Faculté des Sciences, AMU — Jeudi 10 janvier 2013 – Durée 2h
Licence d'Informatique — UE SIN5U3 – Sites Luminy et Aix

Réseau et Communication

Notes de Cours/TD/TP autorisées ; autres documents, calculettes, ordinateurs interdits.

Vous pouvez appeler sans les recopier les fonctions de la « boîte à outil réseau » vues en TD.

I. Ping d'une liste de machines

La commande `ping` permet de savoir si une machine est visible sur le réseau en utilisant le protocole ICMP. Elle envoie une série de messages `ECHO_REQUEST` à la machine cible et affiche sur la sortie standard des statistiques sur les réponses `ECHO_RESPONSE` obtenues.

On se propose d'écrire en C le programme `multi-ping.c` qui exécute la commande `ping` sur une liste de machines cibles.

1) Écrire la fonction `int rediriger_fd_vers_fichier (int fd, char *nomf)` qui reçoit un descripteur de fichier `fd` ouvert en écriture et un nom de fichier `nomf`. La fonction crée ou écrase le fichier `nomf` avec `open()` puis redirige `fd` vers le fichier `nomf` avec `dup2()`. Elle renvoie 0 en cas de succès, -1 pour erreur.

2) Écrire la fonction `int faire_ping (char *cible, int duree, int silencieux)` où `cible` est l'adresse de la machine à tester, `duree` est la durée de test en secondes, et `silencieux` est vrai ou faux. La fonction crée un fils, puis attend sa terminaison et renvoie le code de terminaison du fils. Le fils se recouvre avec la commande `ping -w duree cible` (l'option `-w` signifie que `ping` arrête les tests au bout de la durée spécifiée). La commande réussit si la machine `cible` est visible sur le réseau. Si `silencieux` est vrai, on redirige la sortie standard de la commande vers `/dev/null`.

3) Écrire le programme principal, dont l'usage est :
`multi-ping [-quiet] duree cible [cible ...]`.
Si les arguments ne sont pas conformes à l'usage attendu, le programme affiche l'usage sur la sortie d'erreur puis échoue. Tous les tests seront faits en mode silencieux si l'option `-quiet` est présente. Pour chaque cible de la liste, le programme fait un ping puis affiche si la cible est visible sur le réseau.

II. Envoi de messages par slots en TCP/IP

On se propose d'écrire en C le programme `para-slot.c` qui envoie des messages en TCP/IP à plusieurs serveurs en parallèle avec un nombre fixé à l'avance de *slots*, c'est-à-dire de connexions simultanées.

On se donne le type `Message` pour mémoriser un buffer à envoyer et l'adresse du serveur destinataire :

```
typedef struct {                /* NE PERDEZ PAS DE TEMPS À RECOPIER CE CODE */
    char *buf, *adr_dest;
    int buf_len, port_dest;
} Message;

void init_message (Message *m, char *buf, char *adr_dest, int port_dest)
{
    m->buf = buf;
    m->buf_len = strlen(buf);
    m->adr_dest = adr_dest;
    m->port_dest = port_dest;
}
```

On se donne le type `Slot` pour associer un message à une connexion par une socket TCP/IP ; on mémorise dans le champ `buf_pos` la position du premier caractère du buffer `mess->buf` qui n'a pas encore été envoyé sur la socket `soc`.

```
typedef struct {                /* NE PERDEZ PAS DE TEMPS À RECOPIER CE CODE */
    Message *mess;
    int soc, buf_pos;
} Slot;

void init_slot (Slot *slot)
{
    slot->mess = NULL;
    slot->soc = -1;    /* Pas de connexion, slot libre */
    slot->buf_pos = 0;
}

void liberer_slot (Slot *slot)
{
    if (slot->soc >= 0) close(slot->soc);
    init_slot (slot);
}
```

1) On suppose disposer de la fonction `int connecter_au_serveur (char *adr_serveur, int port_serveur)` (ne pas l'écrire) qui crée une socket TCP/IP, l'attache à une adresse locale, résout l'adresse du serveur, se connecte au port du serveur, puis renvoie la socket connectée, sinon -1.

Écrire la fonction `int connecter_slot (Slot *slot, Message *mess)` qui mémorise le message `mess` dans le `slot`, se connecte au serveur destinataire du message et mémorise la socket connectée dans le `slot`, initialise le champ `buf_pos` à 0, puis renvoie 0 en cas de succès, -1 sinon.

2) Écrire la fonction `int connecter_slots_libres (Slot slots[], int slot_max, Message messages[], int nb_mess, int *cpt_mess)` qui reçoit un tableau de `slots` de taille `slot_max`, un tableau de `messages` de taille `nb_mess`, ainsi qu'un compteur `*cpt_mess`.

Les messages numérotés 0 à `*cpt_mess-1` sont supposés déjà envoyés, ou en cours d'envoi (dans ce cas ils sont associés à un slot avec une connexion ouverte), ou en échec ; les messages numérotés `*cpt_mess` à `nb_mess-1` sont en attente.

La fonction balaye tous les slots et tente d'associer à chaque slot libre le premier message en attente avec `connecter_slot`. Le compteur `*cpt_mess` est incrémenté à chaque tentative d'association. La fonction renvoie 0 si tout a réussi, -1 s'il y a eu au moins une erreur.

3) Écrire la fonction `int init_select (Slot slots[], int slot_max, fd_set *set)` qui à partir du tableau de `slots` de taille `slot_max` mémorise les sockets connectées dans `set` et renvoie le plus grand descripteur de socket connectée, -1 si aucun.

4) Écrire la fonction `int envoyer_messages (Message messages[], int nb_mess, int slot_max)` qui envoie en parallèle un nombre `nb_mess` de `messages` avec `slot_max` connexions simultanées.

La fonction initialise un tableau de `slots` de taille `slot_max`, puis rentre dans une boucle dans laquelle à chaque itération : elle connecte tous les slots libres, elle scrute en écriture les sockets connectées, puis pour chaque socket éligible, elle envoie la suite du message au serveur correspondant.

Après chaque envoi, on mémorise la prochaine position dans le buffer du message si tout n'a pu être envoyé, sinon on libère le slot (erreur ou tout envoyé).

La fonction renvoie 0 si tous les messages ont été envoyés avec succès, -1 sinon.

Correction

Les fonctions de la « boîte à outil réseau » vues en TD peuvent être appelées sans les recopier.

I. Ping d'une liste de machines

La commande `ping` permet de savoir si une machine est visible sur le réseau en utilisant le protocole ICMP. Elle envoie une série de messages `ECHO_REQUEST` à la machine cible et affiche sur la sortie standard des statistiques sur les réponses `ECHO_RESPONSE` obtenues.

On se propose d'écrire en C le programme `multi-ping.c` qui exécute la commande `ping` sur une liste de machines cibles.

1) Écrire la fonction `int rediriger_fd_vers_fichier (int fd, char *nomf)` qui reçoit un descripteur de fichier `fd` ouvert en écriture et un nom de fichier `nomf`. La fonction crée ou écrase le fichier `nomf` avec `open()` puis redirige `fd` vers le fichier `nomf` avec `dup2()`. Elle renvoie 0 en cas de succès, -1 pour erreur.

```
int rediriger_fd_vers_fichier (int fd, char *nomf)
{
    int f2 = open (nomf, O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (f2 < 0) { perror ("open"); return -1; }
    dup2 (f2, fd); close (f2);
    return 0;
}
```

2) Écrire la fonction `int faire_ping (char *cible, int duree, int silencieux)` où `cible` est l'adresse de la machine à tester, `duree` est la durée de test en secondes, et `silencieux` est vrai ou faux. La fonction crée un fils, puis attend sa terminaison et renvoie le code de terminaison du fils. Le fils se recouvre avec la commande `ping -w duree cible` (l'option `-w` signifie que `ping` arrête les tests au bout de la durée spécifiée). La commande réussit si la machine `cible` est visible sur le réseau. Si `silencieux` est vrai, on redirige la sortie standard de la commande vers `/dev/null`.

```
int faire_ping (char *cible, int duree, int silencieux)
{
    int p, status;

    /* Le processus crée un fils */
    p = fork();
    if (p<0) { perror("fork"); return -1; }

    if (p == 0) { /* Le fils se recouvre avec ping */
        char arg_duree[80];
        sprintf (arg_duree, "%d", duree);

        if (silencieux)
            rediriger_fd_vers_fichier (1, "/dev/null");

        execlp ("ping", "ping", "-w", arg_duree, cible, NULL);
        perror ("exec ping");
        exit (1);
    }

    /* Le père attend la fin du fils et renvoie le code de terminaison */
    wait (&status);
    return WEXITSTATUS (status);
}
```

3) Écrire le programme principal, dont l'usage est :
`multi-ping [-quiet] duree cible [cible ...]`.
Si les arguments ne sont pas conformes à l'usage attendu, le programme affiche l'usage sur la sortie d'erreur puis échoue. Tous les tests seront faits en mode silencieux si l'option `-quiet` est présente. Pour chaque cible de la liste, le programme fait un ping puis affiche si la cible est visible sur le réseau.

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

int main (int argc, char *argv[])
{
    int duree, silencieux, i;

    if (argc < 2+1) {
        fprintf (stderr, "USAGE: %s [-quiet] duree cible [cible ...]\n", argv[0]);
        exit (1);
    }

    silencieux = 0;
    if (strcmp (argv[1], "-quiet") == 0) {
        silencieux = 1; argc--; argv++;
    }

    duree = atoi(argv[1]);
    if (duree <= 0) {
        fprintf (stderr, "ERREUR: mauvaise durée '%s'\n", argv[1]); exit(1);
    }

    for (i = 2; i < argc; i++) {
        char *cible = argv[i];
        int res;
        printf ("ping sur %s ...\n", cible);
        res = faire_ping (cible, duree, silencieux);
        printf ("Résultat du ping : %d %s\n", res, res==0 ? "OK" : "ERREUR");
    }

    exit (0);
}
```

II. Envoi de messages par slots en TCP/IP

On se propose d'écrire en C le programme `para-slot.c` qui envoie des messages en TCP/IP à plusieurs serveurs en parallèle avec un nombre fixé à l'avance de *slots*, c'est-à-dire de connexions simultanées.

On se donne le type `Message` pour mémoriser un buffer à envoyer et l'adresse du serveur destinataire :

```
typedef struct {
    char *buf, *adr_dest;
    int buf_len, port_dest;
} Message;

void init_message (Message *m, char *buf, char *adr_dest, int port_dest)
{
    m->buf = buf;
    m->buf_len = strlen(buf);
    m->adr_dest = adr_dest;
    m->port_dest = port_dest;
}
```

On se donne le type `Slot` pour associer un message à une connexion par une socket TCP/IP ; on mémorise dans le champ `buf_pos` la position du premier caractère du buffer `mess->buf` qui n'a pas encore été envoyé sur la socket `soc`.

```
typedef struct {
    Message *mess;
    int soc, buf_pos;
} Slot;

void init_slot (Slot *slot)
{
    slot->mess = NULL;
    slot->soc = -1;    /* Pas de connexion, slot libre */
    slot->buf_pos = 0;
}

void liberer_slot (Slot *slot)
{
    if (slot->soc >= 0) close(slot->soc);
    init_slot (slot);
}
```

1) On suppose disposer de la fonction `int connecter_au_serveur (char *adr_serveur, int port_serveur)` (ne pas l'écrire) qui crée une socket TCP/IP, l'attache à une adresse locale, résout l'adresse du serveur, se connecte au port du serveur, puis renvoie la socket connectée, sinon -1.

Écrire la fonction `int connecter_slot (Slot *slot, Message *mess)` qui mémorise le message `mess` dans le `slot`, se connecte au serveur destinataire du message et mémorise la socket connectée dans le `slot`, initialise le champ `buf_pos` à 0, puis renvoie 0 en cas de succès, -1 sinon.

```
int connecter_slot (Slot *slot, Message *mess)
{
    slot->mess = mess;
    slot->soc = connecter_au_serveur (mess->adr_dest, mess->port_dest);
    slot->buf_pos = 0;
    return slot->soc < 0 ? -1 : 0;
}
```

```
int connecter_au_serveur (char *adr_serveur, int port_serveur)
{
    int soc;
    struct sockaddr_in adrL, adrS;
    struct hostent *hp;

    /* Création d'une socket domaine internet et mode connecté */
    soc = socket (AF_INET, SOCK_STREAM, 0);
    if (soc < 0) { perror ("socket ip"); return -1; }

    /* Fabrication adresse locale */
    adrL.sin_family = AF_INET;
    adrL.sin_port = htons (0);
    adrL.sin_addr.s_addr = htonl(INADDR_ANY);

    /* Attachement socket à l'adresse locale */
    if (bor_bind_in (soc, &adrL) == -1)
      { close (soc); return -1; }

    /* Fabrication adresse du serveur */
    adrS.sin_family = AF_INET;
    adrS.sin_port = htons (port_serveur);
    printf ("Résolution %s ...\n", adr_serveur);
```

```
    if ((hp = gethostbyname (adr_serveur)) == NULL)
      { herror ("gethostbyname ip"); close (soc); return -1; }
    memcpy (&adrS.sin_addr.s_addr, hp->h_addr, hp->h_length);

    printf ("Connexion à %s:%d ...\n", adr_serveur, port_serveur);
    if (bor_connect_in (soc, &adrS) < 0)
      { close (soc); return -1; }
    printf ("Connexion établie\n");

    return soc;
}
```

2) Écrire la fonction `int connecter_slots_libres (Slot slots[], int slot_max, Message messages[], int nb_mess, int *cpt_mess)` qui reçoit un tableau de `slots` de taille `slot_max`, un tableau de `messages` de taille `nb_mess`, ainsi qu'un compteur `*cpt_mess`.

Les messages numérotés 0 à `*cpt_mess-1` sont supposés déjà envoyés, ou en cours d'envoi (dans ce cas ils sont associés à un slot avec une connexion ouverte), ou en échec ; les messages numérotés `*cpt_mess` à `nb_mess-1` sont en attente.

La fonction balaye tous les slots et tente d'associer à chaque slot libre le premier message en attente avec `connecter_slot`. Le compteur `*cpt_mess` est incrémenté à chaque tentative d'association. La fonction renvoie 0 si tout a réussi, -1 s'il y a eu au moins une erreur.

```
int connecter_slots_libres (Slot slots[], int slot_max,
    Message messages[], int nb_mess, int *cpt_mess)
{
    int i, res = 0;

    for (i = 0; i < slot_max && *cpt_mess < nb_mess; i++) {
        if (slots[i].soc != -1) continue;
        printf ("Tentative connection slot %d ...\n", i);
        if (connecter_slot (&slots[i], &messages[*cpt_mess]) < 0) {
            i--; res = -1;
        }
        *cpt_mess += 1;
    }
    return res;
}
```

3) Écrire la fonction `int init_select (Slot slots[], int slot_max, fd_set *set)` qui à partir du tableau de `slots` de taille `slot_max` mémorise les sockets connectées dans `set` et renvoie le plus grand descripteur de socket connectée, -1 si aucun.

```
int init_select (Slot slots[], int slot_max, fd_set *set)
{
    int i, maxfd = -1;

    FD_ZERO (set);
    for (i = 0; i < slot_max; i++) {
        if (slots[i].soc == -1) continue;
        FD_SET (slots[i].soc, set);
        if (maxfd < slots[i].soc) maxfd = slots[i].soc;
    }

    return maxfd;
}
```

4) Écrire la fonction `int envoyer_messages (Message messages[], int nb_mess, int slot_max)` qui envoie en parallèle un nombre `nb_mess` de `messages` avec `slot_max` connexions simultanées.

La fonction initialise un tableau de `slots` de taille `slot_max`, puis rentre dans une boucle dans laquelle à chaque itération : elle connecte tous les slots libres, elle scrute en écriture les sockets connectées, puis pour chaque socket éligible, elle envoie la suite du message au serveur correspondant.

Après chaque envoi, on mémorise la prochaine position dans le buffer du message si tout n'a pu être envoyé, sinon on libère le slot (erreur ou tout envoyé).

La fonction renvoie 0 si tous les messages ont été envoyés avec succès, -1 sinon.

```
#define SLOT_MAX 4

int envoyer_messages (Message messages[], int nb_mess, int slot_max)
{
    Slot slots[SLOT_MAX];
    int i, k, cpt_mess = 0, tout_ok = 1, maxfd, res, n;
    fd_set set_w;

    /* Init des slots */
    if (slot_max > SLOT_MAX) slot_max = SLOT_MAX;
    for (i = 0; i < slot_max; i++) init_slot (&slots[i]);

    while (1) {
        if (connecter_slots_libres (slots, slot_max, messages, nb_mess, &cpt_mess) < 0)
            tout_ok = 0;

        maxfd = init_select (slots, slot_max, &set_w);
        if (maxfd < 0) break; /* Sortie principale du while */

        res = select (maxfd+1, NULL, &set_w, NULL, NULL);
        if (res < 0) {
            perror ("select");
            for (i = 0; i < slot_max; i++) liberer_slot (&slots[i]);
            return -1;
        }

        for (i = 0; i < slot_max; i++) {
            if (slots[i].soc == -1 || !FD_ISSET (slots[i].soc, &set_w)) continue;

            printf ("Envoi pour slot %d pos %d\n", i, slots[i].buf_pos);
            n = slots[i].mess->buf_len - slots[i].buf_pos;
            if (n > 10) n = 10; /* pour tests */
            k = write (slots[i].soc, slots[i].mess->buf + slots[i].buf_pos, n);
            if (k < 0) { perror ("write"); tout_ok = 0; }
            else slots[i].buf_pos += k;

            if (k < 0 || slots[i].buf_pos >= slots[i].mess->buf_len) {
                printf ("Libération slot %d\n", i);
                liberer_slot (&slots[i]);
            }
        }
    }

    return tout_ok ? 0 : -1;
}
```

Voici un exemple de programme principal :

```
#include "bor-util.h"
#define MESS_MAX 100

int main (int argc, char *argv[])
{
    Message messages[MESS_MAX];
    int i, k, mess_nb = 0;

    if (argc <= 1+1) {
        fprintf (stderr, "USAGE: %s {buf adr port}...\n", argv[0]);
        exit (1);
    }

    for (i = 1; i+2 < argc && mess_nb < MESS_MAX ; i+=3) {
        init_message (&messages[mess_nb], argv[i], argv[i+1], atoi(argv[i+2]));
        mess_nb++;
    }

    printf ("Envoi de %d messages sur %d slots\n",
        mess_nb, SLOT_MAX);

    k = envoyer_messages (messages, mess_nb, SLOT_MAX);

    if (k < 0) printf ("Il y a eu des échecs\n");
    else printf ("Tout envoyé avec succès\n");
    exit (k < 0 ? 1 : 0);
}
```

Voici un exemple de script pour tester :

```
#! /bin/bash

s=("premier message" "second message" "troisième message" \
   "quatre" "cinq message plus long que d habitude" "sixième étage" \
   "sept" "huit est le cube de deux" "neuf est le carré de trois")

p=31000
n=${#s[*]}
a=

for ((i=0; i<n; i++)); do
    xterm -e netcat -kl $((p+i)) &
    a="$a '${s[$i]}' localhost $((p+i))"
done

echo "on attend que tous les netcat soient initialisés ..."
sleep 3

echo ./janv13-para-slot $a
eval ./janv13-para-slot $a

echo "Pour fermer les xterm tapez Entrer"
read ligne
killall netcat
exit 0
```