

RÉSEAU ET COMMUNICATION

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

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

I. Client-serveur de vote à jeton en UDP/IP

On se propose de réaliser en C un client-serveur permettant d'effectuer un vote unique, et qui fonctionne selon le principe suivant. Le serveur dispose de la liste des emails des électeurs, et pour chacun d'entre eux, tire au sort un *jeton* (une chaîne alphanumérique unique). Le vote s'effectue alors en deux temps. Primo, un électeur utilise le programme client pour envoyer son email au serveur; celui-ci acquitte le message et envoie le jeton par courrier électronique. Secundo, l'électeur envoie à l'aide du programme client son jeton et son vote au serveur, qui acquitte le message et mémorise le vote. Le détail des opérations est décrit dans la suite.

1) On s'intéresse d'abord à la réalisation du client, pour lequel on définit le type suivant :

```
typedef struct {  
    enum { A_NONDEF, A_RECJETON, A_VOTER } action;  
    char *email, *jeton, *choix;  
} Dialogue;
```

Ce type permet de mémoriser l'action à réaliser lors d'un dialogue avec le serveur. Par défaut elle est non définie (A_NONDEF). Lors de la première étape du dialogue, on demande à recevoir le jeton (A_RECJETON); le champ `email` contient l'adresse de courrier électronique de l'électeur. Lors de la seconde étape, on demande à voter (A_VOTER); le champ `jeton` contient le jeton de vote, et le champ `choix` contient le vote proprement dit (par exemple "oui" ou "non"). Pour simplifier, on suppose qu'aucun champ ne contient de blancs.

Écrire la fonction `int dialoguer_avec_serveur (int soc, struct sockaddr_in *adrS, Dialogue *dia)` recevant en paramètres une socket UDP/IP `soc`, l'adresse valide `adrS` d'un serveur de vote, et les paramètres du dialogue `dia` à effectuer avec le serveur.

Selon l'action demandée, la fonction envoie au serveur la chaîne "GETJ_□*email*" pour obtenir le jeton, ou la chaîne "VOTE_□*jeton*_□*choix*" pour effectuer le vote.

La fonction attend ensuite la réponse du serveur puis l'affiche sur la sortie standard. Si au bout de 5 secondes la réponse n'est pas arrivée, la fonction affiche "délai dépassé" et se termine. Si la réponse ne vient pas du serveur mais d'une autre provenance, la fonction affiche "mauvaise provenance de la réponse" et se termine. La fonction renvoie 0 en cas de succès, -1 dans tous les cas d'erreur.

2) On suppose disposer de la fonction `int creer_socket_udpip (int nport, struct sockaddr_in *adrL)` qui crée une socket UDP/IP, l'attache au port `nport` (0 pour attribution d'un port libre), mémorise l'adresse de cette socket dans `adrL`, puis renvoie la socket. En cas d'erreur, la fonction ferme la socket si besoin et renvoie -1. *Ne pas écrire cette fonction.*

De plus, on suppose disposer de la fonction `int fabriquer_adr_serveur_ip (int nport, char *nom, struct sockaddr_in *adrS)` qui reçoit un numéro de port `nport` et le nom d'un serveur, puis résout l'adresse et remplit les champs de `adrS`. *Ne pas écrire cette fonction.*

Écrire la fonction `main` du client. Lors de la première étape du vote, les arguments exigés sont : *serveur port -g email*. Lors de la seconde étape, les arguments exigés sont : *serveur port -v jeton choix*. En cas d'erreur sur les arguments, le programme affiche l'usage et échoue.

Le programme crée ensuite une socket UDP/IP, fabrique l'adresse du serveur, puis fait un dialogue avec celui-ci, avant de fermer la socket. Le programme se termine avec succès si le dialogue a réussi, sinon il échoue.

3) On se consacre dorénavant à la réalisation du serveur. On suppose que l'on dispose dans le répertoire courant du programme `mai11-email` (voir examen de Mai 2011) exigeant les arguments *serveur port client auteur destinataire sujet corps*, qui sert à envoyer selon le protocole SMTP un courrier électronique en se connectant au *port* d'un *serveur*.

Écrire la fonction `int envoyer_jeton (char *email, char *jeton)` recevant en paramètres une adresse de courrier électronique `email` et un `jeton`. La fonction crée un fils qui se recouvre avec `mai11-email` pour envoyer un courrier électronique à l'adresse `email` en se connectant au port 25 local; le sujet est "Jeton de vote" et le corps du message est "`jeton_``jeton`"; l'adresse de l'expéditeur est laissée à votre convenance.

Le père attend la fin du fils et renvoie : 0 si le fils a réussi, -1 dans tous les cas d'erreur.

4) Le serveur doit pouvoir stocker les emails, les jetons et les votes ainsi que leurs relations, par exemple à l'aide d'une base de données ou de fichiers. Pour simplifier, on suppose disposer des fonctions `chercher_jeton` et `effectuer_vote` pour accéder à la base.

La fonction `int chercher_jeton (char *email, char *jeton)` reçoit en paramètres un `email` et un buffer `jeton` de taille suffisante. Elle cherche dans la base le jeton correspondant au `email` et le copie dans `jeton`. Elle renvoie 0 en cas de succès, -1 si le email n'est pas dans la base, -2 en cas d'erreur interne (par exemple échec de connexion à la base). *Ne pas écrire cette fonction.*

La fonction `int effectuer_vote (char *jeton, char *vote)` reçoit en paramètres un `jeton` et un `vote`. Elle cherche dans la base le `jeton` et enregistre le `vote` correspondant si celui-ci n'a pas déjà eu lieu. Elle renvoie 0 en cas de succès, -1 si le jeton est mauvais, -2 s'il y a déjà un vote pour ce jeton, -3 en cas d'erreur interne. *Ne pas écrire cette fonction.*

Écrire la fonction `void traitement_requete (char *requete, char *reponse, int rep_len)` recevant en paramètres une `requete` provenant du client, et un buffer `reponse` de taille `rep_len`. La fonction analyse la requête, réagit en conséquence et écrit une réponse dans le buffer `reponse`.

Si la requête est de la forme "`GETJ_``email`", la fonction cherche le jeton correspondant au `email` puis envoie le jeton à l'adresse `email` par courrier électronique. Si la requête est "`VOTE_``jeton_``choix`", la fonction essaie d'effectuer le vote `choix` pour le `jeton`.

Dans tous les cas de figure, la fonction doit écrire un message approprié dans `reponse`. Voici la liste des messages possibles :

```
"200 jeton envoyé par mail"
"210 vote ok"
"300 email inconnu"
"310 échec envoi jeton par mail"
"320 jeton inconnu"
"330 jeton a déjà voté"
"400 mauvaise syntaxe"
"500 erreur interne"
```

5) Écrire la fonction `int dialoguer_avec_un_client (int soc)` recevant en paramètre une socket UDP/IP `soc`. La fonction lit une requête dans la socket, traite la requête puis envoie la réponse dans la socket. Elle renvoie 0 en cas de succès, -1 en cas d'erreur.

La fonction `main` consistera à créer une socket UDP/IP, boucler sur le dialogue avec un client, puis fermer la socket. *Ne pas écrire cette fonction.*

Correction

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

I. Client-serveur de vote à jeton en UDP/IP

On se propose de réaliser en C un client-serveur permettant d'effectuer un vote unique, et qui fonctionne selon le principe suivant. Le serveur dispose de la liste des emails des électeurs, et pour chacun d'entre eux, tire au sort un *jeton* (une chaîne alphanumérique unique). Le vote s'effectue alors en deux temps. Primo, un électeur utilise le programme client pour envoyer son email au serveur ; celui-ci acquitte le message et envoie le jeton par courrier électronique. Secundo, l'électeur envoie à l'aide du programme client son jeton et son vote au serveur, qui acquitte le message et mémorise le vote. Le détail des opérations est décrit dans la suite.

1) On s'intéresse d'abord à la réalisation du client, pour lequel on définit le type suivant :

```
typedef struct {
    enum { A_NONDEF, A_RECJETON, A_VOTER } action;
    char *email, *jeton, *choix;
} Dialogue;
```

Ce type permet de mémoriser l'action à réaliser lors d'un dialogue avec le serveur. Par défaut elle est non définie (A_NONDEF). Lors de la première étape du dialogue, on demande à recevoir le jeton (A_RECJETON) ; le champ `email` contient l'adresse de courrier électronique de l'électeur. Lors de la seconde étape, on demande à voter (A_VOTER) ; le champ `jeton` contient le jeton de vote, et le champ `choix` contient le vote proprement dit (par exemple "oui" ou "non"). Pour simplifier, on suppose qu'aucun champ ne contient de blancs.

Écrire la fonction `int dialoguer_avec_serveur (int soc, struct sockaddr_in *adrS, Dialogue *dia)` recevant en paramètres une socket UDP/IP `soc`, l'adresse valide `adrS` d'un serveur de vote, et les paramètres du dialogue `dia` à effectuer avec le serveur.

Selon l'action demandée, la fonction envoie au serveur la chaîne "GETJ_□*email*" pour obtenir le jeton, ou la chaîne "VOTE_□*jeton*_□*choix*" pour effectuer le vote.

La fonction attend ensuite la réponse du serveur puis l'affiche sur la sortie standard. Si au bout de 5 secondes la réponse n'est pas arrivée, la fonction affiche "délai dépassé" et se termine. Si la réponse ne vient pas du serveur mais d'une autre provenance, la fonction affiche "mauvaise provenance de la réponse" et se termine. La fonction renvoie 0 en cas de succès, -1 dans tous les cas d'erreur.

```
int dialoguer_avec_serveur (int soc, struct sockaddr_in *adrS, Dialogue *dia)
{
    char buf[1500];
    int k, res;
    struct sockaddr_in adr_tmp;
    fd_set set;
    struct timeval tm;

    switch (dia->action) {
        case A_RECJETON : sprintf (buf, "GETJ %s", dia->email); break;
        case A_VOTER    : sprintf (buf, "VOTE %s %s", dia->jeton, dia->choix); break;
        default : fprintf (stderr, "ERREUR action non définie\n"); return -1;
    }

    printf ("Envoi de : %s\n", buf);
    k = bor_sendto_in (soc, buf, strlen (buf), adrS);
    if (k < 0) return -1;

    printf ("Attente réponse ... \n");
    FD_ZERO (&set);
```

```

FD_SET (soc, &set);
tm.tv_sec = 5;
tm.tv_usec = 0;
res = select (soc+1, &set, NULL, NULL, &tm);
if (res < 0) { perror ("select"); return -1; }
if (res == 0) { fprintf (stderr, "ERREUR: délai dépassé\n"); return -1; }

k = bor_recvfrom_in (soc, buf, sizeof(buf)-1, &adr_tmp);
if (k < 0) return -1;
buf[k] = 0;

if (adr_tmp.sin_port != adrS->sin_port ||
    adr_tmp.sin_addr.s_addr != adrS->sin_addr.s_addr) {
    fprintf (stderr, "ERREUR: mauvaise provenance de la réponse\n");
    return -1;
}

printf ("Reçu : %s\n", buf);
return 0;
}

```

2) On suppose disposer de la fonction `int creer_socket_udpip (int nport, struct sockaddr_in *adrL)` qui crée une socket UDP/IP, l'attache au port `nport` (0 pour attribution d'un port libre), mémorise l'adresse de cette socket dans `adrL`, puis renvoie la socket. En cas d'erreur, la fonction ferme la socket si besoin et renvoie -1. *Ne pas écrire cette fonction.*

De plus, on suppose disposer de la fonction `int fabriquer_adr_serveur_ip (int nport, char *nom, struct sockaddr_in *adrS)` qui reçoit un numéro de port `nport` et le nom d'un serveur, puis résout l'adresse et remplit les champs de `adrS`. *Ne pas écrire cette fonction.*

Écrire la fonction `main` du client. Lors de la première étape du vote, les arguments exigés sont : *serveur port -g email*. Lors de la seconde étape, les arguments exigés sont : *serveur port -v jeton choix*. En cas d'erreur sur les arguments, le programme affiche l'usage et échoue.

Le programme crée ensuite une socket UDP/IP, fabrique l'adresse du serveur, puis fait un dialogue avec celui-ci, avant de fermer la socket. Le programme se termine avec succès si le dialogue a réussi, sinon il échoue.

```

int main (int argc, char *argv[])
{
    struct sockaddr_in adrC, adrS;
    int soc, nport, res;
    char *adrS_nom;
    Dialogue dia;

    dia.action = A_NONDEF;
    if (argc == 5 && strcmp (argv[3], "-g") == 0) {
        dia.action = A_RECJETON;
        dia.email = argv[4];
    } else if (argc == 6 && strcmp (argv[3], "-v") == 0) {
        dia.action = A_VOTER;
        dia.jeton = argv[4];
        dia.choix = argv[5];
    } else {
        fprintf (stderr, "USAGE: %s serveur port {-g email}|{-v jeton choix}\n",
                argv[0]);
        exit (1);
    }
    adrS_nom = argv[1];
    nport = atoi (argv[2]);

    soc = creer_socket_udpip (0, &adrC);
    if (soc < 0) exit (1);
}

```

```

    if (fabriquer_adr_serveur_ip (nport, adrS_nom, &adrS) < 0)
        { close (soc); exit (1); }

    res = dialoguer_avec_serveur (soc, &adrS, &dia);

    close (soc);
    return res == 0 ? 0 : 1;
}

```

Pour être complet, voici le code des fonctions manquantes :

```

int creer_socket_udpip (int nport, struct sockaddr_in *adrL)
{
    int soc;

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

    /* Fabrication adresse locale */
    adrL->sin_family = AF_INET;
    adrL->sin_port = htons (nport);          /* 0 pour attribution d'un port libre */
    adrL->sin_addr.s_addr = htonl(INADDR_ANY); /* Toutes les adr. locales */

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

    /* Récupération de l'adresse réelle et du port sous forme Network */
    if (bor_getsockname_in (soc, adrL) < 0)
        { close (soc); exit(1); }
    printf ("port %d ouvert\n", ntohs(adrL->sin_port));

    printf ("Socket locale créée\n");
    return soc;
}

int fabriquer_adr_serveur_ip (int nport, char *nom, struct sockaddr_in *adrS)
{
    struct hostent *hp;

    adrS->sin_family = AF_INET;
    adrS->sin_port = htons (nport); /* forme Network */
    printf ("Résolution adr serveur ...\n");
    if ((hp = gethostbyname (nom)) == NULL) /* h_errno, perror() */
        { perror ("gethostbyname ip"); return -1; }
    memcpy (&adrS->sin_addr.s_addr, hp->h_addr, hp->h_length);

    return 0;
}

```

3) On se consacre dorénavant à la réalisation du serveur. On suppose que l'on dispose dans le répertoire courant du programme `mai11-email` (voir examen de Mai 2011) exigeant les arguments *serveur port client auteur destinataire sujet corps*, qui sert à envoyer selon le protocole SMTP un courrier électronique en se connectant au *port* d'un *serveur*.

Écrire la fonction `int envoyer_jeton (char *email, char *jeton)` recevant en paramètres une adresse de courrier électronique `email` et un `jeton`. La fonction crée un fils qui se recouvre avec `mai11-email` pour envoyer un courrier électronique à l'adresse `email` en se connectant au port 25 local; le sujet est "Jeton de vote" et le corps du message est "jeton_␣jeton"; l'adresse de l'expéditeur est laissée à votre convenance.

Le père attend la fin du fils et renvoie : 0 si le fils a réussi, -1 dans tous les cas d'erreur.

```

int envoyer_jeton (char *email, char *jeton)
{
    int p, status, res;

    p = fork ();
    if (p < 0) { perror ("fork"); return -1; }

    if (p == 0) { /* Fils */

        char corps[1024];
        sprintf (corps, "jeton %s\n", jeton);

        execlp ("../mail1-email", "mail1-email", "localhost", "25", "localhost",
                "noreply@nowhere.fr", email, "Jeton de vote", corps, NULL);
        perror ("execlp");
        exit (1);
    }

    /* Suite du père */
    printf ("Création d'un fils ...\n");
    wait (&status);
    res = WEXITSTATUS (status);

    printf ("Résultat fils : %s\n", res == 0 ? "succès" : "échec");
    return res == 0 ? 0 : -1;
}

```

4) Le serveur doit pouvoir stocker les emails, les jetons et les votes ainsi que leurs relations, par exemple à l'aide d'une base de données ou de fichiers. Pour simplifier, on suppose disposer des fonctions `chercher_jeton` et `effectuer_vote` pour accéder à la base.

La fonction `int chercher_jeton (char *email, char *jeton)` reçoit en paramètres un `email` et un buffer `jeton` de taille suffisante. Elle cherche dans la base le jeton correspondant au `email` et le copie dans `jeton`. Elle renvoie 0 en cas de succès, -1 si le email n'est pas dans la base, -2 en cas d'erreur interne (par exemple échec de connexion à la base). *Ne pas écrire cette fonction.*

La fonction `int effectuer_vote (char *jeton, char *vote)` reçoit en paramètres un `jeton` et un `vote`. Elle cherche dans la base le `jeton` et enregistre le `vote` correspondant si celui-ci n'a pas déjà eu lieu. Elle renvoie 0 en cas de succès, -1 si le jeton est mauvais, -2 s'il y a déjà un vote pour ce jeton, -3 en cas d'erreur interne. *Ne pas écrire cette fonction.*

Écrire la fonction `void traitement_requete (char *requete, char *reponse, int rep_len)` recevant en paramètres une `requete` provenant du client, et un buffer `reponse` de taille `rep_len`. La fonction analyse la requête, réagit en conséquence et écrit une réponse dans le buffer `reponse`.

Si la requête est de la forme `"GETJ_email"`, la fonction cherche le jeton correspondant au `email` puis envoie le jeton à l'adresse `email` par courrier électronique. Si la requête est `"VOTE_jeton_choix"`, la fonction essaie d'effectuer le vote `choix` pour le `jeton`.

Dans tous les cas de figure, la fonction doit écrire un message approprié dans `reponse`. Voici la liste des messages possibles :

<pre> "200 jeton envoyé par mail" "210 vote ok" "300 email inconnu" "310 échec envoi jeton par mail" "320 jeton inconnu" "330 jeton a déjà voté" "400 mauvaise syntaxe" "500 erreur interne" </pre>

```

void traitement_requete (char *requete, char *reponse, int rep_len)
{
    char mot1[1024];
    int pos;

    (void) rep_len;

    if (sscanf (requete, "%s%n", mot1, &pos) != 1)
        { sprintf (reponse, "400 mauvaise syntaxe"); return; }

    if (strcmp (mot1, "GETJ") == 0) {

        char email[1024], jeton[1024];
        int res;

        if (sscanf (requete+pos, "%s", email) != 1)
            { sprintf (reponse, "400 mauvaise syntaxe"); return; }

        res = chercher_jeton (email, jeton);
        if (res == 0) {
            if (envoyer_jeton (email, jeton) == 0)
                strcpy (reponse, "200 jeton envoyé par mail");
            else strcpy (reponse, "310 échec envoi jeton par mail");
        } else if (res == -1) strcpy (reponse, "300 email inconnu");
        else strcpy (reponse, "500 erreur interne");

    } else if (strcmp (mot1, "VOTE") == 0) {

        char jeton[1024], choix[1024];
        int res;

        if (sscanf (requete+pos, "%s %s", jeton, choix) != 2)
            { sprintf (reponse, "400 mauvaise syntaxe"); return; }

        res = effectuer_vote (jeton, choix);
        if (res == 0) strcpy (reponse, "210 vote ok");
        else if (res == -1) strcpy (reponse, "320 jeton inconnu");
        else if (res == -2) strcpy (reponse, "330 jeton a déjà voté");
        else strcpy (reponse, "500 erreur interne");

    } else sprintf (reponse, "400 mauvaise syntaxe");
}

```

À titre indicatif, voici des exemples de réalisation des fonctions manquantes :

```

int chercher_jeton (char *email, char *jeton)
{
    /* Ici connexion à une base de données, ou recherche dans un fichier */

    /* Exemple pour tester */
    if (strcmp (email, "thiel@localhost") == 0)
        strcpy (jeton, "a1b23c78");
    else if (strcmp (email, "yves@gmail.com") == 0)
        strcpy (jeton, "7f78ada1");
    else { strcpy (jeton, ""); return -1; }

    return 0;
}

int effectuer_vote (char *jeton, char *vote)
{
    /* Ici connexion à une B.D., ou recherche et enregistrement dans un fichier. */

    /* Exemple pour tester */
    (void) vote;
    if (strcmp (jeton, "a1b23c78") == 0)
        return 0; /* vote enregistré */
}

```

```

    else if (strcmp (jeton, "7f78ada1") == 0)
        return -2; /* a déjà voté */

    return -1; /* mauvais jeton */
}

```

5) Écrire la fonction `int dialoguer_avec_un_client (int soc)` recevant en paramètre une socket UDP/IP `soc`. La fonction lit une requête dans la socket, traite la requête puis envoie la réponse dans la socket. Elle renvoie 0 en cas de succès, -1 en cas d'erreur.

La fonction `main` consistera à créer une socket UDP/IP, boucler sur le dialogue avec un client, puis fermer la socket. *Ne pas écrire cette fonction.*

```

int dialoguer_avec_un_client (int soc)
{
    char requete[1500], reponse[1500];
    int k;
    struct sockaddr_in adrC;

    printf ("Attente lecture ...\n");
    k = bor_recvfrom_in (soc, requete, sizeof(requete)-1, &adrC);
    if (k < 0) return -1;
    requete[k] = 0;
    printf ("Reçu : %s\n", requete);

    traitement_requete (requete, reponse, sizeof(reponse));

    k = bor_sendto_in (soc, reponse, strlen(reponse), &adrC);
    if (k < 0) return -1;
    printf ("Envoyé : %s\n", reponse);

    return 0;
}

```

Pour être complet, voici la fonction `main` :

```

int boucle_princ = 1;
void capter_sig (int sig)
{
    printf ("Signal %d capté\n", sig);
    boucle_princ = 0;
}

int main (int argc, char *argv[])
{
    struct sockaddr_in adrS;
    int soc, nport, res;

    if (argc != 2)
        { fprintf (stderr, "USAGE: %s port\n", argv[0]); exit (1); }
    nport = atoi (argv[1]);

    soc = creer_socket_udpip (nport, &adrS);
    if (soc < 0) exit (1);

    bor_signal (SIGINT, capter_sig, 0);
    while (boucle_princ) {
        res = dialoguer_avec_un_client (soc);
        if (res < 0) break;
    }
    close (soc);
    return res == 0 ? 0 : 1;
}

```