(a) Code client

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT 12345
#define BUF_SIZE 4096

int main(int argc, char **argv)
{
  int c, s, bytes;
  char buf[BUF_SIZE];
  struct hostent *h;
  struct sockaddr_in channel;

  if (argc != 3) fatal("Usage: ");
  h = gethostbyname(argv[1]);
  if (!h) fatal("gethostbyname failed");

  s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
  if (s < 0) fatal("socket");
  memset(&channel, 0, sizeof(channel));
  channel.sin_family= AF_INET;
  memcpy(&channel.sin_addr.s_addr, h->h_addr, h->h_length);
  channel.sin_port= htons(SERVER_PORT);
  c = connect(s, (struct sockaddr *) &channel, sizeof(channel));
  if (c < 0) fatal("connect failed");

  write(s, argv[2], strlen(argv[2])+1);

  while (1) {
        bytes = read(s, buf, BUF_SIZE);
        if (bytes <= 0) exit(0);
        write(1, buf, bytes);
  }
}

fatal(char *string)
{
  printf("%s\n", string);
  exit(1);
}
```

Questions

1. Quel est le service proposé par cette application client/serveur ? Combien d'arguments sont nécessaires au lancement du client ? Quels sont-ils ?

2. Quel port utilise le serveur ? Aurait-on pu choisir une autre valeur ? Quel port utilise le client ? Comment est-il attribué et par quelle primitive ? S'agit-il d'une connexion en mode connecté ou non et est-ce justifié ?

3. A quoi correspondent les constantes BUF_SIZE et QUEUE_SIZE ?

4. Quand est-ce que le serveur s'arrête ? Que fait le serveur une fois les initialisations terminées (décrire le cas où il y a des connexions pendantes et le cas inverse) ?

5. Que se passe t-il si le client est lancé avant que le serveur n'ait démarré ?

6. Quand est-ce que le client s'arrête si la connexion a réussi ? Que fait le client une fois la connexion établie ?

7. Que pensez-vous de la structure actuelle du serveur ? Peut-il satisfaire un grand nombre de connexions ? Expliquez. Proposez une solution plus adaptée.

(b) Code serveur

```
#include <sys/types.h>
#include <sys/fcntl.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT 12345
#define BUF_SIZE 4096
#define QUEUE_SIZE 10

int main(int argc, char *argv[])
{
  int s, b, l, fd, sa, bytes, on = 1;
  char buf[BUF_SIZE];
  struct sockaddr_in channel;

  memset(&channel, 0, sizeof(channel));
  channel.sin_family = AF_INET;
  channel.sin_addr.s_addr = htonl(INADDR_ANY);
  channel.sin_port = htons(SERVER_PORT);

  s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
  if (s < 0) fatal("socket failed");

  b = bind(s, (struct sockaddr *) &channel, sizeof(channel));
  if (b < 0) fatal("bind failed");

  l = listen(s, QUEUE_SIZE);
  if (l < 0) fatal("listen failed");

  while (1) {
        sa = accept(s, 0, 0);
        if (sa < 0) fatal("accept failed");

        read(sa, buf, BUF_SIZE);
        fd = open(buf, O_RDONLY);
        if (fd < 0) fatal("open failed");

        while (1) {
                bytes = read(fd, buf, BUF_SIZE);
                if (bytes <= 0) break;
                write(sa, buf, bytes);
        }
        close(fd);
        close(sa);                      }
}
fatal(char *string)
{
  printf("%s", string);
  exit(1);
  }
```

mini-inetd.c (quelques lignes ont été supprimées pour améliorer la lisibilité du code)

```
#include /* … */

extern char   ** environ;
extern int       getopt(int, char *const*, const char *);
extern int       optind;
extern char    * optarg;

char   * progname          = "mini-inetd";
int      debug             = 0;
int      max_connections   = 0;

/* code de certaines functions supprimées */

int main(int argc, char **argv)
{
    int sd;
    pid_t child;
    int option;
    char *host;
    char *port;

    while ((option = getopt (argc, argv, "dm:")) != EOF)
    {
       switch ((char)option)
       {
       case 'd':
           debug++;
           break;
       case 'm':
           max_connections = atoi(optarg);
           break;
       default:
           usage();
           /* NOTREACHED */
       }
    }

    if (argc - optind < 2)
       usage();

    port = strrchr(argv[optind], ':');
    if (port != NULL) {
       *(port++) = '\0';
       host = argv[optind];
    } else {
       port = argv[optind];
       host = NULL;
    }
    if ((sd = tcp_listen(host, port)) < 0)
       fatal("Can't listen to port %s: %s", argv[optind], strerror(errno));

    while (1)
    {
       struct sockaddr_in them;
       int len = sizeof them;
       int ns = accept(sd, (struct sockaddr*)&them, &len);
       if (ns < 0) {
           if (errno == EINTR)
              continue;
           fatal("Accept failed: %s", strerror(errno));
       }
       switch (child = fork()) {
       case -1:
           perror("Can't fork");
           break;
       case 0:
           /* Child */
           close(sd);
           dup2(ns, 0);
           dup2(ns, 1);
           execve(argv[optind+1], argv+optind+2, environ);
           fatal("Can't start %s: %s", argv[optind+1], strerror(errno));
       default:
           /* Parent */
           if (debug)
              fprintf(stderr, "Forked child %ld\n", (long)child);
           close(ns);
           break;
       }
       if (max_connections > 0  &&  --max_connections <= 0)
           break; /* sortie de la boucle */
    }

    return 0;
}
```

Page man de mini-inetd (sans la description)

```
MINI-INETD(1)

NAME
       mini-inetd - small TCP/IP connection dispatcher

SYNOPSIS
     mini-inetd [-d] [-m maxconnections] [localaddr:]port program [argv0 argv1 ...]

DESCRIPTION

OPTIONS
       -d     Debug...

       -m maxconnections
              Exit after maxconnections connections has been handled.

SEE ALSO
       tcpconnect(1), tcplisten(1), inetd(1m).

BUGS
       The names of the options are not yet finalized, and may change at a future
release.

                                                                        1997 April 13
MINI-INETD(1)
```

Questions

Voici la page man du programme `mini-inetd` ainsi que son code.

1. Complétez la partie `DESCRIPTION` de la page man. Représentez à l'aide d'un schéma/diagramme la structure algorithmique du programme.

2. Dans le code ci-après, le code de la fonction `tcp_listen()` a volontairement été omis. Quelles sont les paramètres et la valeur de retour de cette fonction ? Quelles sont les opérations qui doivent y être réalisées et où les paramètres interviennent-ils ?

3. Commentez le nom du programme. Quelles sont les différences et similitudes entre `mini-inetd` et `inetd` ?

4. Comment modifieriez vous la structure donnée à la question 1 pour que `mini-inetd` puisse traiter plusieurs couples (port, program) passés en arguments ?