```c
/***** MUTEX - ex2 *****/
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER; // page 14 du manuel C

void thread(unsigned long *cpt){ // on récupère le contenu de la variable dans l'adresse de cpt
    // VARIABLES
    unsigned long tmp; // tampon du compteur
    unsigned long i; // incrémentateur
    // TRAITEMENT
    tmp = 0;
    // compteur
    for(i = 0; i < 10000000; i++){
        pthread_mutex_lock(&mutex); // ON BLOQUE ICI LA VARIABLE
        tmp = *cpt; // *cpt : contenu de la variable à l'adresse de cpt
        tmp ++;
        *cpt = tmp;
        pthread_mutex_unlock(&mutex); // ICI ON DÉBLOQUE LA VARIABLE
    }
    pthread_mutex_destroy(&mutex); // Ici, comme on en a plus besoin, on détruit le mutex
    // afficher le contenu de cpt
    printf("Cpt = %lu\n", *cpt);
}

int main(){
    // VARIABLES
    pthread_attr_t attr; // attributs de création
    pthread_t id1, id2; // identifiant des threads
    unsigned long cpt; // variable compteur

    // TRAITEMENTS
    cpt = 0;
    pthread_attr_init(&attr);
    pthread_create(&id1, &attr, (void*) thread, &cpt); // On passe l'adresse de cpt dans le thread
    pthread_create(&id2, &attr, (void*) thread, &cpt);
    printf("Les 2 threads sont lancés\n");
    pthread_join(id1, NULL);
    pthread_join(id2, NULL);
    exit(EXIT_SUCCESS);
}

/***** PIPES *****/
int main(){
    // VARIABLES
    pid_t id; // pid du fils
    int tube[2]; // tableau à 2 case représentant le tube
    char caractere;// caractère saisi au clavier par l'utilisateur

    // TRAITEMENTS
    setbuf(stdout, NULL); // On débufeurise les sorties standards (on est pas obligé)
    id = fork();

    if(pipe(tube)==-1){ //tentative ouverture du pipe
        printf("erreur\n");
        exit(EXIT_FAILURE);
    }

    switch(id){
    case -1 :
        printf("Il y a une erreur\n");
        exit(EXIT_SUCCESS);
        break;
    case 0: //Processus fils
        close(tube[1]); // fermeture du sens ecriture
        while((read(tube[0],&caractere, sizeof(int))!=-1)){
            // read renvoie -1 si erreur, dans ce cas on arrete la boucle
            if(caractere != '$'){
                printf("Caractère = %c\n", toupper(caractere));
            }else{
                break; // Fin de la boucle
            }
        }
```

```
71          close(tube[0]); // fermeture du sens lecture
72          break;
73      default : //Processus père
74          close(tube[0]); // Fermeture du sens lecture
75          while((caractere = getchar()) != '$'){
76              caractere = getchar(); // Lecture du caractere
77              write(tube[1], &caractere, sizeof(int));
78              if (caractere=='$'){
79                  break;
80              }
81          }
82
83          close(tube[1]); // Fermeture du sens ecriture
84          break;
85      }
86      exit(EXIT_SUCCESS);
87  }
88
89  /***** CLIENT-SERVEUR *****/
90  //Client
91  int main(){
92      // VARIABLES
93      // Déclaration de la socket locale depuis laquelle on se connectera au serveur
94      int fd = socket(AF_INET, SOCK_STREAM, 0); // On initialise socket (fd = File
        descriptor (on déclare un socket))
95      if (fd < 0){
96          printf("Erreur lors de la création du socket\n");
97          exit(EXIT_FAILURE);
98      }
99
100     // Paramétrage de l'adresse à laquelle on souhaite se connecter
101     struct sockaddr_in adresse; // On déclare une structure d'adresse
102     struct hostent *serveur; // Contiendra l'adresse du serveur distant
103     serveur = gethostbyname("iparla.iutbayonne.univ-pau.fr");
104     adresse.sin_family = AF_INET;
105     adresse.sin_port = htons(64100); //htons convertie un entier en "port"
106     adresse.sin_addr = *(struct in_addr *) serveur -> h_addr;
107     bzero((char*)&adresse, sizeof(serveur));
108     //serveur -> h_addr : un membre de la structure hostent.
109     //Contient l'adresse IP du serveur convertie au format réseau et récupéré par la
        fonction gethostbyname
110     //On n'oubliera pas de l'initialiser avec la fonction bzero tel que déclaré
111
112     if(connect(fd, (struct sockaddr *)&adresse, sizeof(adresse)) < 0){
113         //connect renvoie une valeur inférieure à 0 en cas d'erreur
114         printf("connexion impossible\n");
115     }else{
116         printf("Connexion établie\n");
117         int msg = 4;
118         //Envoi du message au serveur
119         write(fd, &msg, sizeof(int));
120         //Lecture du message reçu par le serveur sur le socket
121         read(fd, &msg, sizeof(int));
122         printf("Message reçu : %d\n", msg);
123         // On oublie pas de fermer ce qu'on ouvre
124         close(fd);
125     }
126     exit(EXIT_SUCCESS);
127 }
128
129 //Serveur
130 int main(){
131     int fd; // File descriptor (on déclare un socket)
132     struct sockaddr_in serveur; // On déclare une structure d'adresse pour notre
        serveur
133     // Création du socket
134     fd = socket(AF_INET, SOCK_STREAM, 0);
135     if (fd < 0){
136         printf(("Erreur lors de la création du socket"));
137         exit(EXIT_FAILURE);
138     }
139
140     // Paramétrage de l'adresse serveur (la notre)
```

```
141         serveur.sin_family = AF_INET;
142         serveur.sin_port = htons(9876);
143         serveur.sin_addr.s_addr = INADDR_ANY; //On définit que tlm peut se connecter a
            notre serveur (any adresse in)
144
145         // ici, on tente de lier le socket et le port
146         if((bind(fd, (struct sockaddr *)&serveur, sizeof(struct sockaddr))) < 0){
147             printf("Impossible de lier le socket et le port\n");
148             exit(EXIT_FAILURE);
149         }
150
151         // On attend une connexion
152         listen(fd, 5); // 2eme parametre : nb d'utilisateur simultanés pouvant se
            connecter
153         // remarque: si c'est full et qu'un client veut se connecter, il y aura un
            probleme
154         // coté client, coté serveur c'est bon
155         // On peut maintenant dialoguer avec le client
156         int fa, size;
157         struct sockaddr_in client;
158         size = sizeof(struct sockaddr); // La taille de la structure adresse client et
            serveur
159         while(1){
160             if((fa = accept(fd, (struct sockaddr*)&client, &size)) < 0){
161                 printf("Impossible d'accepter la socket distante\n");
162                 exit(EXIT_FAILURE);
163             }else{
164                 char *message;
165                 read(fa, &message, sizeof(char));
166                 message = toupper(message);
167                 write(fa, &message, sizeof(char));
168             }
169         }
170
171         //Fermeture du socket
172         close(fd);
173         exit(EXIT_SUCCESS);
174     }
175
```