Victor HUGER

# Rapport projet EL-3032

# Sommaire :

Victor HUGER

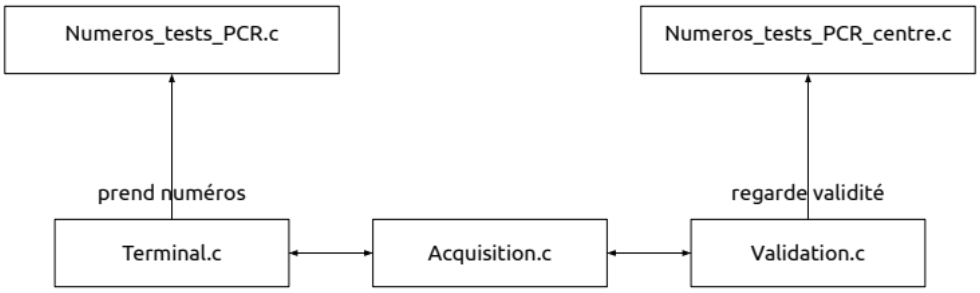
# Architecture

Le projet s'articule autour de cinq programmes : "Numeros_tests_PCR.c", "Numeros_tests_PCR_centre.c", "Terminal.c", "Validation .c" et "Acquisition .c".

Les deux premiers créent de quoi tester la simulation. "Numeros_tests_PCR.c" génère un fichier contenant des numéros de tests. Ce fichier sera utilisé par les terminaux afin de simuler des demandes de validation. "Numeros_tests_PCR_centre.c" génère plusieurs fichiers dans lesquels il place ces numéros de tests et leur ajoute une date ainsi qu'un résultat. Ces fichiers feront office de centre d'archivage et seront donc utilisés par les serveurs de validation pour accepter ou refuser une demande.

Les trois autres programmes créent les différents composants de la simulation. "Terminal.c" génère les terminaux qui font les demandes de validation. "Validation.c" génère les serveurs qui traitent ces demandes. Pour finir, "Acquisition.c" fait le lien entre les différents composants. Il reçoit et envoie les demandes puis il retourne les réponses.

Voici un schéma de l'architecture :




Par manque de temps le projet n'est pas complet, effectivement il manque le serveur *InterArchives*. Ce dernier sert de lien entre les différents serveurs d'acquisitions, il permet de router les demandes de validation et les réponses. Ainsi, avec cette architecture il n'est pas possible de simuler un réseau de centres d'archivage. Seul un centre d'archivage peut-être simulé.

Victor HUGER

# Numeros_tests_PCR.c

Rôle :

Ce processus permet de générer des numéros de tests PCR.

Fonctionnement :

Ce programme requiert deux arguments : le nombre de centres (compris entre un et trois) et le nombre de tests PCR par centre.

Pour simplifier la simulation, seulement trois centres peuvent être créés : Paris, Nice et Madrid. C'est pour cela que le premier argument est un nombre compris entre un et trois. Si on choisit un on aura : Paris, deux : Paris et Nice, trois : Paris, Nice et Madrid. Ces centres ont respectivement pour code : "0000", "1111" et "2222".

Le deuxième argument nous permet d'indiquer le nombre de tests que l'on veut générer par centre. Il n'est pas possible d'avoir un nombre de tests différent entre les centres.

Un test est composé de 16 chiffres, les quatre premiers correspondent au code du centre. Pour les douze autres, la fonction "alea" est utilisée. Cette dernière génère trois nombres aléatoires compris entre 0 et 9999. Des zéros sont ajoutés à la fin s'ils ne sont pas composés de quatre chiffres. Pour finir, le code du centre et les trois nombres sont concaténés pour former le numéro du test.

Ces numéros sont écrits dans un fichier nommé "Numeros_tests_PCR.txt". La première ligne de ce fichier correspond au nombre total de tests créés. Cette information est utile pour les terminaux.

Une fois la génération et l'écriture des numéros de tests terminée, ce programme est recouvert par "Numeros_tests_PCR_centre.c".

Test :

Nous pouvons exécuter "./Numeros_tests_PCR 3 10". Le fichier "Numeros_tests_PCR.txt" apparaît. Dedans nous trouverons 30 numéros de tests, 10 pour Paris, 10 pour Nice et 10 pour Madrid.

Victor HUGER

# Numeros_tests_PCR_centre.c

Rôle :

Ce processus permet de répartir sur différents fichiers les numéros de tests générés par "Numeros_tests_PCR.c". Chaque fichier créé représente un centre d'archivage local. De plus, à chaque test est ajouté une date, représentant le moment où le test a été fait, et un résultat.

Fonctionnement :

Ce programme requiert deux arguments : le nombre de centres (compris entre un et trois) et le nombre de tests PCR par centre. Il vient recouvrir "Numeros_tests_PCR.c" et prend ses arguments puisqu'ils sont identiques.

Ce processus commence par ouvrir "Numeros_tests_PCR.txt". Ensuite il répartit les numéros de tests en utilisant une boucle for avec le nombre de tests PCR par centre (son deuxième argument). Il copie dans un fichier préalablement créé autant de lignes qu'il y a de test PCR par centre. Cette action est répétée, autant de fois qu'il y a de centres, sans fermer le fichier "Numeros_tests_PCR.txt". La première ligne des fichiers créés correspond au nombre de tests PCR par centre. Cette information est utile pour les serveurs de validation.

L'ajout de la date et du résultat se fait dans la boucle for. Pour créer la date, on utilise le timestamp actuel (au moment de l'exécution du fichier) auquel on retire une valeur aléatoire comprise entre 0 et 604 800 (= 7 jours).

Le numéros du test, sa date et son résultat sont écrits sur une même ligne, le tout est séparé par un espace.

Test :

Nous pouvons exécuter "./Numeros_tests_PCR 2 10". En plus de "Numeros_tests_PCR.txt", les fichiers "Numeros_tests_PCR_Paris.txt" et "Numeros_tests_PCR_Nice.txt" apparaissent.

Victor HUGER

# Terminal.c

Rôle :

Ce processus simule un terminal permettant de faire des demandes de validation. On entre un numéro de test PCR et en retour on obtient "Demande acceptée" ou "Demande refusée".

Pour simplifier la simulation, un terminal envoie une seule demande de validation.

Fonctionnement :

Ce programme requiert trois arguments : deux descripteurs de fichiers (un pour l'entrée, un pour la sortie) et le nom du fichier des tests PCR.

Ce processus commence par ouvrir le fichier spécifié en troisième argument. Il lit la première ligne qui correspond au nombre total de tests (confère page 4) et utilise cette donnée pour choisir un test aléatoirement. La validité est arbitrairement prise entre 24 et 168 heures. Il formate ensuite la demande de validation avec la fonction "message" et l'envoie. Pour ce faire, il utilise "ecritLigne" sur le descripteur de fichier correspondant à sa sortie.

Une fois le message envoyé, le programme attend la réponse. Pour ce faire, il utilise "litLigne" sur le descripteur de fichier correspondant à son entrée. Ensuite, avec la fonction "decoupe" il vérifie qu'il s'agisse bien d'une réponse pour le bon numéro de test et retourne "Demande acceptée" ou "Demande refusée".

Test :

Pour tester ce processus, nous pouvons exécuter "./Terminal 0 1 Numeros_tests_PCR.txt". Il faut auparavant avoir appelé le programme "Numeros_tests_PCR.c". Dans la console nous verrons apparaître une demande de validation à laquelle nous pourrons répondre.

Victor HUGER

# Validation.c

Rôle :

Ce processus simule un serveur de validation qui répond aux demandes qu'il reçoit. On entre une demande de validation et en retour on obtient une réponse.

Fonctionnement :

Ce programme requiert trois arguments : deux descripteurs de fichiers (un pour l'entrée, un pour la sortie) et le nom du fichier des résultats des tests PCR.

Ce processus commence par ouvrir le fichier spécifié en troisième argument. Il lit la première ligne qui correspond au nombre de tests par centre d'archivage (confère page 5). Cette donnée sera utilisée plus tard pour parcourir l'ensemble des tests. Ensuite, il attend une demande de validation. Pour ce faire, il utilise "litLigne" sur le descripteur de fichier correspondant à son entrée. Lorsqu'il en reçoit une, il vérifie avec la fonction "decoupe" qu'il s'agit bien d'une demande. Si c'est bien le cas, le programme parcourt l'ensemble des tests à la recherche du numéro qui était dans la demande. Quand il le trouve, il vérifie que le test est encore valable et il regarde son résultat. Pour finir, il envoie sa réponse. Pour ce faire, il utilise "ecritLigne" sur le descripteur de fichier correspondant à sa sortie.

Toutes ces opérations sont effectuées dans une boucle while car le serveur de validation à plusieurs demandes à traiter au cours d'une simulation.

Test :

Pour tester ce processus, nous pouvons exécuter "./Validation 0 1 Numeros_tests_PCR_Paris.txt". Il faut auparavant avoir appelé le programme "Numeros_tests_PCR.c". Dans la console nous pouvons écrire une demande de validation à laquelle le serveur répondra.

Victor HUGER

# Acquisition.c

Rôle :

Ce processus sert de lien entre les terminaux et le serveur de validation, il transmet les demandes de validation et les réponses.

Fonctionnement :

Ce programme existe en trois versions, d'où le fait qu'on trouve les dossiers v1, v2 et v3 dans le projet, la différence entre ces derniers étant "Acquisition.c".

*Première version :*

Cette version requiert deux arguments : la taille de la mémoire (nombre de demandes pouvant être traitées simultanément) et le nom du centre.

Ce processus n'est pas encore relié aux autres composants de la simulation, de ce fait à la place de lire et écrire dans des pipes, il le fait dans des fichiers.

Dans ce programme, il y a une structure appelée "donnees" dans laquelle on peut stocker deux entiers (un pour le descripteur de fichier de l'entrée du terminal et l'autre pour la sortie) et deux chaînes de caractères (une pour la demande et l'autre pour la réponse). La mémoire du processus est un tableau de cette structure.

Le serveur d'acquisition est censé pouvoir transmettre une demande à tout moment, une boucle while a donc été implémentée pour répondre à ce besoin. Cependant cette version étant assez éloignée de la réalité, des conditions d'arrêts ont été mises en place. Dans cette boucle, le terminal, le serveur de validation et le serveur *InterArchives* sont traités séquentiellement. Tout d'abord, pour le terminal, une boucle for parcourt la mémoire, cela permet d'écrire les réponses sur la sortie et de regarder les demandes sur l'entrée. Ensuite, pour le serveur de validation, une autre boucle for parcourt la mémoire et envoie les demandes lorsqu'elles sont à destination du centre d'archivage local. Les réponses sont lues et écrites dans la mémoire. Enfin, pour le serveur *InterArchives*, une dernière boucle for parcourt la mémoire pour traiter les demandes à destination des centres d'archivage distants.

*Deuxième version :*

La deuxième version est semblable à la première sur beaucoup de points. La différence majeure est liée à la présence de threads. Ils sont au nombre de trois : un pour les terminaux, un pour le serveur de validation et un pour le serveur *InterArchives*. Ils intègrent les mêmes boucles for que la première version, avec des sémaphores en plus. L'intérêt des threads se trouve dans le parallélisme qu'ils apportent. Et, pour ne pas qu'il y ait de problèmes, notamment d'interblocage, les sémaphores sont indispensables. Ici il y en a trois : semLibre, semDemandeValidation et semDemandeInterArchives. Le premier est initialisé avec la taille de la mémoire et est utilisé par le thread terminal. Ainsi il ne peut pas écraser des demandes qui n'ont pas encore été traitées, puisqu'une fois qu'il a parcouru la mémoire, il s'arrête. Ce sont les threads validation et interArchives qui libèrent de la place une fois qu'ils ont répondu à une demande. Les

deux autres sémaphores sont initialisés à 0 et sont utilisés par les threads validation et interArchives. Ces derniers attendent que le thread terminal ait écrit en mémoire puis qu'il libère de la place pour pouvoir traiter les demandes.

*Troisième version :*

Cette version requiert cinq arguments : la taille de la mémoire, le nom du centre d'archivage à simuler, le code à quatre chiffres du centre, le nom du fichier des résultats des tests PCR et le nombre de terminaux à créer.

Ce processus intègre la liaison avec les autres composants par le biais de pipes. Il est composé de quatre threads, le thread terminal de la version précédente a été séparé en deux. threadTerminalEntree lit les demandes en provenance des terminaux et threadTerminalSortie écrit les réponses. Le thread lié au serveur *InterArchives* n'est pas fonctionnel car cette partie du projet n'est pas encore simulée.

L'enjeu de ce programme est la gestion des multiples terminaux. Pour ce faire, threadTerminalEntree est dupliqué autant de fois qu'il y a de terminaux. Ainsi, chaque instance de ce thread est associée à un terminal.

Un autre point important est la gestion des descripteurs de fichiers sur lesquels les threads lisent et écrivent. Une structure appelée "ES" est utilisée. Elle est composée de deux entiers, un pour le descripteur de fichier de l'entrée du thread et l'autre pour la sortie. Une variable ayant pour type cette structure est passée à threadTerminalEntree et threadValidation lors de leur création. Pour savoir où écrire, threadTerminalSortie regarde la mémoire. Effectivement, pour chaque demande reçue threadTerminalEntree écrit dans la mémoire où il faut répondre.

Pour ce qui est des interblocages entre les threads, il ne peut y en avoir grâce aux sémaphores. Il y en a trois : semSortie, semEcrit et semDemandevalidation. semEcrit est initialisé à 1 et est utilisé pour que l'écriture dans le mémoire ne puisse être faite que par un thread à la fois. semDemandevalidation est initialisé à 0, threadValidation attend que threadTerminalEntree libère de la place car cela signifie qu'il y a une demande à traiter. semSortie est lui aussi initialisé à 0, threadTerminalSortie attend que threadValidation libère de la place car cela signifie qu'il y a une réponse à envoyer. Ainsi, threadTerminalEntree attend des demandes, lorsqu'il en reçoit il indique à threadValidation qu'il y en a à traiter et une fois fait, threadValidation indique à threadTerminalSortie qu'il peut envoyer les réponses. Cette approche peut paraître séquentielle mais ce n'est pas le cas puisqu'il y a plusieurs instances de threadTerminalEntree.

Test :

*Première version :*

Dans le fichier "Terminal_Acquisition.txt" quatre demandes ont été écrites. Les réponses à ces demandes se trouvent dans les fichiers "Validation_Acquisition.txt" et "InterArchives_Acquisition.txt". Si on exécute "./Acquisition 5 Paris" les fichiers "Acquisition_Terminal.txt" (réponses), "Acquisition_Validation.txt" (demandes) et "Acquisition_InterArchives.txt" (demandes) devraient se remplir.

*Deuxième version :*

Pour tester la deuxième version, la manipulation est la même que pour la première.

*Troisième version :*

Pour tester ce processus, nous pouvons exécuter "./Acquisition 5 0000 Paris Numeros_tests_PCR_Paris.txt 2". Il faut auparavant avoir appelé le programme "Numeros_tests_PCR.c". Nous verrons apparaître deux fenêtres xterm, dans chacune d'elles il y aura une demande, une réponse et le résultat.

## Interblocage entre les processus

Les programmes de ce projet sont conçus de telle sorte qu'il ne peut pas y avoir d'interblocage. En effet, le processus terminal envoie une demande. Cette dernière est transférée par le serveur d'acquisition au serveur de validation. La demande est traitée puis elle est renvoyée au bon terminal. Le goulot d'étranglement est "Acquisition.c" puisqu'il transfère les demandes et les réponses. L'ensemble des messages transite donc par ce processus. Or nous avons précédemment montré qu'il ne pouvait y avoir d'interblocages au sein de ce dernier. Le projet devrait donc toujours s'exécuter sans qu'aucuns blocages ne surviennent.