Institut Galilée — Systèmes d'exploitation, Réseaux
Année 2013-2014 — Licence 2

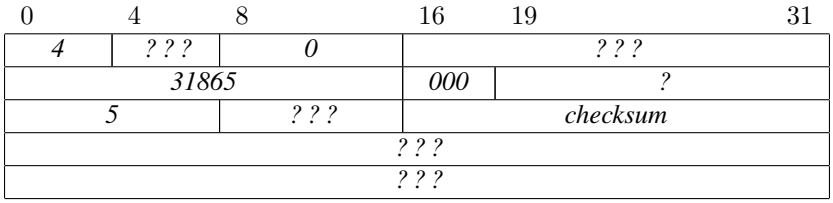
Contrôle Systèmes d'exploitation, Réseaux

Vendredi 23 Mai 2014 — 9h - 12h
Aucun document n'est autorisé

Exercice 1 (4 = 2 + 2)

1. *Donnez dans chacun des cas suivants l'adresse du réseau, du sous réseau, le numéro de la machine pour les configurations suivantes :*
   (a) *Adresse IP = 211.133.33.8 ; masque de réseau = 255.255.255.240*
   (b) *Adresse IP = 24.118.12.12 ; masque de réseau = 255.192.0.0*
2. *Une entreprise s'est vu affecter l'adresse IP 196.179.0.0 pour une gestion plus fine de ses sous réseaux, le responsable informatique désire pouvoir affecter une adresse IP propre à chaque sous réseau des 10 succursales.*
   (a) *De quelle classe s'agit-il ?*
   (b) *Donner et expliquer la valeur du masque de sous réseau correspondant à ce besoin.*
   (c) *Combien de machines chaque sous réseau pourra-t-il comporter et pourquoi ?*
   (d) *Quelle est l'adresse de diffusion (broadcast) du sous réseau 3 ?*

Exercice 2 (3,5 = 3 + 0,5) *Une station X souhaite transmettre à une autre station Y un datagramme UDP dont la taille (incluant l'entête UDP) est égale à 3000 octets. La machine X est dans un réseau A alors que Y est dans un réseau C. L'adresse de X est 154.13.54.6, celle de Y est 223.33.17.45. La MTU du réseau A est égale à 2048 octets. Le datagramme envoyé par X quitte le réseau A en passant par un routeur R1, il atteint le réseau B de MTU = 1024 octets. Il passe ensuite par un routeur R2 pour atteindre le réseau C., dont la MTU est égale à 4096 octets. La structure de l'en-tête IP du datagramme dans le réseau A est présentée ci-dessous :*



1. *Déterminez les paquets IP (en précisant l'en-tête de chaque paquet) qui passent dans les réseaux A, B et C. (La somme de contrôle de l'en-tête n'est pas à calculer)*
2. *Pourquoi le réassemblage de paquets IP n'est-il réalisé que par le destinataire final et pas par un noeud intermédiaire ?*

Exercice 3 (3,5 = 2,5 + 1)
On considère les codes client et serveur donnés respectivement dans les annexes 3 et 4.

1. *Compléter les codes (en rappelant les numéros de ligne).*
2. *Modifier le serveur pour un multiplexage de communication sans scrutation.*

Exercice 4 (7,5 = (1 + 1) + (0,5 + 1 + 1 + 1 + 2))

1. (a) *Donner un exemple simple de système de tâche ne pouvant pas être décrit avec les primitives parbegin/parend.*
   (b) *Rappeler ce qu'est un système de tâches déterminé. Rappeler quand deux tâches sont non interférentes.*
2. *On considère pour la suite de cet exercice le programme utilisant les primitives parbegin/parend donné ci-dessous.*
   (a) *Donner le graphe de flot (on prendra comme tâche chacune des expressions de calcul et les deux opérations de lecture).*

(b) Indiquer les domaines de lecture et écriture des différentes tâches.

(c) Le système de tâches pour le programme ci-dessous est-il de parallélisme maximal ? S'il n'est pas de parallélisme maximal, déterminer le système équivalent de parallélisme maximal.

(d) Donner le programme correspondant de parallélisme maximal en utilisant les primitives de Conway fork/join/quit.

(e) Donner le programme correspondant de parallélisme maximal en C, SOIT avec un processus par tâche et des tubes pour les communications, SOIT avec un thread par tâche et des mutex pour la synchronisation d'exécution (ne mettez pas les fichiers en-tête nécessaires).

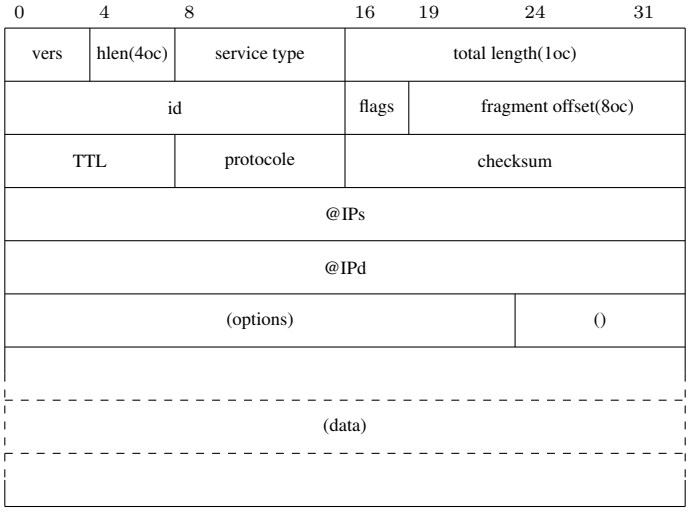
Programme avec primitives parbegin/parend *:*

```
begin
  lire(a) ;
  lire(b) ;
  parbegin
    c := a*b |
    begin
      d := a*a ;
      e := a*b ;
      e := e + b*b ;
    end
  parend ;
  e := c + d + e
end
```

(Annexe 1) Fonctions et structures utiles :

```
pid_t fork(void);
int pipe(int tableau[2]);

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

int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *attr);
int pthread_mutex_destroy(pthread_mutex_t *mutex);

int open(const char *pathname, int flags);
int close(int fd);
ssize_t read(int fd, void *buf, size_t count);q
ssize_t write(int fd, const void *buf, size_t count);
int getchar(void);

int socket(int domaine, int type, int protocole);
int bind(int sock, struct sockaddr * adr, socklen t size);
int listen(int sock, int backlog) ;
int accept(int sock, struct sockaddr *adresse, socklen_t *longueur) ;
int connect(int sock, struct sockaddr *addr serv, socklen t longueur) ;

uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);

int inet aton(const char *cp, struct in_addr *in) ;
char * inet ntoa(struct in_addr in) ;
struct hostent * gethostbyname(char * hostname) ;

struct sockaddr {
   short sa family ;
   char sa data[14] ;
} ;

struct sockaddr in {
   short sin_family ;
   u_short sin_port ;
   struct in_addr sin_addr ;
   char sin_zero[8] ;
} ;

struct hostent {
   char * h_name ;
   char **h_aliases ;
   int h_addrtype ;
   int h_length ;
   char ** h_addr list ;
} ;
```

(Annexe 2) Structure d'un paquet IPv4 :



(Annexe 3) client.c :

```c
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <sys/types.h>
#include <errno.h>
#include <netdb.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <strings.h>
#include <unistd.h>
#include <arpa/inet.h>

#define PORT 7777

void init_client(char *host);
void operation(int);

void erreur(char *message_erreur) {
   perror(message_erreur) ; exit(-1) ;
}

int main(int argc,char *argv[]) {
   if (argc != 2) erreur("usage : client <host>\n") ;
   init_client(argv[1]);
   exit(0);
}

void init_client(char *host) {
   int sock;
   struct sockaddr_in client, serveur;
   struct hostent *hp;
   socklen_t size;

   sock = socket( ???? );
   if (sock < 0) erreur("Erreur de creation de la socket") ;
   bzero(&serveur, sizeof(serveur));
   serveur.sin_family = AF_INET ;
   hp = (struct hostent *) gethostbyname (host);
   if (hp == NULL) erreur("Le nom de serveur n'est pas connu") ;
   bcopy((char *) hp->h_addr_list[0], (char *) &serveur.sin_addr, hp->h_length);
   serveur.sin_port = ???? ;
   printf("Connexion au serveur d'adresse %s et de port %d\n", ???? , ???? );
   if (connect( ???? ) < 0) erreur("Erreur de connexion") ;
   getsockname(sock,(struct sockaddr *)&client,&size);
   printf("Connexion du client d'adresse %s et de port %d\n", ???? , ???? );
   operation(sock);

   close(sock);
}

void operation(int sock) {
   char c;
   do {
      c = getchar();
      if (write(sock, &c, sizeof(char)) <= 0)
         erreur("Erreur d'ecriture dans la socket") ;
   } while (c != 'Z');
   return;
}
```

(Annexe 4) serveur.c :

```c
#include <stdio.h>
#include <fcntl.h>
#include <sys/types.h>
#include <errno.h>
#include <netdb.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <strings.h>
#include <unistd.h>
#include <arpa/inet.h>

#define PORT 7777

void init_serveur(void);
void operation(int);

int main(int argc, char *argv[]) {
   init_serveur();
   return 0;
}

void init_serveur(void) {
   int sock, nsock;
   struct sockaddr_in serveur;
   struct sockaddr_in client;
   socklen_t size;

   sock=socket( ???? );
   bzero(&serveur, sizeof(serveur));
   serveur.sin_family = ???? ;
   serveur.sin_addr.s_addr = INADDR_ANY;
   serveur.sin_port = ???? ;

   bind( ???? );
   listen( ???? , 5);

   size = sizeof(client);
   nsock = accept( ???? );
   printf("Connexion du client d'adresse %s et de port %d\n", ???? , ???? );

   operation(nsock);
   close(nsock);

   return;
}

void operation(int sock) {
   char c;
   do {
      read(sock, &c, sizeof(char));
      printf("%c",c);
   } while (c != 'Z');
   printf("\n");
   return;
}
```