

Enoncé de Réseau & Programmation Réseau V 1.3
(2 gestion , 2 telecom , 2 indus)

Les plateformes de développement

1) Il faut utiliser la librairie de composants

lib2017Etud.tar

2) Le programme se fera en c sous linux avec le compilateur gcc soit avec Ubuntu ,soit avec sun. L'installation d'ubuntu voir le dossier ubuntu dans mon centre de ressources.

Les composants de l'application Cinéma

La librairie lib2016 <u>HV</u> Les deux dernières lettres sont à remplacer par vos noms respectifs	Cette librairie contient les fonctions partagées entre Admin et le programme Ser
admin	Ce programme permet de créer les séances de cinéma qui sont stockée dans un fichier Séances. Il permettra aussi de facturer la vente de ticket et le résultat sera stocké dans le fichier Facture
ser	Ce programme reçoit les requêtes réseaux du programme cli et selon la requête effectue une recherche d'une séance dans le fichier Séances créé par admin ou une vente qui est stockée alors dans le fichier Facture
cli	Ce programme permet à l'utilisateur de consulter les séances disponibles et de vendre des tickets. Il les transmet au serveur via des requêtes udp qui lui fournit toujours par réseau les réponses.

Personnalisation du programme

- Le nom de la librairie lib2016 doit se terminer par vos initiales
- Les structures de données que vous créez (y compris Transaction) doivent se terminer par les initiales.
- Même principe pour tout les fonctions ou procédures définies.
- Chaque programme quand il se lance doit afficher le nom et prénom de l'étudiant
- **Vous devrez avoir un champ personnel au niveau de votre application.**

La liste des champs personnels

Numéro	Nom du champ	Valeurs
1	Réalisateur	Spielberg , Lucas , Tarentino
2	Studio	Warner, Sony , Disney
3	Acteur Principal	Tom Cruise, Daniel Craig, Brad Pitt, Robert Downey
4	Actrice Principale	Jennifer Lawrence , Kristen Stewart, Charlize Theron
5	Genre	Policier, comique, Science-fiction
6	Budget	50 m\$, 100 m\$, 150 m\$
7	Projection	Couleur, noir et blanc, 3D
8	Année	1975, 1980 , 1990
9		

1)Le programme Admin, champ Personnel & Ecrire la fonction recherche

documentation

T4P , annexe les opérations sur les fichiers.

Un admin de base à compléter est fourni dans lib2017Etud\step0

Copier le contenu de ce dossier dans lib2017Etud\step1

Le makefile

```
LIBS=
all:    admin  lib2016HV.o

lib2016HV.o:  lib2016HV.c    lib2016HV.h    data.h
    echo "compilation de lib2016"
    gcc -c lib2016HV.c

admin: data.h  admin.c lib2016HV.o
    echo "Compilation de admin"
    gcc -o admin    admin.c lib2016HV.o
```

pour exécuter taper make. Si des problèmes de compilation se produisent, effacer les fichier admin et lib2016HV.o

Rappelons que LIB2016 doit être renommé avec vos initiales

Modifiez data.h pour votre champ perso et les fonctions correspondantes

```
#ifndef DATAH

#else
#define DATAH

struct Seance {
    int Reference ;
    char Film[60] ;
    int Places ;
    int Salle ;
    /* Rajoutez ici votre champ perso */
    char Date[10] ;
};

struct Facture
{
    int NumeroFacturation ;
    char NomClient[60] ;
    int DateFacturation ;
    int Places ;
    int Reference ;
};

#endif
```

En première étape, il vous est demandé de rajouter un champ que je donnerais
Vous modifier la saisie des séances et leur affichage

Modifier la fonction a propos

Faire l’affichage de vos noms & prénoms

Ecrire la fonction recherche

La fonction suivante est à écrire, Le prototype est à type indicatif, elle sera placée dans lib2016.c

```
int Recherche(char* NomFichier,int Reference ,struct Seance
*UnRecord)
```

int Recherche retourne 1 si la recherche a réussi.

Char *Nom désigne chaque fois le nom du fichier qui contient les séances,

Reference : le numéro de la séance à chercher

struct Seance pointera vers le Record qui correspondra à la recherche

Je vous conseille vivement de lire le code de la fonction SaiSieRecord pour se rappeler la syntaxe d’un pointeur vers une structure. Regarder aussi CreationAjoutFichier ;

Fonctionnement illustré

```
vanstap@vanstap2:~/lib2017/Step1$ ./admin
-----
1) Ajout
2) Seances
3) Recherche
4) Achat
5) Factures
6) A propos
7) exit
-----
2
Ouverture reussie
Record lu 1 et Position actuelle dans le fichier 84
1 Terminator 1          100
Record lu 1 et Position actuelle dans le fichier 168
2 Terminator 2          100
Record lu 1 et Position actuelle dans le fichier 252
3 Terminator 3          100
-----
1) Ajout
2) Seances
3) Recherche
4) Achat
5) Factures
6) A propos
```

7) exit

3

Saisie Reference:2

Ouverture reussie

Reference lue 1 et Position actuelle dans le fichier 84

2 Terminator 2 100 **(**)**

Les messages en italique sont des messages de log. Il permettent de détecter plus facilement les erreurs en cas de problèmes

En **(**)** vous devez rajouter votre propre champ personnel

2) Un Client et un seul serveur la requête recherche

documentation

T4P EX02 : Un serveur et un seul client : structure de donnée

Préalables

Copier le contenu du dossier step1 dans le dossier step2. Y copier aussi le contenu de EX02

Le makefile

Il est conseillé d'utiliser la makefile suivant :

```
LIBS=
all:    admin  cli      ser      udplib.o lib2016.o

lib2016.o:    lib2016.c      lib2016.h      data.h
             echo "compilation de lib2016"
             cc -c lib2016.c

admin: data.h  admin.c lib2016.o
      echo "Compilation de admin"
      cc -o admin      admin.c lib2016.o

udplib.o:    ../udplib/udplib.h      ../udplib/udplib.c
             echo "Compilation de udplib.o"
             cc -c ../udplib/udplib.c

cli:    cli.c      structure.h      udplib.o
      echo "Compilation de client"
      cc -o cli cli.c      udplib.o      $(LIBS)

ser:    ser.c      structure.h      data.h  udplib.o lib2016.o
      echo "Compilation de serveur"
      cc -o ser ser.c      udplib.o lib2016.o      $(LIBS)
```

Lib2016 est à compléter avec vos initiales

Si vous travaillez avec SUN ; complétez la variable LIBS comme ci-dessous

```
LIBS=-lsocket -lnsl
```

En cas de problèmes de compilation, effacer les exécutable cli et ser et effacer tous les .o

La structure structure.h

```
enum TypeRequete {
    Question = 1 ,
    Achat = 2 ,
    Livraison= 3 ,
    OK = 4,
    Fail = 5
};
```

```
struct Requete
{
    enum TypeRequete Type ;
    int Numero ;
    int NumeroFacture ;
    int Date ;
    int Reference ;
    int Places ;
    int Prix ;
    char Film[80] ;
    char NomClient[80] ;
    /* Rajoutez votre champ perso */
};
```

La structure requête est utilisée dans les échanges réseaux entre le programme **cli** et le programme **ser**.

Le menu du client

Note en italique, vous avez les informations de log

```
vanstap@vanstap2:~/lib2017/Step2$ ./cli 127.0.0.1 1400 127.0.0.1 1300
port 1400
CreateSockets 3
Saisie numero :1          ←----- Vous indiquez le numéro de séance que vous voulez
Envoi de 188 bytes

bytes:188
Film:Terminator 1        **    ← Le serveur vous renvoie le nom du film et votre champ perso
Saisie numero :
```

L'affichage du serveur

```
vanstap@vanstap2:~/lib2017/Step2$ ./ser 127.0.0.1 1300
Ceci est le serveur
port 1300
CreateSockets : 3
bytes:188 Reference:1    <- le serveur a reçu le numéro de la séance du client
Ouverture reussie
res :1 Reference:Terminator 1 <- le serveur a trouvé le record dans le fichier séances
bytes:188                <- la réponse est renvoyée au client
```

Le serveur utilise la fonction recherche incluse dans lib2016.c

**** Vous devrez rajouter comme information le champ qui vous est propre.**

3) Un Serveur et plusieurs clients

Documentation

Consulter Ex03 : Un serveur et plusieurs clients: Structure de donnée du tome T4P

Préalables

Copier le contenu du dossier step2 dans le dossier step3.

Fonctionnement illustré des deux clients

Le client intègre maintenant un menu.

Le client demande le numéro de séance à chercher et envoie l'identifiant au serveur par réseau qui répond en fournissant le nom du film, le nombre de places restantes et le champ perso.

```
vanstap@vanstap2:~/lib2017/Step3$ ./cli 127.0.0.1 1400 127.0.0.1 1300
```

```
port 1400
```

```
CreateSockets 3
```

```
-----  
1) Demander une reference
```

```
3) Quitter
```

```
4) A propos  
-----
```

```
Choix :1
```

```
Reference :1
```

```
Envoi de 188 bytes
```

```
  
bytes:188
```

```
Pour 1 Film:Terminator 1 Quantiteg: 100 (**)
```

```
-----  
1) Demander une reference
```

```
3) Quitter  
-----
```

```
Choix :
```

```
vanstap@vanstap2:~/lib2017/Step3$ ./cli 127.0.0.1 1500 127.0.0.1 1300
```

```
port 1500
```

```
CreateSockets 3
```

```
-----  
1) Demander une reference
```

```
3) Quitter  
-----
```

```
Choix :1
```

```
Reference :2
```

```
Envoi de 188 bytes
```

```
  
bytes:188
```

```
Pour 2 Film:Terminator 2 Quantiteg: 100 (**)
```

La structure à adopter au niveau du code du serveur

Un switch case est recommandé à cette étape pour pouvoir intégrer les étapes suivantes.

```
while(1)
{
    char *Res ;
    int res ;
    struct Seance UnRecord ;
    tm = sizeof(struct Requete) ;
    rc = ReceiveDatagram( Desc,&UneRequeteR ,tm, &sor ) ;
    if ( rc == -1 )
        perror("ReceiveDatagram") ;
    else
        fprintf(stderr,"bytes:%d Type:%d Numero:%d\n",rc,UneRequeteR.Type, UneRequeteR.Numero ) ;
    // Le traitement doit être séparé des requetes réseaux */
    switch(UnRequeteR.Type )
    {
        case Question:
            fprintf(stderr,"A rechercher %d \n", UneRequeteR.Reference , UneRequeteR.Numero ) ;
            res = Recherche("Seances",UneRequeteR.Reference ,&UnRecord) ;
            fprintf(stderr,"res :%d Reference:%s Quantité %d\n",res,UnRecord.Film,UnRecord.Places ) ;
            /* reponse avec psor qui contient toujours l'adresse du dernier client */
            strncpy(UnRequeteE.Film,UnRecord.Film,sizeof(UnRequeteE.Film)) ;
            UnRequeteE.Places = UnRecord.Places ;
            UnRequeteE.Numero = UneRequeteR.Numero ;
            UnRequeteE.Reference = UneRequeteR.Reference ;
            if (res)
                UnRequeteE.Type = OK ;
            else
                UnRequeteE.Type = Fail ;
            rc = SendDatagram(Desc,&UneRequeteE ,sizeof(struct Requete) ,&sor ) ;
            if ( rc == -1 )
                perror("SendDatagram:") ;
            else
                fprintf(stderr,"bytes:%d\n",rc ) ;
            break ;
        case Achat:
            {

                res = Reservation("Stock",UneRequeteR.Reference ,UneRequeteR.Places ) ;
                if ( res == 1 )
```

Le fonctionnement du serveur

```
vanstap@vanstap2:~/lib2017/Step3$ ./ser 127.0.0.1 1300
Ceci est le serveur
port 1300
CreateSockets : 3
bytes:188 Type:1 Numero:0
```

A rechercher 1

Ouverture reussie

res :1 Reference:Terminator 1 Quantité 100

bytes:188

bytes:188 Type:1 Numero:0

A rechercher 2

Ouverture reussie

Reference lue 1 et Position actuelle dans le fichier 84

res :1 Reference:Terminator 2 Quantité 100 **

bytes:188

**** Vous devrez rajouter comme information le champ qui vous est propre.**

3B) Un Serveur Affichant l'ip & port de ses clients

Documentation

Consulter l'exemple ex03 du tome4P

Préalables

Copier le contenu du dossier step3 dans le dossier step3B

Le fonctionnement illustré du serveur

Comme rien ne change au niveau du client, on fait uniquement l'affichage au niveau du serveur. On affiche comme info supplémentaire, l'ip et le port des clients qui se connectent après réception de leur requête.

```
vanstap@vanstap2:~/lib2017/Step3B$ ./ser 127.0.0.1 1300
Ceci est le serveur
port 1300
CreateSockets : 3
bytes:188 Type:1 Numero:0
Received packet from 127.0.0.1:1400
A rechercher 1
Ouverture reussie
res :1 Reference:Terminator 1 Quantité 100
bytes:188
```

4A) Le programme admin gérant l'achat et la génération de factures

Préalables

Copier le contenu du dossier step3B dans le dossier step04a

Le fonctionnement illustré

Il faut modifier le programme admin pour ajouter une option **Achat**. J'achète des places pour un numéro de séance donnée (par ex 1). Le programme modifie le nombre de places existantes en retirant la quantité commandée et ajoute une facture au nom de l'acheteur dans le fichier facture

```
vanstap@vanstap2:~/lib2017/Step4A$ ./admin

-----
1) Ajout
2) Seances
3) Recherche
4) Achat                                <- L'option à encoder
5) Factures
6) A propos
7) exit
-----
4
NomClient :toto hack                    <- on entre le nom du client
Reference :1                            <- le numéro de la séance
Places:1
Ouverture reussie
Trouve Reference Terminator 1
Record Ecrits 1
Ouverture reussie de Facturation
Mise à jour du Seances réussi
```

On va maintenant vérifier que la facture a été ajoutée dans le fichier facture

```
vanstap@vanstap2:~/lib2017/Step4A$ ./admin

-----
1) Ajout
2) Seances
3) Recherche
4) Achat
5) Factures
6) A propos
7) exit
-----
5
Ouverture reussie
...
Record lu 1 et Position actuelle dans le fichier 532
7 Toto 104                               1 1 0
Record lu 1 et Position actuelle dans le fichier 608
8 toto hack                             1 1 0
```

On notera que 8 est le numéro de la facture (on avait déjà créé des factures)

L'implémentation

Pour réserver la quantité commandée ; dans lib2016.c ajouter la ligne suivante :

```
int Reservation(char* NomFichier,int Reference ,int Quantite )
```

Attention la fonction retournera 0 si la Quantité n'est pas disponible, sinon elle décrémente le nombre de places de la quantité Demandée

Dans lib2016.c ; il faut ajouter la ligne suivante pour générer la facture

```
int Facturation(char NomFichier[80], char NomClient[60], int Date,int Quantite,int Reference)
```

Pour le moment le champ date, vous le mettez à zéro.

Pour générer les numéros de facture , vous pouvez utiliser la formule suivante :

```
UneFacture.NumeroFacturation = ftell( sortie ) / sizeof( struct Facture ) + 1 ;
```

4B) l'achat et la génération de factures par réseau

Préalables

Copier le contenu du dossier step4A dans le dossier step4B

On demande maintenant d'intégrer la fonction achat au menu client et d'intégrer les fonctions Reservation & Facturation de lib.c au programme serveur

Le fonctionnement illustré du client

```
vanstap@vanstap2:~/lib2017/Step4B$ ./cli 127.0.0.1 1400 127.0.0.1 1300
port 1400
CreateSockets 3
-----
1) Demander une reference
2) Acheter un ticket
3) Quitter
4) A propos
-----
Choix :2
NomClient :oscar 1004
Reference :2
Places:1
Envoi de 188 bytes

bytes:188
Achat Reussi Facture : 3
```

Le fonctionnement illustré du serveur

```
vanstap@vanstap2:~/lib2017/Step4B$ ./ser 127.0.0.1 1300
Ceci est le serveur
port 1300
CreateSockets : 3
bytes:188 Type:2 Numero:0
Received packet from 127.0.0.1:1400
Ouverture reussie
Record lu 1 et Position actuelle dans le fichier 84
Trouve Reference Terminator 2
Record Ecrits 1
Ouverture reussie de Facturation
Mise à jour du stock réussi
bytes:188
```

Vérification du résultat avec admin

```
vanstap@vanstap2:~/lib2017/Step4B$ ./admin
```

- ```

1) Ajout
2) Seances
3) Recherche
4) Achat
5) Factures
6) A propos
7) exit

```

```
5
```

```
Ouverture reussie
```

```
Record lu 1 et Position actuelle dans le fichier 76
```

```
1 ZEZEZEZEZE 7 3 0
```

```
Record lu 1 et Position actuelle dans le fichier 152
```

```
2 ZEZEZEZEZE 10 3 0
```

```
Record lu 1 et Position actuelle dans le fichier 228
```

```
3 oscar 1004 1 2 0
```



## 5A) L'achat, les clients gèrent les timeout et les doublons, le serveur ne gère pas les doublons

### Documentation

Ex07 du tome4P

### Préalables

Copier le contenu du dossier step4B dans le dossier step5A

### Analyse

Les transactions doivent maintenant être numérotées de manière à détecter les doublons au niveau du client. le client chaque fois qu'il transmet une demande d'achat, démarre un timer. Si timeout , le client retransmet la demande telle quelle sans incrémenter le numéro de transaction.

Au niveau du serveur via le ctrl z, on mettra le serveur en pause d'environ 30 secondes. Le serveur ne répond pas pendant ce temps aux requêtes du client. Quand le serveur se réveille il devra répondre à toutes les requêtes du client.

Le client vérifie que le paquet reçu correspond bien au numéro de transaction envoyé. Si ce n'est pas le cas , il affiche doublon et se remet en attente du bon numéro de transaction

### Le fonctionnement illustré du client

```
vanstap@vanstap2:~/lib2017/Step5A$./cli 127.0.0.1 1400 127.0.0.1 1300
port 1400
CreateSockets 3

1) Demander une reference ← Je vérifie en premier si il y'a assez de place
2) Acheter une reference
3) Quitter
4) A propos

Choix :1
Reference :1
Envoi de 188 bytes

bytes:188
Pour 1 Film:Terminator 1 Placesg: 96 ← Il reste 96 places pour le film terminator

1) Demander une reference
2) Acheter une reference
3) Quitter

Choix :2
NomClient :toto le fou
```

```

Reference :1 <- Je commande une place pour la séance 1
Places:1
Envoi de 188 bytes
error sur receive:: Interrupted system call ← Comme le serveur ne répond pas, le timeout se déclenche
rc -1 errno:4
Envoi de 188 bytes ← On renvoie la même requête sans la modifier
error sur receive:: Interrupted system call ← Comme le serveur ne répond pas, le timeout se déclenche
rc -1 errno:4
Envoi de 188 bytes ← On renvoie la même requête sans la modifier

bytes:188
Achat Reussi Facture : 7 <- Le serveur a enfin répondu

```

Attention si vous faites des achats en premier, la fonction demander une référence ne donnera plus de résultats corrects car elle peut être perturbée par les doublons générés par L'option acheter

### Le fonctionnement illustré du serveur

```

vanstap@vanstap2:~/lib2017/Step5A$./ser 127.0.0.1 1300
Ceci est le serveur
port 1300
CreateSockets 3
bytes:188 Type:1 Numero:0
A rechercher 1 ← le serveur fait l'opération de recherche
Ouverture reussie
res :1 Reference:Terminator 1 Quantité 96 ← il trouve la séance 1 et la renvoie
bytes:188
^Zlongjumped from interrupt CTRL Z 20 ← On met le serveur en pause de 30 secondes
Demarrage du sleep
Fin du sleep ← Fin de la pause
bytes:188 Type:2 Numero:1 ← Le serveur répond à la première requête
Ouverture reussie
Trouve Reference Terminator 1
Record Ecrits 1
Ouverture reussie
Mise à jour du stock réussi
bytes:188 ← Envoi de la réponse
bytes:188 Type:2 Numero:1 ← le serveur répond au premier doublon
Ouverture reussie ← Le serveur doit toujours répondre , car il ne sait pas
Trouve Reference Terminator 1 ← Si le client a reçu la réponse
Record Ecrits 1
Ouverture reussie
Mise à jour du stock réussi
bytes:188
bytes:188 Type:2 Numero:1 ← Le serveur répond au second doublon
Ouverture reussie
Trouve Reference Terminator 1
Record Ecrits 1
Ouverture reussie
Mise à jour du stock réussi

```

bytes:188

### Consulter les factures créées avec admin

```
vanstap@vanstap2:~/lib2017/Step5A$./admin
```

```

1) Ajout
```

```
2) Seances
```

```
3) Recherche
```

```
4) Achat
```

```
5) Factures
```

```
6) exit

```

```
2
```

```
Ouverture reussie
```

```
Record lu 1 et Position actuelle dans le fichier 84
```

```
1 Terminator 1 93 ← Un acheté et deux doublons
```

```
Record lu 1 et Position actuelle dans le fichier 168
```

```
2 Terminator 2 100
```

```
Record lu 1 et Position actuelle dans le fichier 252
```

```
3 Terminator 3 80

```

```
1) Ajout
```

```
2) Seances
```

```
3) Recherche
```

```
4) Achat
```

```
5) Factures
```

```
6) exit

```

```
5
```

```
Ouverture reussie
```

```
...
```

```
Record lu 1 et Position actuelle dans le fichier 532
```

```
7 toto le fou 1 1 0 ← Achat
```

```
Record lu 1 et Position actuelle dans le fichier 608
```

```
8 toto le fou 1 1 0 ← Premier doublon
```

```
Record lu 1 et Position actuelle dans le fichier 684
```

```
9 toto le fou 1 1 0 ← Second doublon
```

Le serveur à cette étape ne gère pas les doublons, ce sera corrigé plus tard

## 5B) Corrigeons le bug de la recherche

### Documentation

Ex07 du tome4P

### Préalables

Copier le contenu du dossier step5A dans le dossier step5B

Après le point 5A, si vous faites une recherche après avoir fait des achats ou des timeouts se sont produits, vous verrez que cela ne fonctionne. Car les doublons des achats sont lus avant le résultat de la recherche

Modifier le code de la recherche pour que maintenant ses transactions soient également numérotées.

Intégrer le timeout n'est pas nécessaire mais il faut intégrer au minimum le code de la gestion des doublons

Dans l'exemple suivant, on fait en premier un achat et on force les timeout en faisant ctrl z sur le serveur. Le client affiche le résultat. Je fais ensuite une recherche sur la référence un pour connaître le nombre de places restantes. Le client reçoit la réponse et traite les doublons.

```
vanstap@vanstap2:~/lib2017/Step5B$./cli 127.0.0.1 1400 127.0.0.1 1300
port 1400
CreateSockets 3

1) Demander une reference
2) Acheter une reference
3) Quitter
4) A propos

Choix :1
Reference :1
Envoi de 188 bytes

bytes:188
Pour 1 Film:Terminator 1 Places: 69

1) Demander une reference
2) Acheter une reference
3) Quitter

Choix :2
NomClient :yyyy
Reference :1
Places:1
Envoi de 188 bytes
error sur receive:: Interrupted system call ← le serveur est en sleep , le timeout se déclenche
rc -1 errno:4
Envoi de 188 bytes

bytes:188
Achat Reussi Facture : 10

1) Demander une reference
2) Acheter une reference
3) Quitter

Choix :1
Reference :1
Envoi de 188 bytes
doublon 1 !!!!! ←----- On ne traite pas les doublons résultant de l'achat

bytes:188
Pour 1 Film:Terminator 1 Places: 67 ← -2 à cause du doublon
```

## 6) L'achat, les clients gèrent les timeout et les doublons, le serveur traite les doublons

### Documentation

Ex07 du tome4P

### Préalables

Copier le contenu du dossier step5 dans le dossier step6

### Analyse

Quand les timers se déclenchent, le serveur répond toujours au client car il ne sait savoir si les paquets sont arrivés. Le problème est que le serveur ne vérifie pas si une facture a déjà été établie pour le client pour la séance. Donc si trois doublons pour une quantité commandée de un , on va retirer quatre fois les places alors que le client n'en a demandé qu'une place.

Pour corriger ce problème, le client doit fournir un renseignement supplémentaire qui est la date d'achat. Le serveur fait une recherche sur le nom de client et la date.

Si aucune correspondance n'est trouvée, Le serveur génère une nouvelle facture et réduit le nombre de places. Si on trouve une correspondance, le serveur retourne au client le numéro de facture déjà généré et ne touche pas au stock

La date sera stockée dans un entier sous forme simplifiée. Le 18 janvier à 11 heures donnera 011811

### L'affichage du client

```
vanstap@vanstap2:~/lib2017/Step6$./cli 127.0.0.1 1400 127.0.0.1 1300
port 1400
CreateSockets 3
```

```

1) Demander une reference
2) Acheter une reference
3) Quitter
4) A propos
```

```

Choix :1
Reference :1
Envoi de 188 bytes
```

```
bytes:188
Pour 1 Film:Terminator 1 Places: 67
```

```

1) Demander une reference
2) Acheter une reference
3) Quitter
```

```

Choix :2
NomClient :ulyse 31
Reference :1
```

```
Places:1
Date:1111
Envoi de 188 bytes
error sur receive:: Interrupted system call
rc -1 errno:4
Envoi de 188 bytes
error sur receive:: Interrupted system call
rc -1 errno:4
Envoi de 188 bytes

bytes:188
Achat Reussi Facture : 2

1) Demander une reference
2) Acheter une reference
3) Quitter

Choix :1
Reference :1
Envoi de 188 bytes
doublon 1 !!!!
doublon 1 !!!!

bytes:188
Pour 1 Film:Terminator 1 Places: 66
```

### L'affichage du serveur

```
vanstap@vanstap2:~/lib2017/Step6$./ser 127.0.0.1 1300
Ceci est le serveur
port 1300
CreateSockets 3
bytes:188 Type:1 Numero:0
A rechercher 1
Ouverture reussie
res :1 Reference:Terminator 1 Quantité 67
bytes:188
^Zlongjumped from interrupt CTRL Z 20
Demarrage du sleep
Fin du sleep
bytes:188 Type:2 Numero:1
Ouverture reussie
Record lu 1 et Position actuelle dans le fichier 76
On va tenter de réserver
Ouverture reussie
Trouve Reference Terminator 1
Record Ecrits 1
Reservation Reussie
Ouverture reussie
Mise à jour du stock réussie
```

```
bytes:188
bytes:188 Type:2 Numero:1
Ouverture reussie
Record lu 1 et Position actuelle dans le fichier 76
Doublon
bytes:188
bytes:188 Type:2 Numero:1
Ouverture reussie
Record lu 1 et Position actuelle dans le fichier 76
Doublon
bytes:188
bytes:188 Type:1 Numero:2
A rechercher 1
Ouverture reussie
res :1 Reference:Terminator 1 Quantité 66
bytes:188
```

## Le programme admin \*\*

```
vanstap@vanstap2:~/lib2017/Step6$./admin

1) Ajout
2) Seances
3) Recherche
4) Achat
5) Factures
6) exit

5
Ouverture reussie
Record lu 1 et Position actuelle dans le fichier 76
1 gros malin 1 1 111111
Record lu 1 et Position actuelle dans le fichier 152
2 ulyse 31 1 1 1111
```

## 7) Le calcul du crc

### Documentation

Voir Tome 21

### Analyse

Ajouter à la structure requête du client , un champ crc. Le crc doit être calculé avant envoi, on affichera sa valeur avant envoi et sa valeur à la réception. La politique pour une trame défectueuse est de la jeter sans acquittement.



## **8) Le serveur multiport ( Pour les gestion, indus dispensé ; telecom pour administration & sec)**

### **Documentation**

Ex05 : un serveur multclients du tome 4P

### **Analyse**

Le serveur écoute chaque requête sur un seul port.

Dans la version multi port, le serveur écoute les clients sur différents ports par exemple 1301, 1302 ou 1304.

Pour tester , il suffit de connecter un client sur le port 1301 , un autre sur le 1302, un dernier sur le 1304

## 9) La recherche sous forme de thread ( indus dispensé ; telecom pour administration & sec)

On demande d'écrire une fonction ThreadRecherche dans **ser.c** qui appellera la fonction recherche et répondre au client.

La syntaxe de cette fonction est la suivante

```
void *ThreadRecherche (void* Param)
```

Il est impératif de passer une copie des paramètres car sans copie. Imaginons qu'un second thread soit lancé avant l'achèvement du premier thread. Le second thread modifierait les paramètres du premier thread qui répondrait plus à son client . C'est pour cela qu'il est impératif de faire une copie. On déclare en premier une structure ST , a compléter par vos initiales.

```
struct STXX { /* XX vos initiales */
 int DescPublic ;
 struct sockaddr_in psorPublic ; /* r = remote */
 struct Requete UneRequeteR ;
};
```

Voici maintenant comment procéder dans le programme principal ser.c

```
struct STXX *pST ;
 pST = malloc(sizeof(struct ST)) ; // chaque thread doit avoir une copie unique des paramètres
 pST->DescPublic = Desc ;
 pST->psorPublic = psor ;
 pST->UneRequeteR = UneRequeteR ;

Lancement du thread
```

Pour récupérer les paramètres dans ThreadRecherche, Voici comment faire.

```
void *ThreadRecherche (void* Param)
{
 struct STXX *pST ;
 struct Requete UneRequeteE ;
 int DescPublic ;
 struct sockaddr_in psorPublic ;
 struct Requete UneRequeteR ;

 int res,rc ;
 fprintf(stderr,"Demarrage du Thread & attente section critique \n") ;
 pST = (struct STXX *) Param ;
 DescPublic = pST->DescPublic ;
 psorPublic = pST->psorPublic ;
 UneRequeteR = pST->UneRequeteR ;
```

ThreadRecherche doit être lancé en mode detach, voici la syntaxe

```
rc=pthread_attr_init(&attr);
rc=pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);
```

## 9) La fonction Facturation sous forme de thread en utilisant une section critique ( indus dispensé ; telecom pour administration & sec)

Il faut maintenant créer la fonction suivante

```
void *ThreadFacturation (void* Param)
```

qui sera appelé toujours en mode detach et le passage des paramètres se fera toujours comme expliqué au point 8.

La modification des fichier doit être impérativement protégée par une section critique.

Il est impératif de déclarer le mutex en global dans la fonction ser pour qu'il fonctionne correctement

```
pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER; // Toujours déclaré public
```

Pour montrer l'effet du mutex , un affichage sera fait avant l'entrée dans la section critique, mutex\_lock et après l'entrée, ainsi qu'un sleep pour permettre de lancer un second thread alors que le premier n'est pas achevé

```
fprintf(stderr,"Demarrage du Thread & attente section critique \n") ;
pthread_mutex_lock(&mutex1);
fprintf(stderr,"Entrée Section critique\n") ;
sleep(20) ;
```

## Grille d'évaluation pour les Deuxièmes telecom

L'évaluation est continue pour les trois premiers points de labo.

Les autres points peuvent être présentés au plus tard à l'examen.

| Numéro | Description                                                                                                                                 | A présenter<br>Semaine du (*)       |
|--------|---------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------|
| 1      | Le programme Admin, champ Personnel & Ecrire la fonction recherche                                                                          | 14/11<br>A présenter avant l'examen |
| 2      | Un Client et un seul serveur la requête recherche                                                                                           | 21/11<br>A présenter avant l'examen |
| 3      | Un Serveur et plusieurs clients                                                                                                             | 28/11<br>A présenter avant l'examen |
| 3B     | Un Serveur Affichant l'ip & port de ses clients                                                                                             | 5/12<br>Au plus tard à l'examen     |
| 4A     | Le programme admin gérant l'achat et la génération de factures                                                                              | 12/12<br>Au plus tard à l'examen    |
| 4B     | l'achat et la génération de factures par réseau                                                                                             | Au plus tard à l'examen             |
| 5A     | L'achat, les clients gèrent les timeout et les doublons, le serveur ne gère pas les doublons                                                | Au plus tard à l'examen             |
| 5B     | Corrigeons le bug de la recherche                                                                                                           | Au plus tard à l'examen             |
| 6,7    | L'achat, les clients gèrent les timeout et les doublons, le serveur traite les doublons<br>, Le calcul du crc                               | Au plus tard à l'examen             |
|        | Remise de votre dossier Lib avec toutes vos étapes par mail à herman.vanstapel@hepl.be. <b>Pénalité si recopie ou non remise du dossier</b> | Au plus tard avant l'examen         |

**(\*) La planning est à titre indicatif, il faut retenir qu'on peut présenter deux points par semaine au laboratoire**

**Si le programme n'est pas personnalisé comme demandé, champ supplémentaire, c'est d'office 0/20**

**Je peut demander de faire des modifications**

## Grille d'évaluation pour les Deuxièmes indus

L'évaluation est continue pour les trois premiers points de labo.

Les autres points peuvent être présentés au plus tard à l'examen.

| Numéro | Description                                                                                                                                 | A présenter<br>Semaine du (*)          |
|--------|---------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------|
| 1      | Le programme Admin, champ Personnel & Ecrire la fonction recherche                                                                          | 21/3<br><br>A présenter avant l'examen |
| 2      | Un Client et un seul serveur la requête recherche                                                                                           | 11/4<br><br>A présenter avant l'examen |
| 3      | Un Serveur et plusieurs clients                                                                                                             | 18/4<br><br>A présenter avant l'examen |
| 3B     | Un Serveur Affichant l'ip & port de ses clients                                                                                             | 25/4<br><br>Au plus tard à l'examen    |
| 4A     | Le programme admin gérant l'achat et la génération de factures                                                                              | 2/5<br>Au plus tard à l'examen         |
| 4B     | l'achat et la génération de factures par réseau                                                                                             | 9/5<br>Au plus tard à l'examen         |
| 5A     | L'achat, les clients gèrent les timeout et les doublons, le serveur ne gère pas les doublons                                                | 16/5<br>Au plus tard à l'examen        |
| 5B     | Corrigeons le bug de la recherche                                                                                                           | Au plus tard à l'examen                |
| 6,7    | L'achat, les clients gèrent les timeout et les doublons, le serveur traite les doublons<br>, Le calcul du crc                               | Au plus tard à l'examen                |
|        | Remise de votre dossier Lib avec toutes vos étapes par mail à herman.vanstapel@hepl.be. <b>Pénalité si recopie ou non remise du dossier</b> | Au plus tard avant l'examen            |

**(\*) La planning est à titre indicatif, il faut retenir qu'on peut présenter deux points par semaine au laboratoire**

**Si le programme n'est pas personnalisé comme demandé, champ supplémentaire, c'est d'office 0/20**

**Je peut demander de faire des modifications**

## Grille d'évaluation pour les Deuxièmes Gestion