Université Paris Cité Année 2023–2024

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,

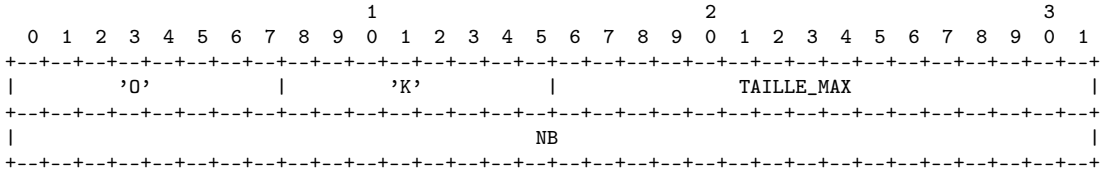
— les nombres de lignes annoncés entre parenthèse sont indicatifs du nombre de lignes de code nécessaires et ne prennent pas en compte les lignes de déclaration des variables.

> Lorsque vous aurez du code à modifier, il ne faut pas tout réécrire. Vous pourrez réutiliser le code du sujet et numéroter votre code afin de le réutiliser facilement. Ainsi, vous pourrez insérer votre code ou modifier ou remplacer une ligne en précisant, à l'aide des numéros de lignes, où cela se fait. Par exemple, si vous voulez récupérer les lignes 1 à 7, insérer du code, puis récupérer les lignes 18 et 21 à 25, vous pouvez écrire :
> ```
> lignes 1-7
> [votre code ...]
> lignes 18, 21-25
> ```

Exercice 1 On décrit un protocole de téléchargement en UDP d'un fichier, d'une application mono-agent B vers une application A :

Étape I l'application A envoie sur le port 7777 un message composé uniquement d'un nom de fichier à l'application B,

Étape II Si le fichier demandé n'est pas disponible [1], l'application B répond par le message de 3 caractères `NOK` puis termine. A termine également dans ce cas.
Si le fichier est disponible, l'application B commence par envoyer un message au format suivant



où `TAILLE_MAX` et `NB` sont au format réseau. `NB` est le nombre de messages qui seront envoyés par l'application B à l'application A et `TAILLE_MAX` est la taille maximale de chaque message envoyé. A récupère le nombre de messages envoyés et leur taille maximale.

Étape III Ensuite, l'application B envoie `NB` messages correspondant aux données du fichier demandé. Chaque message est de taille `TAILLE_MAX` sauf le dernier qui peut être plus court.

1. nous considèrerons ici, que cela signifie qu'il n'existe pas de fichier du nom reçu dans le répertoire courant

L'application A stocke les données reçues dans un fichier de même nom que celui du fichier demandé.

Étape IV Les deux applications terminent.

1. Quels problèmes peuvent survenir à l'étape II ?

   On ignorera ces problèmes dans la suite de l'exercice.

2. Pourquoi le fichier téléchargé par l'application A peut être corrompu ?

3. Donner une valeur raisonnable pour `TAILLE_MAX` et justifier ce choix.

On suppose dans tout l'exercice que l'on dispose des fonctions suivantes :

— `int fic_existe(char *nomfic)` qui prend un nom de fichier en paramètre et retourne 1 si un fichier de ce nom est présent dans le répertoire courant, 0 sinon.

— `int fic_taille(char *nomfic)` qui prend une référence d'un fichier en paramètre et retourne la taille du fichier correspondant si celui-ci existe, -1 sinon.

— `int cree_fic(char *nomfic)` qui crée ou écrase le fichier de référence `nomfic`. La fonction retourne 0 si tout s'est bien passé, 1 en cas de problème.

— `int lire_fic(char *nomfic, char *buf, int i, int tmax)` qui lit sur le disque le fichier de référence `nomfic` à partir de l'octet `i * tmax` sur au plus `tmax` octets (*ie.* `min(tmax, len-i)`) où `len` est la longueur du fichier, et stocke les `tmax` octets lus dans `buf` qui doit être alloué au préalable. La fonction retourne le nombre d'octets lus, c'est-à-dire `tmax` sauf s'il y a moins d'octets à lire. En cas de problème, elle retourne -1.

— `int texte_append(char *nomfic, char *texte, int i, int tmax)` qui ajoute au fichier de référence `nomfic`, à partir de la position `i * tmax`, la chaîne de caractères `texte`. La fonction retourne 0 si tout s'est bien passé, 1 en cas de problème.

La communication entre A et B se fait sur IPv6. Les entités de A doivent communiquer avec une entité de B qui tourne sur une machine reliée à l'internet global et a les caractéristiques suivantes :

```
$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 7c:57:58:68:ae:a9 brd ff:ff:ff:ff:ff:ff
    altname eno1
    altname enp0s31f6
    inet 192.168.70.100/24 brd 192.168.70.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fdc7:9dd5:2c66:be86:7e57:58ff:fe68:aea9/64 scope global
       valid_lft forever preferred_lft forever
    inet6 fe80::7e57:58ff:fe68:aea9/64 scope link
       valid_lft forever preferred_lft forever
```

On suppose, pour le moment, que les problèmes évoqués à la question 2 ne surviennent pas.

4. Écrire le code de l'application A correspondant aux étapes I et II, sachant que le nom du fichier à télécharger est passé en argument du programme. (∼15 lignes)

5. Écrire le code de l'application B correspondant aux étapes I et II. (∼20 lignes)

6. Écrire le code de l'application A correspondant aux étapes III et IV. (∼10 lignes)

7. Écrire le code de l'application B correspondant aux étapes III et IV. (∼6 lignes)

On veut maintenant modifier le protocole afin qu'il soit plus fiable. Pour cela, on modifie l'étape III du protocole :

— l'application B envoie les `NB` messages, chacun étant précédé de son numéro codé sur 4 octets et au format réseau. Les messages sont numérotés dans l'ordre et à partir de 0. Chaque message est donc de taille `TAILLE_MAX+4` sauf éventuellement le dernier qui peut être plus court.

— L'application A stocke les données reçues dans un fichier de même nom que celui du fichier demandé. Chaque fois qu'elle constate qu'un message numéroté manque, elle fait une demande de réémission du message en envoyant un message à B composé uniquement du numéro du message manquant.

8. Modifiez le code de B correspondant à l'étape III en accord avec le protocole modifié. B devra en parallèle :
   — lire les demandes de réémission d'un message provenant de A et réémettre le message demandé vers A,
   — envoyer les messages numérotés à A.

   Il devra par ailleurs attendre au moins cinq secondes à chaque fois qu'il pensera avoir tout envoyé afin de ne pas manquer des demandes de réémissions de A. Enfin, B ne devra créer aucun nouveau processus système ou léger. (∼26 lignes)

   Vous pourrez supposer que vous disposez de la fonction `void mess_num(int nb, char *mess)` qui ajoute au début de `mess` les 4 octets de l'entier `nb` mis au format réseau.

Exercice 2 On considère les codes des applications 1 et 2 présentés à la fin de l'exercice.

On rappelle qu'on ne se préoccupe pas ici des erreurs des appels systèmes. Vous considérerez de plus que les exécutions ne rencontrent pas de problème réseau.

1. Les lignes 26 et 32 de l'application 1 peuvent poser problème. Pourquoi ? Corriger la ligne 26 sans changer le reste du code.

2. Décrire textuellement, étape par étape, le protocole entre les deux applications 1 et 2.

3. Écrire le code de la fonction `void prepa_recept(int sock)`. (∼6 lignes)

4. Écrire le code de la fonction `int prepa_env_recept(char *h, char *p)`. (∼15 lignes)

5. Modifier le code de l'application 2 afin que celle-ci puisse communiquer avec deux entités de l'application 1 en parallèle et déterminer un gagnant (le premier qui donne une réponse correcte) s'il y en a un. L'application 2 attend d'être en communication avec deux entités de l'application 1 avant d'envoyer en parallèle à chacune le calcul. Elle renvoie à chaque entité son statut, 'g' pour gagnant, 'p' pour perdant et 'n' lorsqu'il n'y a pas de gagnant.

   Pour avoir tous les points à cette question, vous devez respecter les contraintes suivantes : utiliser des threads et ne pas utiliser de variable globale. (∼35 lignes sans compter les lignes réutilisées)

6. Décrire textuellement les étapes du côté de l'application 2 pour sécuriser les échanges des lignes 20 et 21 avec OpenSSL.

```c
//Application 1

#define BUF_SIZE 1024

int main(int argc, char const *args[]) {
        int sock1 = socket(AF_INET6, SOCK_DGRAM, 0);

        struct sockaddr_in6 adr;
        memset(&adr, 0, sizeof(adr));
        adr.sin6_family = AF_INET6;
        inet_pton(AF_INET6, "ff12::ae2:b", &adr.sin6_addr);
        adr.sin6_port = htons(1212);

        char buf[BUF_SIZE];
        int l;
        l = sprintf(buf, "%s\0%s\0", "nivose.informatique.univ-paris-diderot.fr", args[1]);
        sendto(sock1, buf, len, 0, (struct sockaddr*)&adr, sizeof(adr));
        close(sock1);

        int sock2 = socket(PF_INET6, SOCK_STREAM, 0);
        struct sockaddr_in6 adr2 = {AF_INET6, htons(atoi(args[1])),0, IN6ADDR_ANY_INIT,0};
        bind(sock2, (struct sockaddr *) &adr2, sizeof(adr2));
        listen(sock2, 0);

        int sock3 = accept(sock2, NULL, NULL);
        int lu = recv(sock3, buf, BUF_SIZE, 0);
        buf[lu] = 0;
        printf("%s\n", buf);
        int result;
        scanf("%d", &result);
        result = htonl(result);
        send(sock3, &result, sizeof(result), 0);

        close(sock2);   close(sock3);
        return 0;
}
```

```c
//Application 2

#define BUF_SIZE 1024

int main(int argc, char const *argv[]) {
        int sock1 = socket(AF_INET6, SOCK_DGRAM, 0);

        //prepare sock1 pour la reception du 1er message de l'application 1
        prepa_recept(sock1);

        char buf[BUF_SIZE];
        memset(buf, 0, sizeof(buf));
        int lu=read(sock1, buf, BUF_SIZE);

        //retourne une socket preparee pour l'envoi et la reception des messages suivants
        int sock2 = prepa_env_recept(buf, memchr(buf, '\0', lu) +1);

        char calcul[] = "37x45+12";
        int res;
        send(sock2, calcul, strlen(calcul), 0);
        recv(sock2, &res, sizeof(res), 0);
        printf("%d\n", ntohl(res));

        close(sock1);   close(sock2);
        return 0;
}
```

Structures et 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);
int fcntl(int fd, int cmd, ... /* arg */ );
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);
void *memchr(const void s[.n], int c, size_t n);
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 poll(struct pollfd *fds, nfds_t nfds, int timeout);
int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
                        void *(*start_routine) (void *), void *arg);
int pthread_mutex_destroy(pthread_mutex_t *mutex);
int pthread_mutex_init(pthread_mutex_t *restrict mutex,
                            const pthread_mutexattr_t *restrict attr);
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);
```

```
struct sockaddr_in {
  short sin_family;
  unsigned short sin_port;
  struct in_addr sin_addr;
  char sin_zero[8];
}

struct sockaddr_in6 {
  u_int16_t sin6_family;
  u_int16_t sin6_port;
  u_int32_t sin6_flowinfo;
  struct in6_addr  sin6_addr;
  u_int32_t sin6_scope_id;
}

struct pollfd {
  int   fd;
  short events;
  short revents;
}
```

```
struct in_addr {
  unsigned long s_addr;
}

struct in6_addr {
    unsigned char s6_addr[16];
}

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