

Simulation d'un système de validation de tests Covid-19 PCR

TABLE DES MATIÈRES

| | |
|---------------------------------------------------------------|-----------|
| TABLE DES MATIÈRES | 1 |
| Introduction | 2 |
| Mode d'emploi | 3 |
| Compilation de l'application | 3 |
| Fichiers indispensables | 3 |
| Exécution de l'application et fonctionnement de l'application | 3 |
| Manuel Technique | 4 |
| Processus Terminal | 4 |
| Processus Validation | 5 |
| Processus Acquisition | 7 |
| Processus InterArchive | 9 |
| Conclusion | 10 |

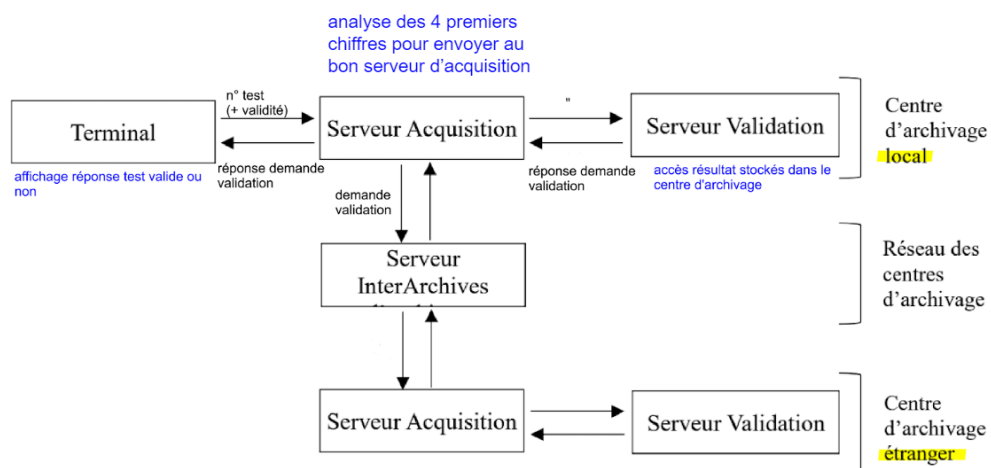
Introduction

La crise sanitaire actuelle a changé très radicalement notre mode de vie. Des règles sanitaires plus strictes ont été mises en vigueur et notamment dans les transports en commun. Les procédures de déplacement en avion ont été modifiées: il faut maintenant avoir réalisé un test PCR au préalable avant de pouvoir monter dans l'appareil. Ce test doit ensuite être vérifié par l'ordinateur afin de déterminer sa validité.

Informatiquement parlant, l'ordinateur doit envoyer des demandes et en recevoir les réponses. Les messages sont sous la forme `|Numéro_du_test|Type|Valeur|`. Le numéro du test PCR est sur 16 chiffres où les 4 premiers indiquent si le test a été réalisé localement ou à l'étranger. Le type du message permet d'indiquer si c'est une demande ou une réponse. La valeur du message correspond à la date de validité du test dans le cas d'une demande ou un booléen (0 ou 1) qui permet de savoir si le test est valide dans le cas d'une réponse.

Les demandes sont envoyées au serveur d'acquisition par le terminal. Le serveur d'acquisition va ensuite servir de routeur car les tests PCR peuvent avoir été effectués nationalement ou encore à l'étranger. Selon l'endroit où a été effectué le test, la demande va être envoyée à un serveur différent. En effet, si le test est local, la demande va être envoyée directement au serveur de validation. Dans l'autre cas de figure, la demande va d'abord être envoyée à un serveur InterArchive puis au serveur de validation.

Les réponses vont être générées suivant 2 critères: la durée de validité du test et le résultat du test. Le serveur de validation va s'occuper de vérifier ces 2 critères pour ensuite créer le message de réponse. Si la demande est validée, la valeur de fin de la réponse sera 1, dans le cas contraire, la valeur de fin sera 0. Cette réponse fera le chemin inverse de la demande jusqu'au Terminal où elle pourra être affichée.



Mode d'emploi

Compilation de l'application

Afin de compiler l'application il suffit de rentrer la commande *make* dans un terminal positionné dans le dossier PCR. Afin de nettoyer tous les fichiers exécutables et fichiers objet il faut utiliser la commande *make cleanall*.

Fichiers indispensables

Pour utiliser l'application il faut au préalable posséder au moins 3 fichiers :

- Un fichier de configuration (exemple présent dans l'archive : *centre_archivage.txt*). Ce fichier conservera la liste des différents serveurs d'acquisition a pour forme :
Numéro_du_Centre(4chiffres) *Nombre_de_Terminaux* *Fichier_test_PCR*
Nom_Centre.
- Un fichier contenant la liste de tous les numéros de test PCR possédant l'intitulé : *Liste_test.txt*. Ce fichier est utile pour la démonstration d'un tirage aléatoire d'un test PCR.
- Au moins un fichier contenant les résultats des tests PCR. Les fichiers seront de la forme *Numéro_du_Test_PCR* *Date_Du_Test* (TimeStamp Unix) *Résultat* (1 = positif, 0 = négatif). Ces fichiers sont utilisés dans le fichier de configuration (*Fichier_test_PCR*). Il est conseillé de posséder autant de fichier de résultat que de centres d'acquisition (exemple présent dans l'archive : *Pcr1.txt* et *Pcr2.txt*).

Exécution de l'application et fonctionnement de l'application

Dans le terminal, il vous suffira d'exécuter la commande suivante: `./interarchives [<Taille mémoire>] [<Nom fichier configuration>]`. Ce terminal sert d'interface à InterArchive, toutes les requêtes seront affichées avec la destination et la source si c'est une demande.

Plusieurs fenêtres de *xTerm* vont s'afficher :

- Les fenêtres Acquisition, préciserons le numéro, le nom et le fichier dans lequel sont stockés les résultats du centre d'acquisition. Comme pour InterArchive le terminal affiche toutes les requêtes avec la destination et la source si c'est une demande.
- Les fenêtres du processus Terminal affichent une demande aléatoire tirée de *Liste_test.txt* ainsi que la réponse reçue. Puis elles attendent la rentrée manuelle d'une nouvelle demande de test.

Pour terminer l'application, il faut fermer les terminaux, soit en tapant *exit*, soit en fermant la fenêtre. Une fois tous les Terminaux d'un serveur d'acquisition fermé, il faut aussi taper *exit* dans le bash, afin de finir le processus Validation et libérer la mémoire. Une fois tous les serveurs d'acquisition fermés, le processus d'InterArchive se fermera tout seul.

Manuel Technique

Processus Terminal

Le Terminal a pour rôle d'envoyer les demandes et d'afficher les réponses obtenues. Dans le cadre de ce projet, les numéros de tests PCR seront simulés par un exécutable, directement inclus dans le processus Terminal, qui tirera aléatoirement un numéro de test existant dans le fichier *Liste_test.txt* avec la fonction *open()*.

Pour ce faire, nous avons redirigé l'entrée et la sortie vers des descripteurs de fichiers passés en paramètres grâce à la fonction *dup2()*.

Grâce à la fonction *createMsg()*, nous allons créer un message dans le format énoncé précédemment avec le numéro de test entré en paramètre. Les messages créés dans le Terminal seront forcément des demandes, la fonction ne peut donc créer que des messages de type "Demande". Afin d'obtenir un message complet, nous lui attribuons une durée de validité aléatoire grâce à la fonction *alea()*¹.

Suite à la génération du message, le message est écrit sur la sortie redirigée en utilisant la fonction *ecritLigne()*. Cela permet aux autres serveurs de récupérer les demandes envoyées et de les traiter par la suite.

Le Terminal a aussi pour rôle de traiter les réponses qu'il reçoit par le descripteur de fichier d'entrée. Nous utilisons *litLigne()*² afin de pouvoir récupérer une réponse et l'analyser. Le message est découpé en 3 parties grâce à la fonction *decoupe()*³ afin de pouvoir utiliser les informations dont nous avons besoin. Nous utilisons la valeur retournée par *analyseValeur()* pour vérifier si la réponse est validée ou refusée. La valeur de réponse est 1 pour dire que la demande est validée et 0 pour dire qu'elle est refusée. Si la valeur de la réponse est différente de ces 2 cas, le Terminal va afficher "Valeur non reconnue".

Une fois la première réponse du message aléatoire reçue, le Terminal passe en mode manuel en lisant les messages sur un descripteur de fichier qui est un duplicata de *stdin*. Afin de stopper le processus, il faut rentrer la commande *exit* dans le terminal.

Pour tester le processus indépendamment des autres fichiers, il suffit d'entrer "make terminal.c" puis "./terminal" suivis des noms des descripteurs de fichiers voulus. Le premier argument suivant "./terminal" est le descripteur de fichier correspondant à l'entrée et le second correspond à celui de la sortie. Pour illustrer cela, nous avons passé 0 en entrée pour lire les réponses à partir du clavier et 1 en sortie pour afficher les demandes à l'écran.

¹ Fonction fournie dans le fichier *alea.c*: Retourne un nombre aléatoire en min et max (bornes comprises)

² Fonction fournie dans le fichier *lectureEcriture.c*: Retourne la ligne lue en char*

³ Fonction fournie dans le fichier *message.c*: Découpe du message

```
pogo@pogo:~/Charlotte/Documents/E3/IGI/Système d'exploitation/S2/PCR/PCR$ make terminal
gcc -Wall -c terminal.c
gcc terminal.c message.o alea.o lectureEcriture.o -o terminal
pogo@pogo:~/Charlotte/Documents/E3/IGI/Système d'exploitation/S2/PCR/PCR$ ./terminal 0 1
print : Voici le message : |0001000000000000|Demande|9462|
|0001000000000000|Demande|9462|
```

Il nous est ensuite demandé d'entrer un nouveau message qui sera une réponse. La réponse n'est pas forcément celle associée au message de demande que le Terminal a généré, le processus va uniquement analyser la valeur de la réponse qu'il reçoit.

```
pogo@pogo:~/Charlotte/Documents/E3/IGI/Système d'exploitation/S2/PCR/PCR$ ./terminal 0 1
print : Voici le message : |0001000000000000|Demande|20178|
|0001000000000000|Demande|20178|
|0001000000000001|Reponse|1|
Validé
```

```
pogo@pogo:~/Charlotte/Documents/E3/IGI/Système d'exploitation/S2/PCR/PCR$ ./terminal 0 1
print : Voici le message : |0001000000000000|Demande|49152|
|0001000000000000|Demande|49152|
|0001000000000001|Reponse|0|
Refusé
```

```
pogo@pogo:~/Charlotte/Documents/E3/IGI/Système d'exploitation/S2/PCR/PCR$ ./terminal 0 1
print : Voici le message : |0001000000000000|Demande|33759|
|0001000000000000|Demande|33759|
|0001000000000000|Reponse|4|
print : Valeur non reconnue.
```

Processus Validation

Le serveur de validation est un serveur qui va traiter les demandes reçues et qui va en créer les réponses associées. Il a un accès aux résultats stockés dans le fichier passé en paramètre. Il a donc pour rôle de vérifier que le numéro de test figure bien dans sa liste, que le test est récent et que le résultat est bien négatif.

De manière similaire au processus Terminal, nous avons redirigé son entrée et sa sortie vers des descripteurs de fichiers. Le descripteur de fichier en entrée possède les demandes que le serveur doit lire. Le descripteur de fichier en sortie est utilisé pour écrire les messages de réponses que le processus va créer. Nous avons également séparé les messages en 3 parties pour les utiliser par la suite.

Nous avons créé un fichier *Pcr1.txt* qui contient les numéros de tests locaux, commençant par 0001, associés à leur durée de validité en seconde et à son approbation. Le fichier *Pcr2.txt* est similaire au précédent, il contient les numéros de test réalisés à l'étranger. Les 4 premiers numéros des tests étrangers commencent par 0002.

```
PCR > Pcr1.txt
1 0001000000000000 1625888897 1
2 0001000000000001 1625294720 0
3 0001000000000002 1625294629 0
4 0001000000000003 1625294718 1
5 0001000000000004 1625294718 1
```

```
PCR > Pcr2.txt
You, 3 hours ago | 1 author (You)
1 0002000000000000 1625888897 1
2 0002000000000001 1625294720 0
3 0002000000000002 1625294629 0
4 0002000000000003 1625294718 1
5 0002000000000004 1625294718 1
```

Afin de pouvoir vérifier si la demande est bien valide, chaque Validation possède son propre fichier texte. Si les tests sont positifs, la valeur de fin sera un 1 et la demande sera alors refusée. Dans le cas contraire, nous enverrons une réponse avec une valeur égale à 0 pour dire que le test est bien négatif et que la demande est donc bien validée. Afin d'éviter l'erreur d'exécution du "core dumped", nous avons modifié la fonction *litLigne()* dans le fichier *LectureEcriture.c*. Nous avons rajouté une partie de code qui nous permet de savoir quand la lecture du fichier entré en paramètre est terminée, la fonction nous retourne "erreur".

```
char * litLigne(int fd)
{
    int i = 0, err = 0;
    char *c = malloc(TAILLEBUF+1);
    do {
        if( ((err = read(fd, &c[i],1)) <= 0) || (i == TAILLEBUF)){
            free(c);

            // read a positionn l'erreur si il y en a une dans le read
            // Sinon, c'est que l'on dépasse TAILLEBUF
            if (i == TAILLEBUF){
                errno = EIO; // ligne trop grande
                return NULL;
            } else {return "erreur";}
        }
        i++;
    } while(c[i-1] != '\n');
    c[i] = '\0';
    return c;
}
```

Le serveur de Validation doit lire en continu les demandes qu'il reçoit. Il lit la demande sur son entrée puis la découpe. Pour vérifier si le test est valide, on parcourt le fichier Pcr.txt du centre et on compare les numéros. Lorsqu'il aura bien vérifié que le numéro du test est bien existant, il va ensuite récupérer la durée de validité pour pouvoir l'analyser avec la fonction *validation()*. Si le temps obtenu à l'aide de *time(NULL)* est inférieur au temps dans Pcr.txt + le temps de validité stockés dans la demande, alors on regarde si le test est bien négatif (0). Le message de réponse va ensuite être créé et envoyé sur la sortie.

Pour tester ce processus de manière indépendante, il suffit de rentrer "make validation" puis "./validation" suivis de 3 descripteurs de fichier. Pour illustrer cette exécution, nous avons mis 0, 1 et *Pcr1.txt* afin d'afficher les résultats à l'écran et de lire les demandes au clavier. Le troisième argument correspond au fichier vers lequel le processus va vérifier la validité des tests.

```
etienne@etienne-VirtualBox:~/Documents/Projet 3002/PCR$ ./validation 0 1 Pcr1.txt
les deux sorties : 0 , 1
|0001000000000004|Demande|1780|
|0001000000000004|Reponse|0|
|0001000000000004|Demande|17800|
|0001000000000004|Reponse|1|
```

```

Pcr1.txt
You, 2 hours ago | 1 author (You)
1  0001000000000000 1618251737 1
2  0001000000000001 1618251737 0
3  0001000000000002 1618251737 0
4  0001000000000003 1618251737 0
5  0001000000000004 1618251737 0
6  0001000000000005 1618251737 0
```

Dans cet exemple, le fichier Pcr1.txt nous indique la date à laquelle le test a été réalisé. Ce temps a été généré en seconde grâce à la fonction `time()` qui indique les secondes écoulées depuis le 1er janvier 1970 à 00:00:00. En convertissant ce temps, nous obtenons la date à laquelle le test a été réalisé: le 12 avril 2021 à 18:22:17. La dernière valeur indiquée sur le fichier `Pcr1.txt` nous indique si le test réalisé est positif (1) ou négatif (0).

En ayant exécuté la commande le 12 avril à 22:00, la première demande est refusée car sa durée de validité de 1780 secondes équivaut à environ 30 minutes. Le temps de validité étant dépassé, la valeur de la réponse est donc refusée (0). Dans la seconde demande, le temps de validité équivaut environ à 4h. Cela signifie que le test n'est plus valable à partir de 22:22. La durée de validité étant encore valable dans notre cas, la réponse à la demande est donc positive.

Processus Acquisition

Le serveur Acquisition a pour rôle de recevoir et de transmettre les requêtes qu'il reçoit. Il reçoit les demandes des serveurs de Terminal et d'InterArchive et les réponses de Validation et d'InterArchive.

Pour ce processus, nous avons utilisé les thread pour paralléliser les requêtes. Nous avons aussi créé une structure pour pouvoir passer plusieurs arguments aux threads qui seront deux descripteurs de fichier. Chaque thread correspond à un processus: un thread pour Validation, un pour InterArchive et un pour chaque serveur Terminal. Pour chaque thread, nous créons une paire de tubes qui seront passés dans les divers arguments.

Tout d'abord, un processus Validation ou Terminal, est créé via `execvp()` et prend en paramètre deux descripteurs de fichiers: un pour l'entrée et un autre pour la sortie. Lors de la création du thread, deux autres descripteurs de fichiers sont passés en paramètre.

Une fois tous les threads initialisés, nous attendons la fin de tous les processus Terminal afin de `cancel()` tous les thread et de les mettre en attente afin de pouvoir `exit()` proprement le processus. Nous n'arrêtons pas le processus immédiatement car des demandes peuvent encore être validées sur le serveur. Acquisition va attendre de recevoir le message `exit` via

stdin pour tuer le processus Validation et annuler les deux derniers threads. Puis, nous libérons la mémoire.

Deux sémaphores ont été créés, un pour la place en mémoire (*s_memoire*), et l'autre lors du changement d'une partie du tableau mémoire qui est partagée entre les threads (*s_indice*). Pour garder en mémoire la provenance des demandes, nous utilisons une table de routage qui est composée d'un tableau simple d'*int* en 2D ainsi que d'un tableau de *long long int*. Le tableau simple permettra de stocker les numéros des tests PCR. Le tableau en 2D stockera dans la première ligne un booléen permettant de savoir si la place est libre ou non, la deuxième ligne permet de savoir le descripteur de fichier retour associé.


Lors de la réception d'une demande, le thread va appeler *sem_wait(&s_memoire)* pour voir s'il y a encore de la place en mémoire. Le thread entrera en section critique, il cherchera une place de libre dans le tableau de mémoire partagée: si la case vaut 1, la place est occupée, dans le cas contraire, la case vaudra 0. Les descripteurs de fichier et le numéro du test PCR seront ensuite placés dans un emplacement vide, la valeur de la case passera donc de 0 à 1 avant de sortir de la section critique.

Si la requête est une réponse, nous allons la router vers le serveur Terminal ou le serveur Validation. Afin de ne pas prendre une ancienne demande ayant le même numéro de test PCR, nous devons vérifier la présence de ce numéro de test et la valeur de sa case (1) dans les deux tableaux lors de la réception d'une réponse. Nous libérons la place dans les tableaux pour finir sur le *sem_post(&s_memoire)* qui laisse la place à une nouvelle demande.

Le serveur Acquisition devant faire le routage entre les serveurs Terminal et Validation, il faut compiler ces deux derniers avant de pouvoir compiler et exécuter Acquisition. Pour l'exécution d'Acquisition, il faut entrer *./acquisition* suivis de la taille mémoire voulue, le nom du centre d'archivage voulue (en majuscule), les 4 premiers chiffres des tests de ce centre d'archivage, le nom du fichier où sont stockés les résultats PCR, le nombre de terminal, le descripteur de fichier correspondant à l'entrée et à la sortie d'InterArchive. Suite à l'exécution, des fenêtres correspondant aux nombres de Terminaux demandés vont apparaître.

Pour illustrer cette exécution, nous avons entré la commande *./acquisition 6 PCR0001 0001 Pcr1.txt 2 0 1*". Avec cette commande, nous simulons les descripteurs de fichiers pour InterArchive par le clavier en entrée et l'affichage de la console Linux en sortie. Nous pouvons voir que 2 Terminaux se sont ouverts. Dans la console Linux, il nous est demandé de taper la réponse au clavier, puis la console répondra au Terminal.


```
etienne@etienne-VirtualBox:~/Documents/Projet 3002/PCR$ ./acquisition
usage : ./acquisition [<Taille mémoire>] [<Nom centre archivage>] [<Code de 4 chiffres>] [<Nom fichier résultats test PCR>] [Nombre terminal] [<Entrée Inter Archives>] [<Sortie Inter Archives>]
etienne@etienne-VirtualBox:~/Documents/Projet 3002/PCR$ ./acquisition 6 PCR0001 0001 Pcr1.txt 2 0 1
Centre d'archivage numéro : 0001 et de nom : PCR0001 crée, Résultat sotcké dans : Pcr1.txt
Demande n°0002000000000010, transmie à Inter_Archives de 8
|0002000000000010|Demande|1113342|
Demande n°0001000000000006, transmie à Validation de 12
Reponse n°0001000000000006, transmie à 12
|0002000000000010|Reponse|1|
Reponse n°0002000000000010, transmie à 8
|
```



Processus InterArchive

Le processus InterArchive sert de passerelle entre les serveurs Acquisitions. Il reçoit les requêtes d'un serveur pour ensuite les router au bon serveur Acquisition en analysant les 4 premiers chiffres du numéro des requêtes.

Le fonctionnement de ce processus est similaire à celui d'Acquisition. Nous créons autant de thread qu'il y a de processus Acquisition. Comme dans Acquisition, nous créons une paire de tube et donnons aux processus et aux threads les descripteurs de fichier dont ils auront besoin. Mais contrairement à Acquisition, une fois tous les processus d'Acquisition finis, InterArchive va directement annuler tous ses threads et libère la mémoire.

La gestion de la mémoire et des sémaphores est exactement la même que dans Acquisition. Il en est de même pour la gestion de la table de routage des demandes. La seule différence est que si une demande ne possède pas de centre d'archivage, alors elle ne sera pas ajoutée à la table de routage et renverra une réponse négative.

Les ajouts dans InterArchive sont présents dans la lecture du fichier de configuration *centre_archivage.txt* qui est de la forme : *Numéro_du_Centre Nombre_de_Terminaux Fichier_test_PCR Nom_Centre*.

```
≡ centre_archivage.txt

1  0001 1 Pcr1.txt Pcr1
2  0002 1 Pcr2.txt Pcr2
3
```

Pour chaque ligne de ce fichier de configuration, Interarchive va créer un nouveau processus Acquisition avec les bonnes informations. De plus, nous ajoutons le numéro du centre et le descripteur de fichier associé dans un tableau 2D qui nous servira à router les demandes vers les bons centres d'Acquisition.

InterArchive étant le programme principale, on ne peut pas le tester indépendamment car cela revient à lancer le programme comme décrit dans le mode d'emploi.

Conclusion

Ce projet a été bénéfique dans la compréhension de l'utilisation du langage C. Nous avons pu acquérir de meilleurs réflexes de programmation en C, notamment dans l'utilisation des `calloc`. Le sujet du projet étant très vaste, nous avons eu des difficultés à cerner le sujet et à comprendre les attentes. Une fois le sujet compris, nous avons réussi à démarrer le processus Terminal et Validation. Le processus Acquisition a été la plus grosse difficulté du projet car ce processus était le centre du système de validation de tests Covid PCR. Il regroupe énormément de fonctions que nous avons étudiées pendant les cours. Cela nous a permis d'appliquer ces fonctions sur un gros projet.