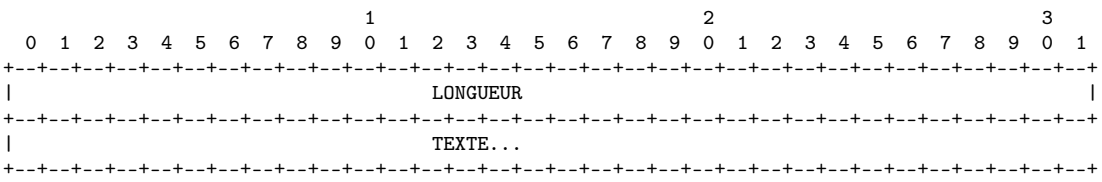
Lors de cet examen :

— votre code doit être écrit en **langage C** et fonctionner sur les **machines de l'UFR**,

— vous **ne devez pas** gérer les erreurs des appels systèmes,

— les nombres de lignes annoncés entre parenthèse sont indicatifs du nombre de lignes de code nécessaires et ne prennent pas en compte les lignes de déclaration des variables.

Exercice 1 Une application 1 veut envoyer un message au format suivant à une application 2 :



où `TEXTE` est une suite de `LONGUEUR` caractères et `LONGUEUR` est au format big-endian.

1. Donnez les tailles minimale et maximale du champs `TEXTE`.

2. Écrire les lignes de code de l'application 1 pour initier une connexion IPv4 ou IPv6 (les deux doivent être possible) à la machine `hope` sur le port 4444.

   On ne demande pas d'écrire le code d'envoi du message.

3. On suppose que ces deux applications ont établi une connexion TCP et que la socket de communication de l'application 2 se nomme `sock2`. Écrire les lignes de code de l'application 2 pour recevoir ce message et afficher son champs `TEXTE`.

Exercice 2 On suppose que les appels de fonctions s'exécutent sans erreur.

```
1  //application 1
2  int main() {
3    int sock=socket(PF_INET, SOCK_DGRAM, 0);
4
5    int ok=1;
6    setsockopt(sock, SOL_SOCKET, SO_BROADCAST, &ok, sizeof(ok));
7
8    struct sockaddr_in adr;
9    memset(&adr, 0, sizeof(adr));
10   adr.sin_family = AF_INET;
11   adr.sin_port = htons(1234);
12   inet_pton(AF_INET, "255.255.255.255", &adr.sin_addr);
13
14   int i, nb;
15   srand(time(NULL));
16   for(i=0; i<=20; i++) {
17     nb = htonl(rand()%50-25);
18     sendto(sock, &nb, sizeof(nb), 0, (struct sockaddr*) &adr, sizeof(struct sockaddr_in));
19   }
20
21   close(sock);
22   return 0;
23 }
```

```c
//application 2
void *fa(void *arg) {
  int *v = (int *)arg;
  double a = *v;
  for(int i=0; i<3; i++)
    a = a * (log(a)+1);
  int fd = open("fic.txt", O_WRONLY|O_CREAT|O_APPEND, 0600);
  write(fd, &a, sizeof(double));
  close(fd);

  return NULL;
}

void *fb(void *arg) {
  int *v = (int *)arg;
  double b = *v;
  for(int i=0; i<3; i++)
    b = b + (exp(b)+1);
  int fd = open("fic.txt", O_WRONLY|O_CREAT|O_APPEND, 0600);
  write(fd, v, sizeof(double));
  close(fd);

  return NULL;
}

int main() {
  int sock=socket(PF_INET,SOCK_DGRAM,0);

  struct sockaddr_in adrsock;
  memset(&adrsock, 0, sizeof(adr));
  adrsock.sin_family = AF_INET;
  adrsock.sin_port = htons(1234);
  adrsock.sin_addr.s_addr = htonl(INADDR_ANY);

  bind(sock,(struct sockaddr *)&adrsock,sizeof(struct sockaddr_in));

  int n;
  pthread_t thread;
  while(1) {
    recvfrom(sock, &n, sizeof(n), 0, NULL, NULL);
    n = ntohl(n);
    if(n > 0)
      pthread_create(&thread, NULL, fa, &n);
    else
      pthread_create(&thread, NULL, fb, &n);
  }

  return 0;
}
```

1. Quel type de communication est utilisée ici ?

2. Connaissant les valeurs de `nb` envoyées par l'application 1 à l'application 2, pour quelles raisons ne peut-on savoir, sans consulter le contenu du fichier `fic.txt`, ce qu'il contient ?

3. Écrire le code qui permet de corriger le problème souligné à la question précédente en précisant où il s'insère dans le code donné ci-dessus.

4. Décrire, étape par étape, le protocole entre les deux applications 1 et 2 en prenant en compte vos corrections.

5. On efface les lignes 39 et 46 du code de l'application 2. Corrigez ce code (précisez où il s'insère ou les lignes modifiées), afin que l'application 2 s'exécute toujours normalement.

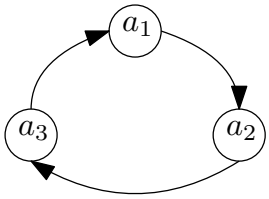
Exercice 3 Le but est d'écrire une application sur IPV6 qui permet la communication en anneau avec ses pairs.

Cela signifie, par exemple, que si 3 instances de l'application a_1, a_2 et a_3 communiquent :

— a_1 envoie ses messages à a_2 et reçoit ceux de a_3

— a_2 envoie ses messages à a_3 et reçoit ceux de a_1

— a_3 envoie ses messages à a_1 et reçoit ceux de a_2



1 Insertion dans l'anneau

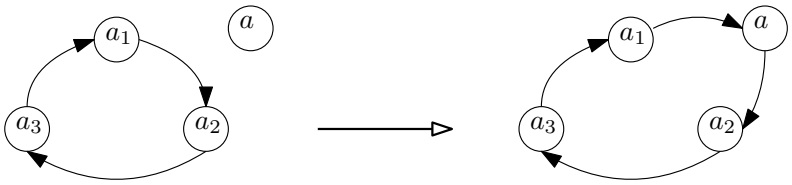
Au lancement d'une instance a de l'application, celle-ci commence par s'insérer dans l'anneau de pairs. Elle doit donc demander à s'insérer et attendre qu'un pair accepte qu'elle s'insère entre lui et le pair suivant. Le protocole est le suivant :

étape 1 : au lancement, l'instance a envoie un message « **HELLO** » sur le port 4444 à toutes les machines du réseau local ayant une application écoutant sur ce port.

étape 2 : Elle attend en retour une réponse et s'occupe de la première réponse reçue. Si la réponse provient de l'instance a_1 et que a_1 est suivie de a_2 dans l'anneau, alors le message contient l'adresse IPv6 de l'hôte de a_2 sous sa forme chaînes de caractères. L'instance a stocke cette adresse.

étape 3 : dès qu'elle reçoit une réponse d'une instance a_1, elle lui renvoit l'argument du main qui correspond à son adresse IPv6 sous sa forme chaînes de caractères. Elle ne se préoccupe pas des autres réponses éventuelles.

L'instance a peut alors s'insérer dans l'anneau pour recevoir les messages de a_1 et envoyer ses messages à a_2.



1. À l'étape 1, quel type de socket IPV6 et quel type de communication doit-on utiliser ? Pourquoi ?
2. Il manque la valeur d'une donnée réseau dans l'énoncé. Donnez une valeur possible à cette donnée et expliquez en quoi cette valeur est pertinente.
3. À l'étape 2, quel type de communication différente de celle de l'étape 1 peut-on utiliser ? Pourquoi ?
4. Le protocole ci-dessus est incomplet. Quel problème n'est pas évoqué ? Expliquez quelles sont les raisons possibles pour que ce problème advienne. Proposez une solution à ce problème sans écrire le code mais en expliquant ce qu'il faudrait faire (scénario).
5. Écrire le code de l'application correspondant aux trois étapes. (∼20 lignes)

2 Partage des tâches

Les messages circulant dans l'anneau correspondent à des tâches. Chaque instance de l'anneau peut recevoir une tâche via son entrée standard ou via un message reçu du pair précédent dans l'anneau.
Si le message provient de l'entrée standard, alors elle l'envoit au pair suivant dans l'annneau.
Si le message provient du pair précédent, elle attend pendant maximum 20 secondes que l'utilisateur signale qu'il se saisit de la tâche, sinon la tâche est envoyée au pair suivant. Les applications doivent donc faire plusieurs actions simultanément :

action 1 : écouter sur le port 5555 les messages UDP de tâches provenant du pair précédent et réagir,

action 2 : lire sur l'entrée standard une nouvelle tâche et réagir,

action 3 : écouter sur le port 4444 les demandes d'insertion dans l'anneau et réagir.

Un message de tâche ne contient pas plus de 255 caractères.

1. Écrire le code C de l'application qui initialise les différentes sockets et adresses nécessaires aux trois actions. (∼15 lignes)

2. Que faudrait-il ajouter dans votre code précédent pour pouvoir faire tourner plusieurs instances d'un anneau sur la même machine ?

3. Écrire le code C de l'application permettant la simultanéité des actions sans créer de nouveau processus système ou léger et sans écrire, pour le moment, le code correspondant aux réactions des actions. Vous mettrez, pour le moment, à la place du code correspondant aux réactions des trois actions les commentaires respectifs `//reaction1`, `//reaction2` ou `//reaction3`. La prise en charge de la réaction 1 sera traité à la question suivante. Vous ne devez pas vous en préoccuper dans cette question. (∼15 lignes)

4. La réaction de l'action 1 consiste à attendre pendant au plus 20 secondes que l'utilisateur entre n'importe quelle chaîne de caractères sur l'entrée standard. Dans ce cas, cela signifie que la tâche reçue va être traitée par l'utilisateur et elle ne doit pas être renvoyée dans l'anneau. L'action 1 est donc terminée. Sinon (20 secondes se sont écoulées sans message sur l'entrée standard), il faut envoyer au pair suivant dans l'anneau la tâche reçue.

   Par souci de simplification, on interdit à l'utilisateur d'initier une action 2 pendant ces 20 secondes. Il doit également laisser un temps minimal de 20 secondes entre deux actions 2.

   Écrire le code de la réaction à l'action 1. Attention, vous ne devez toujours pas créer de nouveau processus système ou léger, mais la simultanéité des tâches doit être conservée. Cela implique qu'outre l'écriture du code correspondant au commentaire `reaction1`, vous devez modifier le code de la question précédente. Ne réécrivez pas tout, mais juste les morceaux modifiés ou ajoutés en expliquant où ils s'insèrent dans votre code de la question précédente. (∼10 lignes)

Prototypes de fonctions pouvant être utiles

```
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
int close(int fd);
int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
void FD_CLR(int fd, fd_set *set);
int  FD_ISSET(int fd, fd_set *set);
void FD_SET(int fd, fd_set *set);
void FD_ZERO(fd_set *set);
pid_t fork(void);
void freeaddrinfo(struct addrinfo *res);
int getaddrinfo(const char *node, const char *service,
                        const struct addrinfo *hints, struct addrinfo **res);
uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);
int listen(int sockfd, int backlog);
int memcmp(const void *s1, const void *s2, size_t n);
void *memcpy(void *dest, const void *src, size_t n);
void *memmove(void *dest, const void *src, size_t n);
void *memset(void *s, int c, size_t n);
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
int open(const char *pathname, int flags, mode_t mode);
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);
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
const char *inet_ntop(int af, const void *src, char *dst, socklen_t size);
int inet_pton(int af, const char *src, void *dst);
int poll(struct pollfd *fds, nfds_t nfds, int timeout);
ssize_t read(int fd, void *buf, size_t count);
ssize_t recv(int sockfd, void *buf, size_t len, int flags);
ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,
                       struct sockaddr *src_addr, socklen_t *addrlen);
int select(int nfds, fd_set *readfds, fd_set *writefds,
                  fd_set *exceptfds, struct timeval *timeout);
ssize_t send(int sockfd, const void *buf, size_t len, int flags);
ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,
                      const struct sockaddr *dest_addr, socklen_t addrlen);
int setsockopt(int sockfd, int level, int optname,
                      const void *optval, socklen_t optlen);
int snprintf(char *str, size_t size, const char *format, ...);
int socket(int domain, int type, int protocol);
int sprintf(char *str, const char *format, ...);
int strcmp(const char *s1, const char *s2);
int strncmp(const char *s1, const char *s2, size_t n);
```

```
char *strcpy(char *dest, const char *src);
char *strncpy(char *dest, const char *src, size_t n);
pid_t waitpid(pid_t pid, int *wstatus, int options);
ssize_t write(int fd, const void *buf, size_t count);
```