Faculté des Sciences Aix*Marseille Université

Année universitaire 2014/2015

Site : ☒ Luminy ☐ St-Charles ☐ St-Jérôme ☐ Cht-Gombert ☒ Aix-Montperrin ☐ Aubagne-SATIS
Sujet de : ☒ 1er semestre ☐ 2ème semestre ☐ Session 2 Durée de l'épreuve : 2h
Examen de : L3 Nom du diplôme : Licence d'Informatique
Code du module : SIN5U3 Libellé du module : Réseau et Communication
Calculatrices autorisées : NON Documents autorisés : OUI, notes de Cours/TD/TP

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

I. Moniteur POP3 de emails en C

On se propose de réaliser un programme C `monipop3.c` qui affiche régulièrement le nombre de messages et leur taille totale présents dans un compte email via le protocole POP3. On rappelle que POP3 est un protocole ASCII 7 bits sur TCP/IP à base de question/réponses.

Voici un exemple de session réalisé avec `netcat` (ici on se trompe volontairement de mot de passe pour montrer l'effet) ; le résultat est qu'il y a 3 messages pour un total de 9251 octets.

`$ netcat` *serveur* `110`	
	`+OK POP3 ready`
`USER` *user*	
	`+OK`
`PASS` *mauvais mdp*	
	`-ERR Invalid login or pw`
`PASS` *bon mdp*	
	`-ERR Invalid command`
`USER` *user*	
	`+OK`
`PASS` *bon mdp*	
	`+OK Mailbox ready`
`STAT`	
	`+OK 3 9251`
`QUIT`	
	`+OK`
`$`	

On se donne le type `Etat` pour coder l'état correspondant aux différentes phases du dialogue :

```
typedef enum {
    DEMARRER_CONN, ATTENDRE_CONN_OK, ENVOYER_USER, ATTENDRE_USER_OK,
    ENVOYER_PASS,  ATTENDRE_PASS_OK, ENVOYER_STAT, ATTENDRE_STAT_OK,
    ENVOYER_QUIT,  ATTENDRE_QUIT_OK, TERMINER_CONN
} Etat;
```

On définit le type `Moniteur` pour mémoriser tous les paramètres de notre programme :

Les champs `user` et `pass` servent à l'identification ; `mess` contient le message en émission ou en réception (selon `etat`), `pos` est la position d'émission ou d'insertion dans `mess` ; `delai` est le délai en secondes au bout duquel on réinterroge le serveur POP3 ; `pop3_ok` est vrai si le dialogue en POP3 s'est déroulé sans erreur (c'est-à-dire si aucun message commançant par `"-ERR"` n'a été reçu).

```
#define MESS_SIZE 1024
typedef struct {
    int soc;
    char *serveur_nom;
    struct sockaddr_in client_adr,
                       serveur_adr;
    int serveur_port;
    char *user, *pass;
    Etat etat;
    char mess[MESS_SIZE];
    int pos, delai, pop3_ok;
} Moniteur;
```

1) Écrire la fonction `void changer_etat (Moniteur *mon, Etat etat)` qui reçoit en paramètre un nouvel `etat`. La fonction le mémorise dans `mon`, puis initialise le champ `pos` à 0 (signifiant que dans ce nouvel `etat`, aucun octet n'a encore été envoyé ou reçu). Si `etat` est `ENVOYER_USER` ou `ENVOYER_PASS`, le champ `mess` est affecté à `"USER` *user*`\n"` ou à `"PASS` *pass*`\n"`, respectivement, en remplaçant *user* ou *pass* par leur valeur ; si `etat` est `ENVOYER_STAT` ou `ENVOYER_QUIT`, le champ `mess` est affecté à `"STAT\n"` ou `"QUIT\n"`, respectivement ; pour toutes les autres valeurs de `etat`, le champ `mess` est affecté à la chaîne vide.
Par la suite, *tous* les changements d'état seront faits en appelant cette fonction.

2) Écrire la fonction `int connecter_pop3 (Moniteur *mon)`, qui passe le champ `etat` à `DEMARRER_CONN`, puis crée une socket TCP/IP qui sera mémorisée dans le champ `soc` ; elle affiche l'adresse IPv4 du serveur POP3 mémorisée dans le champ `serveur_adr` (on suppose que la résolution d'adresse est faite dans le `main`), puis s'y connecte. En cas d'erreur, la fonction renvoie -1 ; en cas de succès elle passe `etat` à `ATTENDRE_CONN_OK`, `pop3_ok` à 1, puis renvoie 0.

3) Écrire la fonction `int envoyer_message (Moniteur *mon)`, dans laquelle on suppose que la socket `soc` est connectée au serveur POP3 et que le champ `etat` est dans la catégorie `ENVOYER_*`, ce qui signifie que le champ `mess` contient la chaîne à envoyer. La fonction écrit dans la socket le message `mess` à partir de la position `pos` en une seule opération. En cas d'erreur elle renvoie `-1`, sinon elle met à jour `pos`, puis teste si la chaîne complète a été envoyée ; si tel est le cas, `etat` est incrémenté. Enfin, la fonction renvoie le nombre de caractères qu'elle a écrit lors de cet envoi.

4) Écrire la fonction `void afficher_stats_mails (char *reponse)` qui reçoit en paramètre la `reponse` positive du serveur pour la commande `STAT`. La fonction en extrait le nombre de mails et leur taille totale en octets, puis les affiche dans une phrase dans la sortie standard si l'extraction a réussi, sinon affiche un message d'erreur de format dans la sortie d'erreur.

5) Écrire la fonction `int lire_et_traiter_reponse (Moniteur *mon)`, dans laquelle on suppose que la socket `soc` est connectée au serveur POP3 et que le champ `etat` est dans la catégorie `ATTENDRE_*_OK`, ce qui signifie que le champ `mess` contient la réponse partiellement lue. La fonction lit dans la socket la (suite de la) réponse et la stocke dans `mess` à partir de la position d'insertion `pos`, en une seule opération. En cas d'erreur ou de fin de fichier elle renvoie `-1` ou 0, sinon elle met à jour `pos`. Elle teste ensuite si `mess` contient le marqueur de fin `"\n"`, sinon renvoie le nombre de caractères lus au début de la fonction. Le début de `mess` est ensuite testé : s'il n'est pas `"+OK"`, alors on affiche un message d'erreur présentant ce message, on passe `etat` à `ENVOYER_QUIT` et on met `pop3_ok` à 0 ; sinon (en cas de réponse positive donc), d'une part si `etat` est `ATTENDRE_STAT_OK` on appelle `afficher_stats_mails`, puis dans tous les cas on incrémente l'état. Finalement on renvoie le nombre de caractères lus au début de la fonction.

6) On suppose disposer des macros `etat_est_envoyer(x)` qui s'évalue à vrai si `x` vaut `ENVOYER_USER`, `ENVOYER_PASS`, `ENVOYER_STAT` ou `ENVOYER_QUIT`, et `etat_est_attendre(x)` qui s'évalue à vrai si `x` vaut `ATTENDRE_CONN_OK`, `ATTENDRE_USER_OK`, `ATTENDRE_PASS_OK`, `ATTENDRE_STAT_OK` ou `ATTENDRE_QUIT_OK`.

Écrire la fonction `int dialoguer_en_pop3 (Moniteur *mon)` dans laquelle on suppose que la socket `soc` est connectée au serveur POP3. La fonction fait une boucle infinie interruptible par `SIGINT`. À chaque itération, la fonction scrute la socket `soc` en lecture ou en écriture selon `etat`, avec un timeout de 10 secondes. En cas d'erreur la fonction renvoie -1 ; si le délai est dépassé, la fonction affiche un message et renvoie 0. Si la socket est éligible, selon `etat` on appelle `envoyer_message` ou `lire_et_traiter_reponse` ; si la fonction appelée renvoie une valeur $\leqslant 0$ on renvoie cette valeur. À la fin de la boucle de scrutation on renvoie 0 si `etat` est `TERMINER_CONN`.

7) Écrire le programme principal dont l'usage est : `monipop3 host port user pass delai`
où `host` est l'adresse du serveur et `port` le port du service `POP3`, `user` et `pass` identifient l'utilisateur, et `delai` est le délai en secondes au bout duquel on se reconnecte au serveur pour redemander les statistiques.
Le programme décode les arguments, les stocke dans une variable `mon` de type `Moniteur`, met en place les handlers de signaux habituels ; ensuite le programme résout l'adresse du serveur et la stocke ainsi que son port dans le champ `adr_serveur` de `mon`.
Le programme entre ensuite dans une boucle infinie interruptible par `SIGINT`, dans laquelle il se connecte au serveur POP3, effectue un dialogue POP3 dans le but d'afficher les statistiques, ferme la socket ; s'il n'y a pas eu d'erreur, en particulier dans le dialogue POP3, la boucle itère au bout du `delai` en secondes, sinon le programme se termine immédiatement.

Correction

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

I. Moniteur POP3 de emails en C

On se propose de réaliser un programme C `monipop3.c` qui affiche régulièrement le nombre de messages et leur taille totale présents dans un compte email via le protocole POP3. On rappelle que POP3 est un protocole ASCII 7 bits sur TCP/IP à base de question/réponses.

Voici un exemple de session réalisé avec `netcat` (ici on se trompe volontairement de mot de passe pour montrer l'effet) ; le résultat est qu'il y a 3 messages pour un total de 9251 octets.

`$ netcat` *serveur* `110`	
	`+OK POP3 ready`
`USER` *user*	
	`+OK`
`PASS` *mauvais mdp*	
	`-ERR Invalid login or pw`
`PASS` *bon mdp*	
	`-ERR Invalid command`
`USER` *user*	
	`+OK`
`PASS` *bon mdp*	
	`+OK Mailbox ready`
`STAT`	
	`+OK 3 9251`
`QUIT`	
	`+OK`
`$`	

On se donne le type `Etat` pour coder l'état correspondant aux différentes phases du dialogue :

```
typedef enum {
    DEMARRER_CONN, ATTENDRE_CONN_OK, ENVOYER_USER, ATTENDRE_USER_OK,
    ENVOYER_PASS,  ATTENDRE_PASS_OK, ENVOYER_STAT, ATTENDRE_STAT_OK,
    ENVOYER_QUIT,  ATTENDRE_QUIT_OK, TERMINER_CONN
} Etat;
```

On définit le type `Moniteur` pour mémoriser tous les paramètres de notre programme :

Les champs `user` et `pass` servent à l'identification ; `mess` contient le message en émission ou en réception (selon `etat`), `pos` est la position d'émission ou d'insertion dans `mess` ; `delai` est le délai en secondes au bout duquel on réinterroge le serveur POP3 ; `pop3_ok` est vrai si le dialogue en POP3 s'est déroulé sans erreur (c'est-à-dire si aucun message commançant par `"-ERR"` n'a été reçu).

```
#define MESS_SIZE 1024
typedef struct {
    int soc;
    char *serveur_nom;
    struct sockaddr_in client_adr,
                       serveur_adr;
    int serveur_port;
    char *user, *pass;
    Etat etat;
    char mess[MESS_SIZE];
    int pos, delai, pop3_ok;
} Moniteur;
```

1) Écrire la fonction `void changer_etat (Moniteur *mon, Etat etat)` qui reçoit en paramètre un nouvel `etat`. La fonction le mémorise dans `mon`, puis initialise le champ `pos` à 0 (signifiant que dans ce nouvel `etat`, aucun octet n'a encore été envoyé ou reçu). Si `etat` est `ENVOYER_USER` ou `ENVOYER_PASS`, le champ `mess` est affecté à `"USER` *user*`\n"` ou à `"PASS` *pass*`\n"`, respectivement, en remplaçant *user* ou *pass* par leur valeur ; si `etat` est `ENVOYER_STAT` ou `ENVOYER_QUIT`, le champ `mess` est affecté à `"STAT\n"` ou `"QUIT\n"`, respectivement ; pour toutes les autres valeurs de `etat`, le champ `mess` est affecté à la chaîne vide.
Par la suite, *tous* les changements d'état seront faits en appelant cette fonction.

```
void changer_etat (Moniteur *mon, Etat etat)
{
    mon->etat = etat;
    mon->pos = 0;

    switch (etat) {
        case ENVOYER_USER : sprintf (mon->mess, "USER %s\n", mon->user); break;
        case ENVOYER_PASS : sprintf (mon->mess, "PASS %s\n", mon->pass); break;
        case ENVOYER_STAT : sprintf (mon->mess, "STAT\n"); break;
        case ENVOYER_QUIT : sprintf (mon->mess, "QUIT\n"); break;

        default : mon->mess[0] = '\0';
    }
}
```

2) Écrire la fonction `int connecter_pop3 (Moniteur *mon)`, qui passe le champ `etat` à `DEMARRER_CONN`, puis crée une socket TCP/IP qui sera mémorisée dans le champ `soc` ; elle affiche l'adresse IPv4 du serveur POP3 mémorisée dans le champ `serveur_adr` (on suppose que la résolution d'adresse est faite dans le `main`), puis s'y connecte. En cas d'erreur, la fonction renvoie -1 ; en cas de succès elle passe `etat` à `ATTENDRE_CONN_OK`, `pop3_ok` à 1, puis renvoie 0.

```
int connecter_pop3 (Moniteur *mon)
{
    changer_etat (mon, DEMARRER_CONN);
    mon->soc = bor_create_socket_in (SOCK_STREAM, 0, &mon->client_adr);
    if (mon->soc < 0) return -1;

    printf ("Connexion avec %s ...\n", bor_adrtoa_in (&mon->serveur_adr));
    if (bor_connect_in (mon->soc, &mon->serveur_adr) < 0)  {
        close (mon->soc); return -1;
    }

    changer_etat (mon, ATTENDRE_CONN_OK); mon->pop3_ok = 1;
    return 0;
}
```

3) Écrire la fonction `int envoyer_message (Moniteur *mon)`, dans laquelle on suppose que la socket `soc` est connectée au serveur POP3 et que le champ `etat` est dans la catégorie `ENVOYER_*`, ce qui signifie que le champ `mess` contient la chaîne à envoyer. La fonction écrit dans la socket le message `mess` à partir de la position `pos` en une seule opération. En cas d'erreur elle renvoie `-1`, sinon elle met à jour `pos`, puis teste si la chaîne complète a été envoyée ; si tel est le cas, `etat` est incrémenté. Enfin, la fonction renvoie le nombre de caractères qu'elle a écrit lors de cet envoi.

```
int envoyer_message (Moniteur *mon)
{
    //printf ("envoyer_message '%s'\n", mon->mess+mon->pos);
    int k = bor_write_str (mon->soc, mon->mess+mon->pos);
    if (k <= 0) return k;
    mon->pos += k;

    // Tout a été envoyé ?
    if (mon->pos >= (int)strlen(mon->mess))
        changer_etat (mon, mon->etat+1);
    return k;
}
```

4) Écrire la fonction `void afficher_stats_mails (char *reponse)` qui reçoit en paramètre la `reponse` positive du serveur pour la commande `STAT`. La fonction en extrait le nombre de mails et leur taille totale en octets, puis les affiche dans une phrase dans la sortie standard si l'extraction a réussi, sinon affiche un message d'erreur de format dans la sortie d'erreur.

```
void afficher_stats_mails (char *reponse)
{
    int nb_mails, taille_mails;
    if (sscanf (reponse, "+OK %d %d", &nb_mails, &taille_mails) != 2)
        fprintf (stderr, "Erreur de format dans la réponse : \"%s\"\n",
            reponse);
    else printf ("STATS: il y a %d mails pour un total de %d octets \n",
            nb_mails, taille_mails);
}
```

5) Écrire la fonction `int lire_et_traiter_reponse (Moniteur *mon)`, dans laquelle on suppose que la socket `soc` est connectée au serveur POP3 et que le champ `etat` est dans la catégorie `ATTENDRE_*_OK`, ce qui signifie que le champ `mess` contient la réponse partiellement lue. La fonction lit dans la socket la (suite de la) réponse et la stocke dans `mess` à partir de la position d'insertion `pos`, en une seule opération. En cas d'erreur ou de fin de fichier elle renvoie `-1` ou 0, sinon elle met à jour `pos`. Elle teste ensuite si `mess` contient le marqueur de fin `"\n"`, sinon renvoie le nombre de caractères lus au début de la fonction. Le début de `mess` est ensuite testé : s'il n'est pas `"+OK"`, alors on affiche un message d'erreur présentant ce message, on passe `etat` à `ENVOYER_QUIT` et on met `pop3_ok` à 0 ; sinon (en cas de réponse positive donc), d'une part si `etat` est `ATTENDRE_STAT_OK` on appelle `afficher_stats_mails`, puis dans tous les cas on incrémente l'état. Finalement on renvoie le nombre de caractères lus au début de la fonction.

```
int lire_et_traiter_reponse (Moniteur *mon)
{
    //printf ("lire_reponse ...\n");
    int k = bor_read_str (mon->soc, mon->mess+mon->pos, MESS_SIZE-mon->pos);
    if (k <= 0) return k;
    mon->pos += k;
    //printf ("message lu '%s'\n", mon->mess);

    // Marqueur de fin reçu ?
    if (!strstr(mon->mess, "\n")) return k;

    if (strncmp (mon->mess, "+OK", 3) != 0) {
         fprintf (stderr, "POP3: %s", mon->mess+1);
         changer_etat (mon, ENVOYER_QUIT); mon->pop3_ok = 0;
    } else {
        if (mon->etat == ATTENDRE_STAT_OK) afficher_stats_mails (mon->mess);
        changer_etat (mon, mon->etat+1);
    }
    return k;
}
```

6) On suppose disposer des macros `etat_est_envoyer(x)` qui s'évalue à vrai si `x` vaut `ENVOYER_USER`, `ENVOYER_PASS`, `ENVOYER_STAT` ou `ENVOYER_QUIT`, et `etat_est_attendre(x)` qui s'évalue à vrai si `x` vaut `ATTENDRE_CONN_OK`, `ATTENDRE_USER_OK`, `ATTENDRE_PASS_OK`, `ATTENDRE_STAT_OK` ou `ATTENDRE_QUIT_OK`.

Écrire la fonction `int dialoguer_en_pop3 (Moniteur *mon)` dans laquelle on suppose que la socket `soc` est connectée au serveur POP3. La fonction fait une boucle infinie interruptible par `SIGINT`. À chaque itération, la fonction scrute la socket `soc` en lecture ou en écriture selon `etat`, avec un timeout de 10 secondes. En cas d'erreur la fonction renvoie -1 ; si le délai est dépassé, la fonction affiche un message et renvoie 0. Si la socket est éligible, selon `etat` on appelle `envoyer_message` ou `lire_et_traiter_reponse` ; si la fonction appelée renvoie une valeur $\leqslant 0$ on renvoie cette valeur. À la fin de la boucle de scrutation on renvoie 0 si `etat` est `TERMINER_CONN`.

```
int dialoguer_en_pop3 (Moniteur *mon)
{
    while (boucle_princ) {  // voir question 7
        fd_set set_in, set_out;
        FD_ZERO (&set_in); FD_ZERO (&set_out);
        if (etat_est_envoyer(mon->etat)) FD_SET (mon->soc, &set_out);
        if (etat_est_attendre(mon->etat)) FD_SET (mon->soc, &set_in);

        struct timeval t;
        t.tv_sec = 10; t.tv_usec = 0;
        int res = select (mon->soc+1, &set_in, &set_out, NULL, &t);

        if (res < 0) {
            if (errno == EINTR) continue;
            perror ("select"); return -1;
        }

        if (res == 0) { fprintf (stderr, "Délai dépassé\n"); return -1; }

        if (etat_est_envoyer(mon->etat) && FD_ISSET (mon->soc, &set_out)) {
            int k = envoyer_message (mon);
            if (k <= 0) return k;
        } else if (etat_est_attendre(mon->etat) && FD_ISSET (mon->soc, &set_in)) {
            int k = lire_et_traiter_reponse (mon);
            if (k <= 0) return k;
        }

        if (mon->etat == TERMINER_CONN) return 0;
    }
    return -1;
}
```

7) Écrire le programme principal dont l'usage est : `monipop3 host port user pass delai`
où `host` est l'adresse du serveur et `port` le port du service `POP3`, `user` et `pass` identifient l'utilisateur, et `delai` est le délai en secondes au bout duquel on se reconnecte au serveur pour redemander les statistiques.
Le programme décode les arguments, les stocke dans une variable `mon` de type `Moniteur`, met en place les handlers de signaux habituels ; ensuite le programme résout l'adresse du serveur et la stocke ainsi que son port dans le champ `adr_serveur` de `mon`.
Le programme entre ensuite dans une boucle infinie interruptible par `SIGINT`, dans laquelle il se connecte au serveur POP3, effectue un dialogue POP3 dans le but d'afficher les statistiques, ferme la socket ; s'il n'y a pas eu d'erreur, en particulier dans le dialogue POP3, la boucle itère au bout du `delai` en secondes, sinon le programme se termine immédiatement.

```
#include "bor-util.h"

int boucle_princ = 1;
void capter_fin () { boucle_princ = 0; }

int main (int argc, char *argv[])
{
    Moniteur mon;

    if (argc-1 != 5) {
        fprintf (stderr, "USAGE: %s host port user pass delai\n", argv[0]);
        exit (1);
    }
    mon.serveur_nom  = argv[1];
    mon.serveur_port = atoi(argv[2]);
    mon.user         = argv[3];
    mon.pass         = argv[4];
    mon.delai        = atoi(argv[5]);

    bor_signal (SIGPIPE, SIG_IGN, SA_RESTART);
    bor_signal (SIGINT, capter_fin, 0);

    printf ("Résolution adresse ...\n");
    if (bor_resolve_address_in (mon.serveur_nom, mon.serveur_port,
        &mon.serveur_adr) < 0) exit (1);

    while (boucle_princ) {
        if (connecter_pop3 (&mon) < 0) exit (1);
        int k = dialoguer_en_pop3 (&mon);
        close (mon.soc);
        if (k < 0 || !mon.pop3_ok) exit (1);
        sleep (mon.delai);
    }
    exit (0);
}
```