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

Projet EL-3032 – 2021

Ce sujet est une contextualisation d'un projet proposé par **Jean Cousty** et **Laurent Najman**

Avertissement

Le projet suivant est à faire par groupe de deux étudiants. Tous les projets « semblables » auront une note de zéro, les auteurs seront convoqués en conseil de discipline. Ceux qui arriveront à faire la preuve qu'ils ont écrit le projet duquel les autres se sont inspirés n'auront aucune pénalité supplémentaire par rapport à la note de zéro. Si vous faites le projet à plus de deux, un coefficient de réduction proportionnel sera appliqué. Par exemple, si vous faites le projet à trois, la note sera réduite d'un tiers. Aucun bonus ne sera donné pour un étudiant travaillant seul.

Plagiat

Le plagiat est l'utilisation, sans citation appropriée, du travail intellectuel d'une autre personne dans un travail soumis à évaluation. Ce qui suit est très fortement inspiré et traduit de la page http://www.csd.abdn.ac.uk/teaching/handbook/both/info.php?filename=cheating.txt.

Quand vous écrivez un rapport ou du code qui contient des parties ou qui paraphrase le travail d'autres personnes, vous devez clairement le signaler. En particulier, les citations doivent être données entre guillemets, avec les références appropriées.

1. Le code soumis pour évaluation doit clairement être annoté avec le nom de l'étudiant qui a soumis l'exercice, avec la date de rédaction.
2. Quand un exercice contient du code qui n'a pas été écrit par l'étudiant qui soumet le travail, ou contient du code écrit par l'étudiant lui-même à un autre moment, le code en question doit être clairement identifié et annoté avec le nom de l'auteur, le copyright (si différent), la date d'achèvement ou de publication, et une référence à la source (par exemple une URL ou une référence bibliographique).
3. En principe, la discussion avec d'autres étudiants du contenu du cours et des exercices non-évalués est encouragée, car une telle discussion amène généralement à une meilleure compréhension du sujet. Cependant, les discussions doivent rester à un niveau général, et ne doivent pas descendre à un niveau détaillé de conception ou de codage. **Le travail soumis pour évaluation doit être un travail individuel**. Si vous soumettez un travail produit conjointement avec une autre personne, ou qui inclus le travail de quelqu'un d'autre, **ceci doit être clairement indiqué, et le code en question doit être clairement identifié**. L'évaluation du travail se fera sur la base de la valeur ajoutée par l'étudiant.

4. De manière identique, bien que nous encouragions la réutilisation de logiciel existant comme une bonne pratique d'ingénierie Logicielle, les origines d'une telle réutilisation **doivent être clairement indiquées, et le code identifié**. L'évaluation du travail se fera sur la base de la valeur ajoutée par l'étudiant.
5. Quand le travail est à réaliser en groupe de deux étudiants, le rapport écrit ou le travail doit clairement identifier et distinguer ce qui a été réalisé par un des étudiants, ou ce qui a été fait par le groupe en entier.

Modalité d'évaluation des compétences acquises

- Vous devez créer un dépôt `GitHub`[1] qui va contenir votre projet avant **mardi 13 avril 2021 à 8H** (date de la dernière séance du TD). A cette date, votre dépôt devra contenir le code qui sera évalué (sans les fichiers exécutables ou objet), le makefile, le rapport du projet sous format électronique 'pdf' et le mode d'emploi.
- Le projet devra se compiler en exécutant la commande `make` sur une machine tournant Linux. **Un projet qui ne respectera pas cette consigne ne pourra pas être corrigé.**
- Le rapport du projet décrira les diverses solutions aux différents problèmes rencontrés, et **justifiera** la solution choisie.
- Votre évaluation dépendra de la qualité du rapport, et de la qualité du code.
- Les compétences à acquérir (et donc l'évaluation) comprennent une partie liée à la rédaction et à la justification des choix retenus, et également une partie liée à l'écriture du code correspondant aux choix que vous aurez faits. Vous pourrez donc valider l'acquisition de compétences si vous n'avez pas programmé vos propositions, mais vous n'aurez pas forcément validé l'acquisition de toutes les compétences si le code que vous avez écrit n'est pas proprement justifié.
- Les évaluations auront lieu mardi, le **13 avril 2021** entre **8H** et **12H**.

Remarque

L'énoncé peut sembler épais de prime abord. Ce n'est pas le cas : il se résume aux pages 4 à 11. Le reste du document a pour objectif de vous aider dans la réalisation du projet, en indiquant des éléments de solutions, en attirant l'attention sur les points essentiels et en donnant une démarche saine de progression.

[1] Voir la section 5.1 en annexe pour plus de détails sur les dépôts Git et GitHub.

Table des matières

1. Introduction

L'objectif du projet est de simuler les échanges entre des centres d'archivage des résultats de tests PCR, permettant à un agent de l'aéroport de vérifier la validité d'un test PCR d'un voyageur, même si ce test est réalisé à l'étranger[2]. Avant de présenter le sujet, examinons le fonctionnement de la validation des tests au niveau de l'aéroport.

1.1. Le principe de la validation des tests PCR

La procédure de validation des tests PCR à l'aéroport met en relation plusieurs acteurs :

- Le *voyageur*, qui présente son test PCR à un agent spécialisé de l'aéroport (un agent de validation des tests PCR) ;
- L'*agent de validation*, qui est équipé d'un terminal de validation (ordinateur), qui sert à vérifier si le test présenté par le voyageur est valide ou non ;
- Le *centre d'archivage local*, auquel est connecté le terminal de validation. Il héberge tous les résultats des tests réalisés dans le pays où se trouve l'aéroport.
- Des centres d'archivage étrangers, hébergeant les résultats des tests réalisés à l'étranger.

Le terminal de validation est connecté au centre d'archivage local via internet. Le centre d'archivage local est relié à tous les autres centres d'archivage installés dans le monde grâce à un réseau dédié : le réseau des centres d'archivage[3] (voir la figure 1).

Supposons maintenant que le voyageur se rend à l'aéroport afin de prendre l'avion pour ça destination. À son entrée à l'aéroport (ou au moment de l'enregistrement), il présente son test PCR à l'agent de validation, qui va saisir (ou scanner) la référence (numéro) du test au terminal de validation.

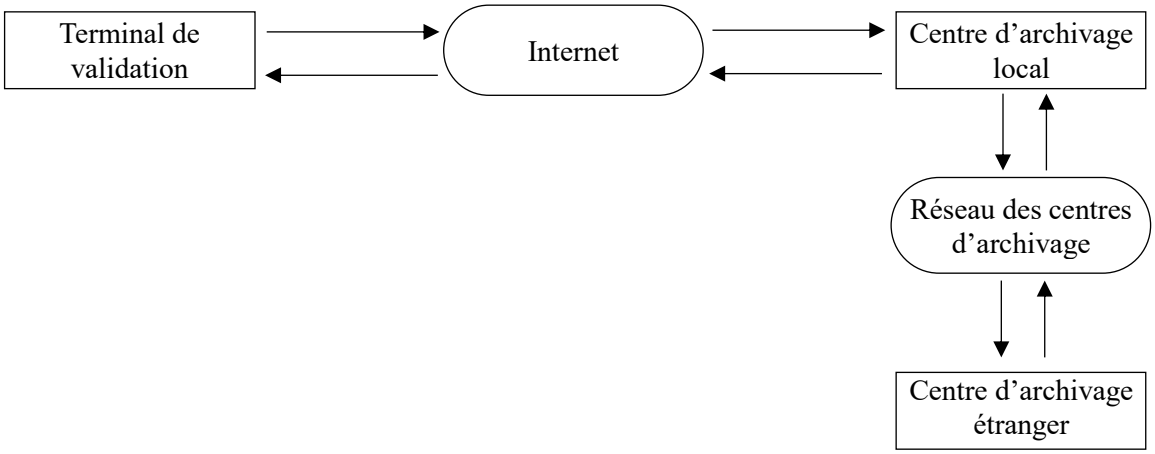

Figure 1. Principe de communication des demandes de validation et des différents réseaux

[2] On suppose qu'un test réalisé à l'étranger est accepté, à condition que sa date de validité ne soit pas dépassée.
[3] Le réseau des centres d'archivage peut être un réseau physique, comme il peut être un réseau virtuel.

Les opérations suivantes auront lieu :

1. Le terminal de validation se connecte au centre d'archivage local et envoie le numéro de test ainsi que la durée de validité des tests PCR[4].
2. Le centre d'archivage local regarde le numéro de test, se rendant compte qu'il ne s'agit pas d'un test réalisé en France, envoie le numéro de test au centre d'archivage étranger, via le réseau reliant les différents centres d'archivage.
3. Le centre d'archivage étranger prend connaissance du numéro de test et vérifie que le résultat du test est bien **négatif**, et que le test est **récent** (datant de moins d'une durée bien déterminée).
4. Si c'est le cas, il répond au centre d'archivage local (toujours via le réseau des centres d'archivage) que le test est valide. Si ce n'est pas le cas, il répond le contraire.
5. Enfin, le centre d'archivage local transmet la réponse au terminal de validation.
6. Le test est validé ou refusé.

1.2.La demande de validation

La suite des opérations décrites ci-dessus se nomme la « demande de validation » et a essentiellement pour but de vérifier que le test présenté par le voyageur est récent et son résultat est bien négatif. Cette suite d'opération montre que le rôle des centres d'archivage est double. En effet, l'action du centre d'archivage diffère selon si le test est réalisé en local ou non. **i)** Si ce n'est pas le cas, son rôle est d'acheminer la demande vers le centre d'archivage qui a la capacité de la traiter (*cf.* étape 2 ci-dessus). **ii)** Si le test est réalisé en local, son rôle est de vérifier la validité du test (*cf.* étape 4 ci-dessus) et de produire une réponse qui est ensuite acheminée vers le terminal de validation depuis lequel la demande a été émise.

Chaque centre d'archivage est donc composé de deux serveurs.

Le serveur d'acquisition. Il s'agit du serveur du centre d'archivage local auquel se connecte le terminal de validation via Internet. Une fois connecté, le terminal envoie au serveur d'acquisition le numéro du test et la durée de validité. Le serveur d'acquisition a ensuite pour rôle d'acheminer les données vers un autre serveur capable de les traiter.

Le serveur de validation. Il s'agit du serveur du centre d'archivage auquel le serveur d'acquisition transmet la demande de validation de test émise par le terminal. C'est lui qui est chargé de produire la réponse à la demande de validation. Cette réponse suit donc le chemin inverse, c'est à dire : serveur de validation du centre d'archivage hébergeant le test du voyageur → serveur d'acquisition du centre d'archivage local → terminal de validation.

[4] On suppose que différentes destinations peuvent exiger des durées de validité de tests différentes.

1.3. Le routage

Pour effectuer le routage des demandes de validation, c'est-à-dire pour déterminer à quel centre d'archivage chaque demande de validation doit être transmise, le serveur d'acquisition utilise les premiers numéros de chaque test concerné : ceux-ci indiquent le centre d'archivage hébergeant le test. Dans ce projet, nous partirons des principes suivants :

- Un numéro de test est constitué de seize chiffres décimaux ;
- Les quatre premiers correspondent à un code spécifique à chaque centre d'archivage ;
- Les serveurs d'acquisition des centres d'archivage sont directement reliés au réseau des centres d'archivage.

Chaque serveur d'acquisition analyse donc le numéro de test qui figure dans la demande de validation qu'il reçoit, puis :

- Si le test est hébergé dans le centre d'archivage local, il envoie la demande directement au serveur de validation de ce centre d'archivage ;
- Si le test est hébergé dans un centre d'archivage étranger, le serveur d'acquisition envoie la demande sur le réseau des centres d'archivage, sans se préoccuper de la suite du transit.

Le réseau des centres d'archivage doit donc effectuer le routage des demandes de validation, c'est-à-dire analyser les demandes qui lui sont fournies, envoyer chaque demande vers le serveur d'acquisition du centre d'archivage correspondant et, enfin, prendre en charge la transmission de la réponse lorsqu'elle lui revient.

2. Cahier des charges

L'objectif de ce projet est de simuler les mécanismes décrits ci-dessus, c'est- à-dire :

- Le terminal envoyant une demande de validation au serveur d'acquisition de du centre d'archivage auquel il est connecté ;
- Le serveur d'acquisition effectuant le routage de la demande vers le bon serveur de validation, et effectuant le routage des réponses qu'il reçoit en retour des terminaux ;
- Le réseau des centres d'archivage auquel sont connectés les différents serveurs d'acquisition, capable d'effectuer le routage des demandes et des réponses relayées par les serveurs d'acquisition ;
- Le serveur de validation fournissant la réponse à la demande de validation ;
- Le tout en permettant un maximum de parallélisme.

Le cahier des charges fonctionnelles précise l'architecture générale, les fonctionnalités devant être programmées ainsi que les contraintes fonctionnelles à respecter. Le cahier des charges techniques fournit les restrictions concernant la mise en œuvre.

2.1. Cahier des charges fonctionnelles

2.1.1. Architecture fonctionnelle

La figure 2 précise le schéma qui a été présenté plus haut et retranscrit la description que nous avons fournie ci-dessus.

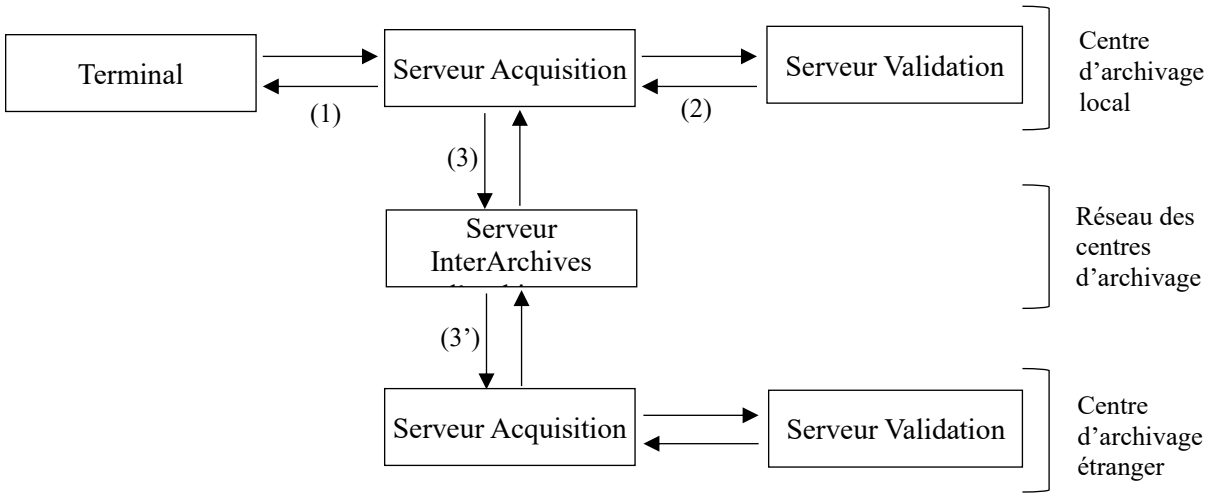

Figure 2. Architecture fonctionnelle du projet

Chaque terminal est relié via Internet (1) au serveur d'acquisition du centre d'archivage local. Celui-ci est connecté au sein du centre d'archivage (2) au serveur de validation de ce même centre d'archivage. Le réseau des centres d'archivage comprend un serveur InterArchives qui est relié (3,3') aux serveurs d'acquisition des différents centres d'archivage. Tous les autres centres d'archivages sont également reliés au serveur InterArchives, mais ne sont pas représentés sur ce schéma.

2.1.2. Terminal

Dans le cadre de ce projet, il n'est pas question d'utiliser de vrais terminaux ou de vrais tests PCR. Un terminal étant un moyen d'envoyer aux programmes des demandes de validation, il sera simulé pour ce projet par un exécutable qui enverra un numéro de test aléatoirement tiré dans la liste des tests existants.

Pour fonctionner, un terminal a donc besoin de connaître la liste des numéros des tests PCR déjà réalisés.

Chaque terminal envoie le numéro de test généré vers son serveur d'acquisition, attend la réponse du serveur d'acquisition et affiche cette réponse à l'écran (test valide ou non).

Les échanges entre les terminaux et leur serveur se réalisent suivant un protocole bien déterminé : les informations sont formatées d'une certaine façon. Afin de simplifier le projet et de garantir l'interopérabilité des différents projets entre eux (voir en annexe pour une explication de ce point), un seul protocole de communication est utilisé. Celui-ci est décrit en annexe de ce document.

2.1.3. Serveur d'acquisition

Un serveur d'acquisition n'a qu'une fonction du routage.

- Il doit pouvoir accepter des demandes de validation provenant de terminaux et du réseau des centres d'archivage.
- Il doit pouvoir effectuer le routage des demandes de validation vers le serveur de validation de son centre d'archivage ou bien vers le réseau des centres d'archivage.
- Il doit pouvoir accepter les réponses provenant du serveur *InterArchives* ou du serveur de validation de son centre d'archivage.
- Il doit pouvoir envoyer les réponses vers le réseau des centres d'archivage ou vers le terminal (en étant capable d'apparier chaque réponse à la demande initiale).

Un serveur d'acquisition doit donc être capable d'utiliser le protocole de communication employé par les terminaux, ainsi que le protocole du réseau des centres d'archivage (voir les protocoles définis en annexe).

Afin d'effectuer correctement le routage des messages, le serveur d'acquisition analyse, pour chaque message, le numéro (codé sur 16 chiffres) du test PCR concerné par le message : les 4 premiers chiffres indiquent le centre qui héberge le test PCR concerné.

Afin d'effectuer correctement le routage des réponses, un serveur d'acquisition doit garder trace des numéros des tests lus par les terminaux de validation et attendant une validation.

2.1.4. Serveur de validation

Le serveur de validation doit être capable de fournir une réponse à une demande de validation. Pour fonctionner, le serveur de validation doit donc avoir accès aux résultats stockés dans le centre d'archivage.

Dans le cadre de ce projet, nous utilisons une méthode simple : le serveur de validation possède la liste des numéros de tests stockés dans le centre d'archivage correspondant, auxquels sont associés les résultats et les dates de tests ; lorsqu'une demande de validation lui parvient, le

serveur vérifie que le numéro de test figure bien dans sa liste. Il contrôle alors que le test est récent et que son résultat est négatif. Si c'est le cas, il répond oui, sinon il répond non.

2.1.5. Le serveur *InterArchives*

Le serveur *InterArchives* n'a qu'une fonction du routage :

- Il doit pouvoir accepter les messages (demande de validation et réponse à une demande de validation) provenant des serveurs d'acquisition.
- Il doit pouvoir analyser le contenu des messages pour déterminer vers quel serveur d'acquisition il doit les envoyer.

Pour fonctionner, le serveur *InterArchives* doit posséder la liste des codes de 4 premiers chiffres des tests PCR et les centres d'archivage associés.

2.2. Cahier des charges techniques

2.2.1. Contraintes générales

- L'ensemble de la simulation tournera sur une seule machine (dans un premier temps, avant l'utilisation des `sockets`).
- Les programmes seront écrits en langage C, et mettront en œuvre les concepts et les techniques apprises pendant le cours de Systèmes d'exploitation.
- Un maximum de parallélismes est exigé dans ce projet.

2.2.2. Mémoire des serveurs

Pour gérer les messages en attente, les serveurs d'acquisition et le serveur *InterArchives* ont une mémoire finie. La taille de cette mémoire (le nombre de case du tableau) sera précisée sur la ligne de commande au lancement d'un serveur.

2.2.3. Nombre de processus

Chaque composant « fonctionnel » tel qu'il a été présenté dans le cahier des charges fonctionnelles correspond à un processus. On trouve donc un processus pour le terminal (que l'on nomme *Terminal*), un processus pour un serveur d'acquisition (*Acquisition*), un processus pour un serveur de validation (*Validation*) et un processus pour le serveur *InterArchives* (*InterArchives*).

La simulation pouvant mettre en jeu plusieurs terminaux et plusieurs centres d'archivage, on trouve en fait un processus *Terminal* par terminal, un processus *Acquisition* par centre d'archivage et un processus *Validation* par centre d'archivage.

Pour favoriser le parallélisme, chaque processus peut (si besoin est) être découpé en plusieurs threads.

2.2.4. Paramètres

Les paramètres nécessaires au fonctionnement des différents processus sont fournis via « la ligne de commande », à l'exception des paramètres suivants qui sont fournis sous forme de fichier :

- La liste des centres d'archivage et leurs codes à 4 chiffres.
- La liste des tests PCR stockés dans chaque centre d'archivage, avec leurs résultats et dates du passage.

Ces fichiers sont au format texte, chaque ligne contenant les informations demandées (code sur 4 chiffres et centre d'archivage associée ou numéro de test avec le résultat et la date), séparées par un espace.

2.2.5. Gestion des échanges par tuyaux

Chaque terminal est connecté à son serveur d'acquisition par une paire de tuyaux. Le serveur a donc comme rôle d'orchestrer simultanément les lectures et les écritures sur ces tuyaux. Ceci implique, pour chacun des processus *Acquisition*, de maintenir une table de routage entre numéro de test et terminaux en attente d'une réponse.

Les échanges entre un serveur d'acquisition et un serveur de validation sont possibles au travers une paire de tuyaux.

En ce qui concerne la connexion entre le serveur *InterArchives* et les serveurs d'acquisition, la problématique est strictement identique à celle de connexion entre terminaux et serveurs d'acquisition. Il faut utiliser une paire de tuyaux pour connecter chaque serveur d'acquisition au serveur *InterArchives*. Ceci implique donc également que le processus *InterArchives* maintient une table de routage entre serveurs d'acquisitions et numéros de tests.

2.3. Comment tester sans s'y perdre ?

Le schéma proposé ci-dessus comporte un petit défaut tant que les communications entre processus se feront sur la même machine (donc sans utiliser de `sockets`, développement proposé en option) : chaque *Terminal* va fonctionner à partir de la même fenêtre (le même terminal, le même *shell*). Tous les affichages vont donc se faire sur cette même fenêtre.

Afin de faire bénéficier chaque processus *Terminal* d'une entrée et d'une sortie standard qui lui soient propres, une astuce peut être utilisée : lors de la création d'un nouveau *Terminal*, recouvrir le processus courant non pas par *Terminal*, mais par `xterm -e Terminal`

Cette astuce n'est pas très esthétique, et n'est qu'un intermédiaire avant la communication par `sockets`.

2.4. Livrables

Doivent être livrés sous format électronique

1. Un ensemble de fichiers `C`, amplement commentés, correspondant aux différents programmes constituant la simulation.
2. Un fichier `Makefile` permettant de compiler tous les programmes (y compris les programmes de test).
3. Un mode d'emploi expliquant comment obtenir une application opérationnelle, comment l'exécuter, et comment la tester.
4. Un manuel technique détaillant l'architecture, le rôle de chaque composant, expliquant comment tester le bon fonctionnement de façon indépendante et justifiant les choix techniques effectués. Ce manuel doit en particulier bien préciser comment le projet répond au cahier des charges (ce qu'il sait faire, ce qu'il ne sait pas faire, et mettre en exergue ses qualités et ses défauts.
5. Dans le manuel technique, on démontrera qu'il ne peut pas y avoir d'interblocage entre les différents processus, pas plus qu'entre les threads d'un même processus.

Tous les documents à produire s'adressent à des ingénieurs généralistes et les rédacteurs doivent donc veiller à donner des explications concises et claires.

Aucun fichier exécutable, fichier objet, fichier de sauvegarde, fichier superflu ou inutile ne devra être transmis avec les livrables.

Il pourra éventuellement vous être demandé de remettre une version papier des rapports.

3. Conduite du projet

Les étapes qui vous sont suggérés dans cette partie permettent une approche « sereine » de la programmation de ce projet.

3.1. Introduction

3.1.1. Un projet en plusieurs étapes

Le développement du simulateur présenté ici doit être réalisé en plusieurs étapes. Ce découpage conduit le développeur à écrire des programmes indépendants les uns des autres, qui communiqueront entre eux par l'intermédiaire de tuyaux, souvent après redirection des entrées/sorties standards.

3.1.2. Méthode « diviser pour régner »

La constitution de l'application globale par écriture de « petits » programmes indépendants qui communiqueront ensuite entre eux est un gage de réussite : chaque « petit » programme générique peut être écrit, testé et validé indépendamment, et la réalisation du projet devient alors simple et progressive.

La meilleure stratégie à adopter consiste à écrire une version minimale de chaque brique du projet, à faire communiquer ces briques entre elles, puis éventuellement, à enrichir au fur et à mesure chacune des briques. La stratégie consistant à développer à outrance une des briques pour ensuite s'attaquer tardivement au reste du projet conduit généralement au pire des rapport résultat/travail.

Avant toute programmation, conception ou réalisation de ce projet, il est fortement conseillé de lire l'intégralité de l'énoncé, y compris les parties que vous pensez ne pas réaliser, et de s'astreindre à comprendre la structuration en étapes proposée ici.

La traduction de cet énoncé sous forme de schéma est indispensable.

En particulier, il est très important de définir dès le départ l'ensemble des flux de données qui vont transiter d'un processus à un autre, de déterminer comment il est possible de s'assurer que ces flux sont bien envoyés et reçus, puis de programmer les fonctions d'émission et de réception correspondantes. Ces fonctions sont ensuite utilisées dans tous les processus du projet.

Un schéma clair et complet associé à des communications correctement gérées garantissent la réussite de ce projet et simplifient grandement son développement et son débogage.

3.2. Etape 1 : les fonctions de communication

La première étape a été faite pour vous. Elle consiste à formater (selon le protocole définis en annexe) un message complet à partir de ses différents champs et réciproquement à extraire les différents champs à partir d'un message déjà formaté. Les messages ainsi formatés peuvent être lus et écrits dans des fichiers grâce aux librairies programmées en TP, dont une version vous est aussi fournie.

Les problèmes de synchronisation seront abordés dans les étapes suivantes et il s'agit pour l'instant uniquement de traduire sous forme de programmes (de fonctions en C) les protocoles de communication : formatage de messages selon la structure définie en annexe et récupération des informations stockées sous cette forme.

Une fois que vous aurez lu le code, et compris son utilisation, vous serez à même de les utiliser ; ces fonctions sont utilisées par tous les processus du projet.

3.3. Etape 2 : réalisation de programmes indépendants

Dans cette deuxième étape, il s'agit de mettre en place les différentes briques du projet et de pouvoir les tester indépendamment.

3.3.1. Processus : *Terminal*

Le processus *Terminal* offre l'interface permettant d'envoyer et de recevoir les informations pour une validation de test (numéro de test et durée de validité). Nous souhaitons utiliser exactement le même code pour tester *Terminal* de manière indépendante et pour l'utiliser dans la simulation globale du système. Pour cela, *Terminal* accepte comme arguments deux descripteurs de fichier vers lesquels il redirige son entrée et sa sortie standard (voir annexe). Ensuite, il lit son entrée standard et écrit sur sa sortie standard. On peut ainsi utiliser *Terminal* :

- En passant 0 et 1 comme arguments, ce qui permet de tester *Terminal* « à la main » en lisant les demandes de paiement à l'écran sur la sortie standard et en lui passant les réponses (validation ou non) via l'entrée standard ; ou
- En passant les descripteurs de tuyaux utilisés par *Acquisition*, ce qui permet d'utiliser *Terminal* dans la simulation complète du système.

3.3.2. Processus : *Validation*

Le processus *Validation* offre l'interface permettant de recevoir des demandes de validation et d'envoyer les réponses à ces demandes. Comme *Terminal*, il accepte comme arguments deux descripteurs de fichier vers lesquels il redirige son entrée et sa sortie standard.

3.3.3. Processus : *Acquisition*

Le processus *Acquisition* reçoit des demandes de validation en provenance soit des terminaux, soit du serveur *InterArchives* et des réponses en provenance soit du serveur *InterArchives* soit du serveur de validation. Les ordres seront redirigés (« routés ») vers le serveur *InterArchives* ou bien vers le serveur de validation et les réponses seront redirigés soit vers les terminaux soit vers le serveur *InterArchives*, conformément au cahier des charges. A cette phase du projet, on peut utiliser :

- Des fichiers dans lesquels le processus *Acquisition* écrit lorsqu'il est supposé envoyer un message ;
- Des fichiers dans lesquels le processus *Acquisition* lit les messages lorsqu'il s'attend à recevoir ; ces derniers seront préparés « à la main ».

Ces fichiers permettent de simuler les échanges avec les processus *Validation*, *Terminal* et *InterArchives*.

Le programme doit accepter sur sa ligne de commande au moins un paramètre représentant la taille de la mémoire servant à la gestion des demandes de validation.

Dans une première phase, le processus *Acquisition* traite *séquentiellement* chaque terminal, puis le serveur de validation et enfin le réseau des centres d'archivage. Dans une seconde phase, on *parallélise* au maximum les requêtes. La première phase doit être pensée et conçue sur le papier afin de pouvoir facilement passer à la seconde phase. La première phase peut suffire pour passer à la suite du projet, mais elle est obligatoire et devra être rendue. On conservera toutes les versions des différentes phases du programme à des fins de test.

3.4. Etape 3 : création d'un réseau centre d'archivage

3.4.1. Préparation de la communication par tuyaux

En pratique, le processus *Acquisition*, le processus *Validation* et chaque processus *Terminal* communiquent par une paire unique de tuyaux. Une fois ces deux tuyaux créés, il est possible de modifier le programme *Acquisition* pour qu'il utilise non pas des fichiers mais ces deux tuyaux. Les processus *Terminal* et *Validation* n'ont pas à être modifiés.

3.4.2. Raccordement

Pour terminer la mise en place des communications au sein d'un même centre d'archivage, il faut maintenant créer d'une part une paire de tuyaux entre *Acquisition* et chaque *Terminal* (il y aura autant de paires de tuyaux que de terminaux) et, d'autre part, une paire de tuyaux entre *Acquisition* et *Validation*.

Modifier le programme *Acquisition* pour qu'il accepte sur sa ligne de commande les paramètres suivants :

- Le nom du centre d'archivage à simuler ;
- Le code de 4 chiffres associé à ce centre d'archivage ;
- Le nom du fichier contenant les résultats des tests PCR ;
- Le nombre de terminaux à créer.

Créer les tuyaux nécessaires, opérer les clonages et recouvrements nécessaires pour créer les processus *Terminal* et *Validation* en nombre suffisant.

3.5. Etape 4 : création du réseau des centres d'archivage

3.5.1. Processus : *InterArchives*

Chaque serveur d'acquisition est relié au processus *InterArchives* par une paire de tuyaux. Celle-ci permet à *InterArchives* de recevoir les messages de demande de validation et de transmettre les réponses en retour, après les avoir routés.

L'architecture à mettre en place entre *InterArchives* et les différents processus *Acquisition* est similaire à celle mise en place entre chaque processus *Acquisition* et les processus *Terminal* qui y sont reliés.

Dans un premier temps, les messages transitant par *InterArchives* sont simplement lus et écrits dans des fichiers, sans qu'aucune communication ne soit mise en place.

Dans une première phase, le processus *InterArchives* traite *séquentiellement* chaque serveur d'acquisition. Dans une deuxième phase, on *parallélise* au maximum les requêtes. La première phase doit être pensée et conçue sur le papier afin de pouvoir facilement passer à la deuxième phase. La première phase peut suffire pour passer à la suite du projet, mais elle est obligatoire et devra être rendue. On conservera toutes les versions des différentes phases du programme à des fins de test.

3.5.2. Raccordement

Pour terminer la mise en place des communications au sein du réseau des centres d'archivage, il faut maintenant créer une paire de tuyaux entre *InterArchives* et chaque serveur *Acquisition*.

Modifier le programme *InterArchives* pour qu'il accepte sur sa ligne de commande un fichier de configuration dans lequel on trouve les informations suivantes :

- Le nom d'un fichier contenant toutes les centre d'archivages et les 4 chiffres associés à chaque centre d'archivage ;
- Les noms des fichiers nécessaires au fonctionnement de chaque centre d'archivage ;
- Le nombre de terminaux pour chaque centre d'archivage.

Modifier *InterArchives* pour qu'il crée les tuyaux de communication avec les processus *Acquisition* et opère les clonages et recouvrements pour créer les processus *Acquisition* en nombre suffisant.

4. Evolutions complémentaires et optionnelles

Utilisation de `socket`

L'utilisation de communications entre machines ne fait pas partie des objectifs de ce cours. Cependant, afin de rendre le projet plus vivant et plus attractif, nous proposons aux plus débrouillards d'utiliser des `sockets` de sorte que la simulation soit plus réaliste :

- Chaque terminal communique avec son serveur d'acquisition par `socket` ;
- Chaque serveur d'acquisition communique avec le réseau interbancaire par `socket`.

Les informations nécessaires à l'utilisation des `sockets` peuvent facilement se trouver sur internet.

Le travail de mise en œuvre des communications par socket doit être entrepris **uniquement après avoir réussi à programmer une application complètement opérationnelle en mode texte**.

Attention : **ne jamais modifier un programme qui fonctionne et toujours travailler sur une copie de celui-ci**.

5. Annexes

5.1. Git et GitHub

Git est un logiciel de contrôle de version pour les développeurs. Le contrôle de version est le processus d'enregistrement de différents fichiers ou « versions » tout au long des différentes étapes d'un projet. Cela permet aux développeurs de garder une trace de toutes les modifications, et de revenir à une version précédente en cas de besoin. Git permet de travailler sur un projet (dépôt) local, et de *pousser* (push) et *tirer* (pull) des modifications vers et depuis des dépôts distants, comme des dépôts GitHub.

GitHub facilite la collaboration en utilisant git. C'est une plateforme qui peut héberger des dépôts de code dans un stockage dans le cloud afin de faciliter la collaboration de plusieurs développeurs sur un même projet et voir les modifications des autres en temps réel. Les dépôts GitHub publics sont accessibles par tout le monde, tandis que l'accès aux dépôts privés est restreint.

Vous trouverez ci-dessous la liste des commandes Git qui vous seront utiles pour gérer votre projet :

- `git init`
- `git add`
- `git status`
- `git commit -m "message"`
- `git push`

- `git pull`
- `git branch`
- `git tag`
- `git config`
- `git checkout`
- `git remote`
- `git help` (exemple: `git help push` pour avoir de l'aide sur la commande `git push`)

Nous vous invitons à consulter un livre Git sur le page: https://git-scm.com/book/fr/v2

Remarque importante.

- Créer un dépôt local pour votre projet (sur votre machine) en utilisant la commande `git init`.
- Utiliser un fichier «`.gitignore`» pour faire en sorte que Git ignore (c'est-à-dire, qu'il exclut du suivi) certains fichiers ou répertoires, comme les fichiers «`.o`» et les fichiers exécutables.
- Créer un compte GitHub (https://github.com/), et créer un dépôt (*repository*) distant.
- Utiliser la commande `git push` pour envoyer le contenu de votre dépôt local vers le dépôt distant.
- Faire `git pull` avant chaque session de travail.
- Faire `git commit -m "message"`, et `git push` après chaque session de travail.
- Faire `git tag -a` *`nom_étiquète`* `-m 'message'` pour garder une trace de chaque version fonctionnelle.

Attention !

Votre dépôt distant sur GitHub correspondant à votre projet doit être privé !

5.2. Codes disponibles

Après la création de votre dépôt Git, vous devrez ajouter dans ce dépôt certaines fonctions C (décrites Sections 5.4, 5.5 et 5.6) qui vous seront transmises par email et qui pourront vous aider dans la réalisation du projet.

Vous aurez également à votre disposition un générateur aléatoire implémenté dans les fichiers `alea.c` et `alea.h`.

Vous trouverez enfin une bibliothèque permettant de lire et écrire une ligne dans un fichier (comme spécifié dans les TP de l'unité 'Système d'exploitation') dans les fichiers `LectureEcriture.{c,h}`.

5.3. De l'intérêt des standards pour assurer une meilleure interopérabilité

Le recours à des protocoles « standardisés » (ou pour le moins communs) à un double intérêt :

- Il permet de gagner du temps sur le développement de ce projet et d'illustrer par la pratique la façon dont les problèmes sont traditionnellement résolus en informatique :
- Il permet de mélanger les projets entre eux afin de tester le respect du protocole et peut-être, d'aider certains binômes à avancer (il suffit d'utiliser les processus d'un autre binôme[5]).

La capacité ainsi esquissée à mélanger des composants issus de programmeurs ou prestataires différents pour constituer un système d'information unique se nomme « interopérabilité ». C'est un des enjeux majeurs de l'informatique actuelle.

5.4. Protocoles de communication et format de messages

Les messages sont constitués de caractères ASCII (sans accent). Chaque message est constitué de différents champs séparés par les caractères `'|'` et terminés par un caractère de fin de message `'\n'`.

Remarque : afin de simplifier au maximum le protocole, on considère qu'il est impossible que plus d'une demande validation ou réponse, correspondante à un test donné, ne circule sur le réseau. Cette hypothèse, assez réaliste, permet de simplifier l'appariement des ordres et des accusés de réception au niveau des processus *Acquisition* et *InterArchives*.

Les échanges utilisent deux types de message, *la demande validation*, et *la réponse*. Ces deux types de messages partagent un format générique comportant trois champs :

Format : `|C...C|T...T|V...V|\n`

- `C...C` est le numéro du test PCR, codé sur 16 chiffres, sur lequel porte le message ;
- `T...T` est le type du message, c'est à dire 'Demande' ou 'Réponse' ;
- `V...V` est la valeur associée au message. Dans le cas d'une demande, il s'agit de la durée de validité des tests. Dans le cas d'une réponse, il s'agit d'un booléen qui vaut '1' si la demande est acceptée ou'0' si elle est refusée.

Les fichiers `message.c` et `message.h` (fournis dans l'archive disponible en ligne) implémentent les fonctions pour créer et interpréter un message formaté selon ce protocole.

[5] Bien entendu, cette utilisation ne peut s'entendre qu'à des fins de mise au point. . .

5.5. Générateur aléatoire

L'archive disponible en ligne contient également un exemple de générateur aléatoire.

5.6. Redirection des entrées et des sorties

La redirection des entrées et des sorties peut se faire grâce à l'appel système `dup`.

```
int dup2 (int oldfiledes, int newfiledes);
```

Cet appel système sert à dupliquer le contenu du descripteur `oldfiledes` dans un autre descripteur `newfiledes`. Il renvoie-1 en cas d'échec, et il prend en argument deux descripteurs de fichiers. Par exemple si on a envie que `STDIN (==1)` soit en fait `STDERR (==2)` on peut écrire ceci :

```
dup2(2,1);
```

L'exemple suivant redirige l'entrée et la sortie standard vers un tube, et recouvre le processus courant et son fils par deux programmes. Le père et le fils vont communiquer par leur entrée et sortie standard.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

void usage(char * basename) {
    fprintf(stderr,
        "usage : %s [<programme 1> [<programme 2>]]\n",
        basename);
    exit(1);
}

int main(int argc, char *argv[]) {
    int pid;       /* permet d'identifier qui on est*/
    int fdpipe[2]; /* sera utilisé pour lier les processus */

    if (argc != 3) usage(argv[0]);

    /* on créé le pipe  qui sera utilisé pour relier
       la sortie du premier processus
       vers l'entrée du second
    */
    if ( pipe(fdpipe) == -1 ) {
        perror("pipe");
        exit(-1);
    }

    switch(pid = fork()) {
        case -1:
            /* le fork a échoué */
            perror("fork");
```

```
            exit(-1);
        case 0:
            /* code du fils */
            /* on fait en sorte que lorsque le processus
               écrira sur l'entrée standard (1)
               il le fera en fait dans le pipe (fdpipe[1])
            */
            dup2(fdpipe[1], 1);
            /* on ferme tout, même le pipe...
               on n'en a plus besoin
            */
            close(fdpipe[0]);
            close(fdpipe[1]);
            execlp(argv[1], argv[1],NULL);
            /* pas besoin de break,
               ce code n'existe déjà plus à l'exécution
            */
        default :
            /* code du père */
            /* on fait en sorte que lorsque le processus
               lira sur la sortie standard (0)
               il le fera en fait dans le pipe (fdpipe[0])
            */
            dup2(fdpipe[0], 0);
            close(fdpipe[0]);
            close(fdpipe[1]);
            execlp(argv[2],argv[2], NULL);
    }
   /*
   cette portion de code ne sera jamais exécutée,
   puisque les processus ont déjà
   été remplacé. On met néanmoins un
   return sinon le compilateur proteste.
   */
   return 0;
}
```

Le code ci-dessus permettant de rediriger les entrées et sorties est disponible dans le fichier `TestRedirection.c` que vous trouverez dans l'archive téléchargées.