UE NSY 103
Programmation Système

EXAMEN 2007-2008

documents autorisés : Non

Objets communicants :

Vous travaillez en collaboration avec l'entreprise « Neo-Object », spécialisée dans la conception d'objets communicants. Cette entreprise cherche de nouveaux débouchés et de nouvelles utilisations de l'informatique nomade. Elle s'intéresse à la conception d'un produit permettant la communication avec des personnes dépendantes (marché des maisons de retraites, hôpitaux, maisons médicalisées). Le système doit permettre d'afficher des messages sur des mini-écrans disséminés dans la pièce (du type cadre photos électroniques). La petite borne centrale qui pilote les écrans via infra-rouge contient le coeur du système, celui chargé de recevoir les messages (via le réseau), et de les diffuser sur les petits écrans (tous les écrans diffusent le même message, envoyé sur le stdout de la borne).

L'ensemble doit être le moins coûteux possible, le plus petit possible, et très peu consommateur d'énergie. Il s'agit d'un petit boîtier comportant un interface réseau sans fil (un peu comme le Wifi, mais moins consommateur d'énergie, tels que les protocoles WiBree, ou ZigBee, par exemple (nota : aucun impact pour le programmeur, le driver est fourni) pour l'accès au réseau, sans clavier écran (si ce n'est les mini-écran en InfraRouge à destination des malades), alimenté par batterie, avec un petit processeur et un peu de mémoire. Afin de réduire au maximum le code embarqué, et toujours dans l'objectif de réduire la consommation au stricte minimum, il a été décidé d'y implanter un noyau Linux, avec un shell très réduit, et très peu de commandes. Ce choix vous oblige donc à écrire des mécanismes du shell qui pourraient manquer.

Vous êtes chargé de concevoir la partie logicielle, qui devra permettre :

- *De récupérer le fichier de message à afficher via le réseau sans fil, à intervalles réguliers (vous utiliserez une connexion TCP)*
- *De traiter son contenu en l'affichant via un système de processus père/fils (le père lit le contenu du fichier et l'envoie au processus fils via un système de tube anonyme)*

Question 1 : Connexion réseau (8 pts)

-1.a) Types [2 points]

Quels sont les différents types de sockets, et quelles sont leurs caractéristiques ?

-1.b) Normalisation des entiers [2 points]

Quel problème peut-on rencontrer en réseau selon les différents CPU impliqués dans l'échange ? Quelles fonctions en C est-on obligé d'utiliser pour pallier ce risque ?

-1.c) Explication du code [2 point]

Expliquer ce que fait le code aux endroits indiqués (ANNEXE 1)

-1.d) Ecriture du code manquant [2 points]

Écrire le code qui récupère simplement le flux fourni par le serveur (ce flux est directement accessible dans le socket, il suffit de l'enregistrer), et l'enregistre dans une fichier /tmp/message (ANNEXE 1)

Question 2 : Matériel (4 pts)

-2.a) Choix de matériel [2 points]

Le fabricant de matériel vous propose une plate-forme matérielle parmi les moins coûteuses possibles (coût et consommation d'énergie). Il y a deux possibilités : un processeur avec une MMU et un autre sans : quel est votre choix ? A quoi sert la MMU ?

-2.b) Explication [2 points]

Décrire les principes de gestion de la mémoire centrale, avec leurs caractéristiques, avantages et inconvénients

Question 3 : Processus (8 pts)

Le système fourni dispose d'un shell très restreint. On voudrait implémenter l'affichage du contenu des messages via un mécanisme utilisant deux processus, un qui lit le fichier, et qui le fourni (via un tube) à un second processus qui sera chargé lui de l'afficher (ceci dans le cas d'éventuelles évolutions du système). Le programme se lance en fournissant le nom du fichier à lire sur la ligne de commande.

-3.a) Communication entre processus [3 points]

En dehors des tubes, quels sont les différents mécanismes permettant à des processus de communiquer (notamment si ceux-ci ne sont pas père et fils) ?

-3.b) Cours [2 points]

Décrire les différents types de tubes.

-3.c) Complétez le code fourni [3 points]

Attention : L'afficheur est limité à une taille de 50 caractères. Il faut faire un retour à la ligne après l'affichage de 50 caractères. (ANNEXE 2)

Annexe 1 : code Connexion réseau

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

#define SERVER_NAME "central" /* serveur ou se connecter         */
#define SERVER_PORT 8090      /* port ou se connecter            */
#define BUFFER_SIZE 10        /* taille du buffer pour la reponse */

int main(int argc,char* argv[]) {
  int                 soc;                /* no de socket          */
  struct sockaddr_in sock_in;             /* infos de connexion    */
  struct hostent   * host;                /* adresse IP du serveur */
  int                ncarlu;              /* nombre d'octets lus   */
  char               buffer[BUFFER_SIZE]; /* buffer                */

  /* Creation de COMMENTEZ*/
  soc = socket(AF_INET, SOCK_STREAM, 0);
  if ( soc == -1 ) {
    perror("socket");
    exit(-1);
  }
  /* Remplissage de COMMENTEZ  */
  bzero(&sock_in, sizeof(sock_in)); /* COMMENTEZ */
  sock_in.sin_family = AF_INET;     /* COMMENTEZ*/
  sock_in.sin_port = htons(SERVER_PORT);  /* COMMENTEZ */


  /* On recupere COMMENTEZ*/
  host = gethostbyname(SERVER_NAME);
  if ( host == NULL ) {
    /* ici, on ne peut PAS utiliser perror car la variable contenant       */
    /* l'erreur n'est PAS errno. Donc il faut gerer les erreurs nous-meme. */
    switch(h_errno) {
    case HOST_NOT_FOUND:
      fprintf(stderr, "gethostbyname : Hote inconnu.\n");
      break;
    case NO_ADDRESS:
      fprintf(stderr, "gethostbyname : Pas d'adresse IP associee a ce nom.\n");
      break;
    case NO_RECOVERY:
      fprintf(stderr, "gethostbyname : Erreur fatale du serveur de noms.\n");
      break;
    case TRY_AGAIN:
      fprintf(stderr, "gethostbyname : Erreur temporaire du serveur de noms.\n");
      break;
    } /* switch */
    exit(-1);
  }
  /* On recopie l'adresse IP du serveur dans la structure */
  bcopy(host->h_addr, &sock_in.sin_addr.s_addr, host->h_length);
  /* Connexion */
  if ( connect(soc, (struct sockaddr *)(&sock_in), sizeof(sock_in)) == -1 ) {
    perror("connect");
    exit(-1);
  }

  /* on crée le fichier et on le remplit */
 ECRIRE LE CODE ICI !
  exit(0);
}
```

Annexe 2 : code lecteur / afficheur

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>


#define BUFFERSIZE 1024
#define MAXREADSIZE 50

int main (int argc,char* argv[]) {
  int fpipe[2]; // les deux files descriptors du pipe
  int fd_in ; //pour lire le fichier
  char buffer[BUFFERSIZE]; //buffer de lecture du fichier
  ssize_t result; //nb de car lus

  if ( argc != 2 ) { /* on a dit que le nom du fichier est fourni sur la ligne de commande */
    fprintf(stderr, "Usage : %s fichier\n", argv[0]);
    exit(-1);
  }


ECRIRE LE CODE ICI !

  exit(0) ;
}
```

Corrigé

1-a)

On s'intéressera ici principalement aux sockets réseau. Les deux principales options sont le mode connecté (TCP, avec mise en place de connexions fiables), et le mode non connecté (UDP, communication via datagrammes).

Dans le cas d'une communication en mode non connecté (UDP), les messages sont envoyés sans accusé de réception , ni séquencement. Cependant, l'exactitude de son contenu est validé par une somme de contrôle. On dit qu'UDP est non fiable.

A contrario, dans TCP (mode connecté), l'échange est considéré comme fiable. La communication s'établit après l'établissement d'une session, et chaque échange est contrôlé (somme de contrôle), acquitté (par accusé de reception) et des numéros de séquences sont utilisés pour ordonner le flux. Cependant, l'échange est un peu plus lourd, ce qui explique la préférence pour UDP dans les échanges simples (DNS..), rapides (jeux, flux vidéo ou audio), ou de proximité...

1-b)

Les architectures des ordinateurs intervenant dans l'échange peuvent différer, notamment sur leurs façons de stocker les mots de plusieurs octets. Dans le cas d'un mot de 32 bits, ceux-ci stokent parfois l'octet de poids fort en fin du mot (on dit que le processeur est **gros-boutiste (big-endian)**), et dans le cas contraire (octet de poids fort stocké en debut), il est appelé **petit-boutiste (little-endian**). Si deux ordinateurs utilisant l'un un petit-boutiste l'autre un gros-boutiste communiquent, il y a fort à parier que l'échange sera altéré par cette différence d'organisation. Il faut donc normaliser l'échange des entiers. On utilisera donc toujours les fonctions *htonl*(), *htons*(), *ntohl*() et *ntohs*() afin de convertir les entiers dans la norme choisie par l'Internet (petit-boutiste).

1-c) et 1-d)

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>

#define SERVER_NAME "localhost" /* serveur ou se connecter         */
#define SERVER_PORT 8090              /* port ou se connecter            */
#define BUFFER_SIZE 10              /* taille du buffer pour la reponse */

int main(int argc,char* argv[]) {
  int                soc;                /* no de socket          */
  struct sockaddr_in sock_in;            /* infos de connexion    */
  struct hostent   * host;               /* adresse IP du serveur */
  int                ncarlu;             /* nombre d'octets lus   */
  char               buffer[BUFFER_SIZE]; /* buffer                */
  int                fd;                 /* fichier en ecriture   */

  /* Creation de la socket (pas de connexion encore) */
  soc = socket(AF_INET, SOCK_STREAM, 0);

  if ( soc == -1 ) {
    perror("socket");
    exit(-1);
  }
  /* Remplissage de la structure permettant d'indiquer ou se connecter */

  bzero(&sock_in, sizeof(sock_in)); /* mise a zero de la structure */
  sock_in.sin_family = AF_INET;     /* Type de connexion */
```

```
  sock_in.sin_port = htons(SERVER_PORT);
  /* port auquel on veut se connecter */
  /* (avec mise des octets dans le bon ordre) */

  /* On recupere l'adresse du serveur auquel on veut se connecter */
  host = gethostbyname(SERVER_NAME);
  if ( host == NULL ) {
    /* ici, on ne peut PAS utiliser perror car la variable contenant       */
    /* l'erreur n'est PAS errno. Donc il faut gerer les erreurs nous-meme. */
    switch(h_errno) {
    case HOST_NOT_FOUND:
      fprintf(stderr, "gethostbyname : Hote inconnu.\n");
      break;
    case NO_ADDRESS:
      fprintf(stderr, "gethostbyname : Pas d'adresse IP associee a ce nom.\n");
      break;
    case NO_RECOVERY:
      fprintf(stderr, "gethostbyname : Erreur fatale du serveur de noms.\n");
      break;
    case TRY_AGAIN:
      fprintf(stderr, "gethostbyname : Erreur temporaire du serveur de noms.\n");
      break;
    } /* switch */
    exit(-1);
  }
  /* On recopie l'adresse IP du serveur dans la structure */
  bcopy(host->h_addr, &sock_in.sin_addr.s_addr, host->h_length);
  /* Connexion */
  if ( connect(soc, (struct sockaddr *)(&sock_in), sizeof(sock_in)) == -1 ) {
    perror("connect");
    exit(-1);
  }

  /* ouverture du fichier */
  fd=open("/tmp/message",O_WRONLY | O_TRUNC | O_CREAT,"0400");
  if (fd==-1) {
     perror("erreur création fichier");
     exit(-1);
  }
  /* lit et affiche la reponse */
  ncarlu = read(soc, buffer, BUFFER_SIZE);
  do {
    if ( write(fd, buffer, ncarlu) <= 0 )
      return 1;
    ncarlu = read(soc, buffer, BUFFER_SIZE);
  } while ( ncarlu > 0 );
  exit(0);
}
```

2-a)

la MMU (Memory Management Unit) est un élément matériel au sien du microprocesseur dont la tâche est de gérer l'accès à la mémoire centrale, notamment en simulant un espace supérieur à l'espace réel. La MMU offre au processeur une vision linéaire d'un espace mémoire logique, et les traduit en accès réel à la RAM, voire à la mémoire virtuelle. Le processeur dispose d'un espace mémoire protégé et cohérent, c'est la MMU qui se charge de gérer réellement les emplacements, qui peuvent être disséminés dans la mémoire physique. De plus, la MMU introduit des mécanismes de sécurité qui détectent les violations d'espace mémoire par les processus (tentative d'accès à une case mémoire en dehors de son espace). La MMU assure aussi le contrôle d'accès aux bus de la RAM, ainsi que la gestion du cache.

2-b)

le processus dispose d'un espace d'adressage, protégé, qui lui est réservé. Il y accède via des adresses logiques, que la MMU traduit en adresses réelles. Cette indirection permet de réorganiser le contenu réel de la RAM sans impacter les processus (il suffit simplement de mettre à jour la table de translation d'adresses pour tenir compte des changements). La mémoire va être découpée en zones, afin d'être mieux gérée, et de pouvoir être réorganisée.

Le découpage peut être fait selon deux philosophies : la pagination ou la segmentation.

- La pagination découpe la mémoire en blocs de taille fixe, les pages. L'intérêt du système est que la réorganisation de la RAM est plus simple à réaliser avec des blocs de taille fixe. L'inconvénient est que ce découpage est arbitraire, et ne correspond pas aux besoins logiques du processus. D'où la fragmentation interne
- La segmentation est une division de la RAM en bloc de taille variable, adaptée aux besoins du processus. La problématique est alors la gestion de la réorganisation, puisqu'avec des blocs de taille variables, il est plus difficile d'optimiser l'occupation, des « trous » risquent d'apparaître, on parle alors de fragmentation externe.

3-a)

Les différents mécanismes qui permettent aux processus de communiquer sont les IPCs (Inter Process Communication). Les trois outils IPCs sont :

les **messages** (MSQ : C'est une communication bidirectionnelle, un peu à la mode du courrier, on dépose et on lit des messages sur une file commune aux différents processus),

la **mémoire partagée** (Shared Memory : contrairement au système normal dans lequel chaque processus est assuré d'être le seul à accéder à l'espace mémoire qui lui appartient, nous avons ici une zone qui est partagée par tous, et où chacun peut (éventuellement) lire et/ou écrire, (attention aux problèmes que cela peut poser...),

et les **sémaphores** (qui permettent le contrôle des accès sur les parties sensibles qui sont partagées, et d'assurer la synchronisation, notamment pour la Shared Memory (on évite ainsi les problèmes d'accès anarchiques à une case donnée, en posant un verrou dessus le temps de la modification, puis en libérant l'accès ensuite)

3-b)

deux types de tubes : anonymes ou nommés. C'est un moyen unidirectionnel de communication entre processus. Le tube simule l'accès à deux fichiers, l'un en écriture et l'autre en lecture.

Un tube anonyme est le dispositif qui permet la connexion du stdout (sortie standard) d'un processus dans le stdin (entrée standard) du suivant.

Un tube nommé (named pipe, ou fifo) est une entrée dans le système de fichier (créé avec la commande mkfifo). N'importe quel processus peut y accéder (selon les droits habituels du système de fichier), même sans lien de parenté avec le créateur. L'écriture dans le named pipe est bloquante, et le lecture aussi. Aucun bloc de donnée n'est affecté à ce named pipe, les échanges ne seront donc pas mémorisés.

3c et 3d)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>


#define BUFFERSIZE 1024
#define MAXREADSIZE 50

int main (int argc,char* argv[]) {
  int fpipe[2]; // les deux files descriptors du pipe
  int fd_in ; //pour lire le fichier
  char buffer[BUFFERSIZE]; //buffer de lecture du fichier
  ssize_t result; //nb de car lus

  if ( argc != 2 ) { /* on a dit que le nom du fichier est fourni sur la ligne de commande */
    fprintf(stderr, "Usage : %s fichier\n", argv[0]);
    exit(-1);
```

```
  }

  if (pipe(fpipe)==-1) { //lancement du pipe
    perror("problème sur le pipe");
    exit(-1);
  }

  switch(fork()) { //On forke, pour créer le père et le fils
  case -1:
    perror("fork");
    exit(-1);
  case 0:  // Le fils :on lit dans le tube
    close(fpipe[1]); //inutile pour nous
     while((result=read(fpipe[0],buffer, MAXREADSIZE )) > 0) { //le lecteur lit sur le pipe,
sur la taille Maxi
       write(STDOUT_FILENO, buffer,result);
       write(STDOUT_FILENO, "\n",1); //le retour à la ligne
    }
    close(fpipe[0]);
    break;
  default:
    fd_in=open(argv[1], O_RDONLY); //ici, le père, qui ouvre le fichier fourni
    if (fd_in == -1 ) {
      perror("open du fichier");
      exit(-1);
    }
    close(fpipe[0]); //inutile pour le père
    while ((result=read(fd_in,buffer,BUFFERSIZE)) > 0) { //lecture du fichier
      if (write(fpipe[1],buffer,result)==-1) { //écriture dans le pipe
       perror("write sur le pipe");
        exit(-1);
      }
    }
    close(fd_in);

  } //fin du switch
  exit(0) ;
}
```