1 Introduction

Le but de ce projet est la réalisation d'une application Client-Serveur mettant en place un service de jeu "Morpion". Les notions de IPC (Inter Process Communication) et System Calls sont explorés.

2 Description d'architecture

Le programme est écrit en C, en utilisant tous les appels systèmes qui permettent de communiquer avec l'API (Application Programming Interface) des sockets. Un socket est une interface Un grand nombre de systèmes d'exploitation offrent cette interface.

Ces sockets font possible la communication inter processus aussi bien sur une même machine qu'à travers un réseau.

Le port!!!
Un serveur devra offrir la possibilité de connexion à un nombre non-limité de clients. Ces derniers, se limittent à envoyer des rêquetes.
Un stream socket typique est choisie, etc...

2.1 Gestion d'erreurs et exceptions

perror

3 System calls

- **`getaddrinfo()`:** L'usage de cette fonction est optionel, vu que client et serveur sont dans la mâchine et on connaît le protocole, le type de socket et la version IP à utiliser. Néanmoins, ça peut servir pour étendre les fonctionnalités du programme.
- **`socket()`:** Crée un point final de communication et renvoie un descripteur de fichier. Les sockets peuvent avoir de différents types, qui correspondent à la famille de protocoles à utiliser.
- **`bind()`:** Fournie un nom (adresse) à un socket, à l'aide des structures de données précédemment completées (comme `sockaddr`). Avant cet appel, le socket créé avec `socket()` n'a pas de nom formel.
- **`listen()`:** fd
- **`connect()`:** Pour les sockets en mode connecté (`SOCK_STREAM` par example, qui utilise le protocole TCP), cet appel extrait la première connexion de la file d'attente et renvoie un nouveau "file descriptor" qui fait référence à un nouveau socket. Ce descripteur de fichier peut être utilisé pour transmettre information entre les processus, tandis que le socket originel est laissé intact et continue à l'écoute.
- **`accept()`:** asdf
- **`send()`:** Permet de transmettre un message à destination d'une autre socket. L'appel système `write()` est le même sans avoir la possibilité d'ajouter des *flags*.

3.1 Structures de données

Deux structures de donnees doivent être remplies (à la main ou avec les méthodes auxilaires).
`sockaddr_storage` contient l'addresse.

3.2 Procotol??

TCP stream socket?

4 Listing

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

#define PORT "5555"
#define MAXDATASIZE 1000

// get sockaddr, IPv4 or IPv6:
void *get_in_addr(struct sockaddr *sa)
{
  if (sa->sa_family == AF_INET) {
    return &(((struct sockaddr_in*)sa)->sin_addr);
  }
  return &(((struct sockaddr_in6*)sa)->sin6_addr);
}
int main(int argc, char *argv[])
{
  int sockfd, numbytes;
  char buf[MAXDATASIZE];
  struct addrinfo hints, *servinfo, *p;
  int rv;
  char s[INET6_ADDRSTRLEN];
  if (argc != 2) {
    fprintf(stderr,"usage: client hostname\n");
    exit(1);
  }
  memset(&hints, 0, sizeof hints);
  hints.ai_family = AF_UNSPEC;
  hints.ai_socktype = SOCK_STREAM;
  if ((rv = getaddrinfo(argv[1], PORT, &hints, &servinfo)) != 0) {
    fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
    return 1;
  }
  // loop through all the results and connect to the first we can
  for(p = servinfo; p != NULL; p = p->ai_next) {
    if ((sockfd = socket(p->ai_family, p->ai_socktype,
                         p->ai_protocol)) == -1) {
      perror("client: socket");
      continue;
    }
    if (connect(sockfd, p->ai_addr, p->ai_addrlen) == -1) {
      close(sockfd);
      perror("client: connect");
      continue;
```

```
    }
    break;
  }
  if (p == NULL) {
    fprintf(stderr, "client: failed to connect\n");
    return 2;
  }
  inet_ntop(p->ai_family, get_in_addr((struct sockaddr *)p->ai_addr),
            s, sizeof s);
  printf("client: connecting to %s\n", s);
  freeaddrinfo(servinfo); // all done with this structure
  if ((numbytes = recv(sockfd, buf, MAXDATASIZE-1, 0)) == -1) {
    perror("recv");
    exit(1);
  }
  buf[numbytes] = '\0';
  printf("number of bytes received: %i\n", numbytes);
  printf("client: received '%s'\n",buf);
  close(sockfd);
  return 0;
}
```

client.c

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

#define PORT "5555"
#define BACKLOG 5               /* Pending connections the queue will
   hold */
#define SIZE 3

struct addrinfo *getGoodies(void) {
  struct addrinfo *goodies;
  struct addrinfo suggestions;            /* Main data structure */
  memset(&suggestions, 0, sizeof(suggestions));
  suggestions.ai_family = AF_INET6;      /* IPv4 not IPv6 */
  suggestions.ai_protocol = IPPROTO_TCP; /* TCP no raw or UDP */
  suggestions.ai_socktype = SOCK_STREAM; /* TCP connection oriented */

  int ecode;
  if((ecode = getaddrinfo(NULL, PORT, &suggestions, &goodies)) != 0) {
    printf("Failed␣getting␣address␣information:␣%s\n" , gai_strerror(
   ecode));
  } /* status should be 0, not -1 */

  return goodies;
}

int prepareSocket(struct addrinfo* goodies) {
  int sockfd;
  if ((sockfd = socket(goodies -> ai_family,
                       goodies -> ai_socktype,
                       goodies -> ai_protocol)) == -1) {
    perror("Failed␣creating␣socket"); /* and print last error encountered
    */
    exit(EXIT_FAILURE);
  }

  int reuse_addr_to_true = 1;
  if (setsockopt(sockfd, SOL_SOCKET,
                 SO_REUSEADDR, &reuse_addr_to_true, sizeof(int)) == -1) {
    perror("Failed␣setting␣reuse␣address␣option␣for␣the␣socket");
    exit(EXIT_FAILURE);
  }

  if (bind(sockfd, goodies -> ai_addr, goodies -> ai_addrlen) == -1) {
```

```
    close(sockfd);
    perror("Failed to assign addr to socket file descriptor");
    exit(EXIT_FAILURE);
  }

  freeaddrinfo(goodies);        /* No need for this anymore */

  if (listen(sockfd, BACKLOG)) {
    perror("Could not mark the socket referenced by sockfd as passive");
    exit(EXIT_FAILURE);
  }

  return sockfd;

}

void sendPicture(int morpion[SIZE][SIZE], int new_fd) {
  int rows;
  int columns;
  char display[50] = "\n-- Game -- \n";
  for (rows = 0; rows < SIZE; ++rows) {
    strcat(display, "[");
    for (columns = 0; columns < SIZE; ++columns) {
      if (morpion[rows][columns] == 1) {
        strcat(display, " X");
      } else if (morpion[rows][columns] == -1) {
        strcat(display, " 0");
      } else {
        strcat(display, " .");
      }
    }
    strcat(display, " ]\n");
  }

  send(new_fd, display, sizeof display, 0);
}

void playMorpion(int new_fd) {
  int morpion[SIZE][SIZE] = {0};
  morpion[1][1] = 1;
  morpion[2][2] = -1;
  sendPicture(morpion, new_fd);

}


int main(void) {

  struct addrinfo *goodies = getGoodies();
  int sockfd = prepareSocket(goodies); /* Listener */
  struct sockaddr_in6 guest;
  socklen_t sin_size;
```

```
104   printf("Server␣waiting␣for␣connection...\n");
106   while (1) {
        int new_fd;                          /* New file descriptor, where
       stuff happens */
108     sin_size = sizeof guest;
        if ((new_fd = accept(sockfd, (struct sockaddr *)&guest, &sin_size))
       == -1) {
110       perror("Error␣extracting␣connection␣request");
          exit(EXIT_FAILURE);
112     }
114     char from[INET6_ADDRSTRLEN];
        inet_ntop(AF_INET6, &(guest.sin6_addr), from, INET6_ADDRSTRLEN);
116     printf("Got␣connection␣from:␣%s\n", from);
118     if (!fork()) {
          close(sockfd);             /* Close listener, dont need it */
120       char invite[50] = "Envoyer␣le␣chiffre␣1␣pour␣jouer,␣le␣chiffre␣2␣
       pour␣se␣déconnecter";
          send(new_fd, invite, sizeof invite, 0);
122       char buf[100];
          recv(new_fd, buf, sizeof buf, 0);
124       playMorpion(new_fd);
          close(new_fd);
126       exit(EXIT_SUCCESS);
        }
128     close(new_fd);               /* Already forked */
130   }
132
      return 0;
134 }
```

server.c

```
CC = gcc
FLAGS = -Wall -Wextra
FILE = rapport
DIR = tex_files
OPTIONS = -xelatex -output-directory=$(DIR)
PDF_V = evince

compile: client.o server.o
client.o: client.c
	$(CC) $< -o $@

server.o: server.c
	$(CC) $< -o $@

# Run

connect: client.o
	./$< localhost

serve: server.o
	./$<

# LaTeX gen

tex: auto view

mac_tex: auto mac_view

auto: $(FILE).tex
	latexmk $(OPTIONS) $<

view: $(DIR)/$(FILE).pdf
	$(PDF_V) $< &

mac_view: $(DIR)/$(FILE).pdf
	open $< &
```

Makefile