Réseau

Exercice 1 : Élection d'un meneur.

(a) La version réseau du programme, «ring-net.c» instancie un seul nœud par exécution. Vous pouvez recopier votre fonction «protocole» depuis «ring-pipe.c». Le programme prend 3 arguments : le port d'écoute du nœud, le nom du nœud voisin (nom de machine), le port de connexion au nœud voisin. Le programme crée un serveur dans un thread et attend la connexion d'un voisin. En parallèle, dans un autre thread, il tente une connexion sur son voisin (nom de machine et port passés en argument). Écrivez un script mettant en œuvre l'exécution de votre code sur 5 machines du département. Commencez par tester l'exécution de votre code localement (machine localhost en utilisant différents ports). Étendre progressivement la taille de l'anneau.

Solution :

```c
// délai moyen avant reception d'un message
#define DELAY 1.0

// les messages
#define MSG_VOISIN 'v'
#define MSG_PROCHAIN 'p'
#define MSG_GAGNANT 'g'

#include "ring-net.h"

// Comportement d'un nœud.
// Termine avec le numéro du gagnant comme code de sortie.

void protocole (int id)
{
        int actif = 1, nb;

        message("Mon identifiant est %d",id);

        envoyer(MSG_VOISIN,id);        // envoyer son identifiant au voisin

        while (1)
        {
                // accepter un message
                char c; int d;
                recevoir(&c,&d);

                if (!actif)
                {
                        // faire passer le message
                        envoyer(c,d);
```

```
                        if (c == MSG_GAGNANT)
                        {
                                message("nœud %d gagne",d);
                                exit(d);
                        }
                        continue;
                }

                if (c == MSG_VOISIN)
                {
                        nb = d;
                        message("id du voisin = %d",d);
                        envoyer(MSG_PROCHAIN,d);
                }
                else if (c == MSG_PROCHAIN)
                {
                        // évaluer les trois identifiants
                        message("ids = (%d,%d,%d)",id,nb,d);
                        if (d == id && nb >= d)
                        {
                                message("J'ai gagné !!");
                                envoyer(MSG_GAGNANT,id);
                        }
                        else if (nb > id && nb > d)
                        {
                                message("Je reste actif");
                                envoyer(MSG_VOISIN,id);
                        }
                        else
                        {
                                message("Je deviens passif");
                                actif = 0;
                        }
                }
                else if (c == MSG_GAGNANT)
                {
                        if (d != id)
                                message("erreur : quelqu'un a gagné alors "
                                        "que je suis actif?? (%d,%d)",d,id);
                        exit(id);
                }
        }
}

// Les arguments qu'on donne sur la ligne de commande
int sock_server;
char *hostname;
int sock_client;

void* server_thread (void *arg)
{
```

```
        // accepter la connection de notre voisin à gauche
        node.fd_in = accept_connection(sock_server);
        return NULL;
}

void* client_thread (void *arg)
{
        // nous connecter à notre voisin à droite
        node.fd_out = connect_to_server(hostname,sock_client);
        return NULL;
}

int main (int argc, char **argv)
{
        pthread_t t1, t2;

        if (argc != 4)
        {
                fprintf(stderr,"usage: ring <sock1> <host> <sock2>\n");
                exit(1);
        }

        sock_server = atoi(argv[1]);
        hostname = argv[2];
        sock_client = atoi(argv[3]);

        pthread_create(&t1, NULL, server_thread, NULL);
        pthread_create(&t2, NULL, client_thread, NULL);
        pthread_join(t1,NULL);
        pthread_join(t2,NULL);

        // générer et lancer le nœud
        create_node();
        protocole(node.id);
        return 0;
}
```

(b) À partir des ordinateurs de la salle informatique (connectez vous en SSH si vous n'y êtes pas), créez un anneau avec vos voisins et l'étendre à toute la salle de TP.

Exercice 2 : Discussions.

(a) On se propose de coder une application pour discuter entre nous sur le terminal. Il faut donc un programme permettant à la fois de recevoir les messages et de les envoyer. Pour cela, on créera ou bien deux threads, ou bien deux processus. L'un d'eux agira en tant que serveur, et l'autre en tant que client. On pourra se servir des fonctions disponibles dans `http://www.lsv.fr/~hondet/resources/archos/packets.c` et `http://www.lsv.fr/~hondet/resources/archos/packets.h`.

Solution :

```
#include <errno.h>
#include <error.h>
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <ctype.h>
#include <unistd.h>
#include <pthread.h>
#include <time.h>
#include <ulimit.h>
#include <netdb.h>
#include <string.h>

#include "packets.h"

int maxconn = 3;

// Argument of server and client threads
struct arg {
  char* peer; // Peer IP adress
  int port; // Port used to transmit messages
};

// Argument of thread that waits for inputs
struct input_arg {
  int socket; // The socket which receives the packets
  struct sockaddr_in addr; // Adress of the peer sending messages
};

// Each thread recevies messages from a peer, whose adress and socket are given
// in the argument.
void* server_input(struct input_arg* inparg) {
  printf("Input waiting thread started...\n");
  char* who = inet_ntoa((inparg->addr).sin_addr);
  // Wait for input
  // fd_set readfds;
  char* msg;
  for (;;) {
    receive_packet(inparg->socket, (void**) &msg);
    printf("\r<%s> %s\n> ", who, msg);
    free(msg);
  }
  pthread_exit(0);
}

// Blocks until there's a connection on host_socket, sock is filled
// with the peer socket
```

```
void accept_conn(int host_sock, int *sock, struct sockaddr_in* peer_addr) {
  for (;;) {
    socklen_t peer_addr_size;
    *sock = accept(host_sock, (struct sockaddr *) peer_addr,
                   &peer_addr_size);
    if (*sock < 0) {
      if (errno == EINTR || errno == EWOULDBLOCK)
        continue;
      else perror("Can't accept connection");
    }
    printf("%s connectd\n", inet_ntoa(peer_addr->sin_addr));
    break;
  }
}

// Server thread
void* server(struct arg* argsrv) {
  printf("Starting server...\n");
  struct sockaddr_in server_address;

  // Create a TCP socket
  int s;
  if ((s = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) == -1)
    perror("No socket available");

  // connect socket to port
  server_address.sin_family = AF_INET;
  server_address.sin_addr.s_addr = INADDR_ANY;
  server_address.sin_port = htons(argsrv->port);
  if ((bind(s, (struct sockaddr *) &server_address, sizeof(server_address)))
      == -1)
    perror("can't bind port");

  // Socket listens for connections
  if ((listen(s, maxconn)) == -1) perror("listen");

  // Get first connection
  int cl_socket;
  struct sockaddr_in cl_addr;
  accept_conn(s, &cl_socket, &cl_addr);
  pthread_t input_th;
  pthread_create(&input_th, NULL, (void *(*)(void*)) server_input, &cl_socket);

  int cl2_socket;
  struct sockaddr_in cl2_addr;
  accept_conn(s, &cl2_socket, &cl2_addr);
  pthread_t input2_th;
  pthread_create(&input2_th, NULL, (void *(*)(void*)) server_input,
                 &cl2_socket);

  pthread_join(input_th, NULL);
```

```
  pthread_join(input2_th, NULL);
  pthread_exit(0);
}

void* client(struct arg* argcli) {
  printf("Starting client\n");

  // Create tcp socket
  int s = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
  if (s < 0) perror("Client can't connect");

  // Find IP address of host
  struct hostent *peerent = gethostbyname(argcli->peer);
  if (!peerent) perror("Unknown peer");

  // Build address from hostname and port
  struct sockaddr_in peer_address;
  peer_address.sin_family = AF_INET;
  memcpy(&peer_address.sin_addr, peerent->h_addr_list[0],
         peerent->h_length);
  peer_address.sin_port = htons(argcli->port);

  // Connect to peer
  printf("Waiting for peer...\n");
  while (connect(s, (struct sockaddr*) &peer_address,
                 sizeof(peer_address)) == -1)
    ;
  printf("Connected to peer\n");

  fd_set readfds;
  char* msg;
  printf("Client waiting for inputs...\n");
  FD_ZERO(&readfds); // Clear set of fds
  FD_SET(0, &readfds); // Add stdin to fds
  for (;;) {
    // Wait for input
    printf("> "); fflush(stdout);
    if (select(4, &readfds, NULL, NULL, NULL) == -1) continue;
    if (FD_ISSET(0, &readfds)) {
      scanf("%ms", &msg);
      send_string(s, msg);
      free(msg);
    }
  }
  pthread_exit(0);
}

// ./client peer [port] where port is 4001 by default
int main(int argc, char* argv[]) {
  if (argc < 2) error(1, 0, "Not enough args\nUsage: %s PEER [PORT]", argv[0]);
```

```
  char* peer = argv[1];
  int port = argc >= 3 ? atoi(argv[2]): 4001;

  struct arg args = { peer, port};
  pthread_t th_cli, th_srv;
  pthread_create(&th_cli, NULL, (void *(*)(void*)) client, &args);
  pthread_create(&th_srv, NULL, (void *(*)(void*)) server, &args);
  pthread_join(th_cli, NULL);
  pthread_join(th_srv, NULL);
  exit(0);
}
```

(b) Comment faire pour recevoir des messages de plusieurs personnes ? Envoyer des messages à plusieurs personnes ? Implémenter.

Solution :