Système d’exploitation TP :

Projet de C : Gestion des fichiers de type fifo.

# Sommaire

[1. Introduction 2](#_Toc344412308)

[2. Consignes du travail 3](#_Toc344412309)

[a) Consignes 3](#_Toc344412310)

[b) Exemples 3](#_Toc344412311)

[3. Plan du travail 4](#_Toc344412312)

[4. Structure des données 5](#_Toc344412313)

[5. Explications détaillées 6](#_Toc344412314)

[a) Le serveur 6](#_Toc344412315)

[b) Le Client ou pilote 7](#_Toc344412316)

[c) Le serveur météo 8](#_Toc344412317)

[6. Conclusion et avis personnels 9](#_Toc344412318)

[7. Avis personnels 9](#_Toc344412319)

# Introduction

En première, nous avions pris du plaisir à réaliser le projet de C, à le rendre parfait, sans aucune erreur possible,… Ce fut une expérience agréable et donc il ressortait qu’on était motivé à la réalisation d’un autre projet.

Ensuite, il y a eu le projet de Java ! Car oui, pendant quelques temps, le C était juste un très joli rapport sur une étagère de la bibliothèque. Nous nous sommes battus avec les constructeurs, les « implements » et autres « extends ». Néanmoins, notre démineur est fini et fonctionnel et nous pouvons alors nous attaquer au C. En période de blocus certes mains étant donné que seul le projet compte pour l’examen, c’est le moment n’est-ce pas ?

Reste à savoir si on sait encore ce que c’est que le C. C’est peut-être bien là une difficulté de plus, et non prévue celle-là. Comment changer en quelques jours d’une logique de programmation orientée objet à un langage système procédural ? Nous allons voir dans ce projet si cela est possible est facile.

Pour être franc, il faut noter que ce projet faisait un peu peur compte tenu du contenu du cours de système d’exploitation (théorie) avec ses algorithmes et ses appels système alors que en fait la programmation C reste de la programmation C mais à la sauce Linux. C’est-à-dire savoir utiliser des fonctions propres au système Unix et donc sans aucune portabilité possible. Mais ce n’est pas le but du travail…

# Consignes du travail

Ceci est issu du document Word disponible sur ecampus

## Consignes

Des pilotes qui souhaitent décoller d’un aéroport non contrôlé ont besoin, pour ce faire, de connaître les informations ATIS. Celles-ci sont accessibles via un serveur.

Chaque pilote va envoyer au serveur une demande ATIS. Le serveur va lors répondre à cette demande en allant chercher les informations nécessaires dans un fichier ATIS.

Le pilote va recevoir ces informations et il doit alors obligatoirement répondre au serveur en lui envoyant soit :

* Un ACK OK qui signifie « informations bien reçues » et provoque la fin de la communication
* Un ACK KO qui signifie « informations mal comprises » et nécessite de renvoyer les informations.

Le serveur doit pouvoir gérer un nombre indéfini de pilotes (restons réalistes).

Le fichier ATIS contenant les informations nécessaires aux pilotes doit être régulièrement mis à jour par le gestionnaire météo.

## Exemples

**EBLG 1803 00000KT 0600 FG OVC008 BKN040 PROB40 2024 0300 DZ FG OVC002 BKN040**

TAF de Liège, valable de 18h à 3h UTC, vent calme, visibilité de 600m, brouillard, ciel couvert à 800ft, nuages fragmentés à 4000ft, probabilité 40% d’avoir, entre 20 et 24h UTC : visibilité 300m, bruine, brouillard, ciel couvert à 200ft, nuages fragmentés à 4000ft.

**EBBR 0615 20015KT 8000 RA SCT010 OVC015 TEMPO 0608 5000 RA BKN005 BECMG 0810 9999 NSW BKN025**

TAF de Bruxelles, validité de 6 à15h UTC, vent 200°/15kt, visibilité 8km, pluie, nuages épars à 1000ft, couvert à 1500ft. Temporairement entre 6 et 8h UTC : visibilité 5km, pluie, nuages morcelés à 500ft, devenant entre 8 et 10h UTC : visibilité + de 10km, disparition de la pluie et nuages morcelés à 2500ft.

# Plan du travail

Notre projet est divisé en trois sous-programmes :

* Les programmes de type serveur qui doivent être lancés avant de commencer à travailler proprement dit. Il s’agit du serveur ATIS ainsi que du programme météo.
* Le client qui peut lancer plusieurs instances en même temps qui communiqueront (uniquement) avec le serveur ATIS.

Voici un petit schéma simple du fonctionnement du programme en termes de déroulement du point de vue du client. Les explications détaillées se trouvent après dans ce rapport.



Pour parler un peu français et donc sans entrer dans les détails. Les pilotes demandent les informations ATIS au serveur. Une communication s’établi entre le serveur et le client. Si le client reçois correctement l’information, il envoi au serveur un OK et le serveur traite cet « OK » en clôturant le transfert. Si le pilote n’a pas bien reçu le message ATIS, il redemande ce dernier jusqu’à ce qu’il accuse réception du message.

Le projet fonctionne avec un fichier de type fifo en entrée que le serveur lit et plusieurs fifo en sortie (un par client) dans lesquels le serveur écrit.

# Structure des données

La structure est assez simple mais malgré tout intégrée dans une bibliothèque présente à la racine du dossier du projet. Elle contient deux types qui sont les demandes et les réponses ainsi que les deux constantes qui sont le nom du fifo d’entrée et le nom du fichier texte météo.

Extrait du code correspondant pour ne pas être trop long :

*#define fifoIN "fifoIN"*

*#define meteo "meteo.txt"*

*// Une demande au serveur composé du PID et d'un type*

*// Le type peut être ASK, OK ou KO*

*typedef struct{*

*int pid;*

*char type[3];*

*} Tdemande;*

*// Une réponse avec le PID (pour être certain) et les infos météo sur 100 caractères*

*typedef struct{*

*int pid;*

*char reponse[100];*

*} Treponse;*

Par soucis de structure du rapport. Ce code sera encore placé dans la section 2 qui contiendra l’ensemble du code du projet. Le papier et les arbres nous pardonnent si ce gaspillage est limité.

# Explications détaillées

## Le serveur

Le serveur doit être le premier programme lancé (si on oublie le serveur météo) avant bien entendu de commencer à lancer les pilotes. Et bon on peut s’amuser un peu car l’expression « lancer des pilotes » convient parfaitement étant donné que les demandes ATIS permettent aux avions de décoller ☺. Une fois le serveur lancé, il crée le fifo ou le recrée et l’ouvre en lecteur seule et attend que le premier client ouvre le fifo en écriture pour permettre de l’utiliser.

Lorsque le premier pilote fait son apparition, il ouvre lui aussi le fifo in mais en écriture cette fois. Il y place une demande qui contient un type « ASK » pour demande et son pid. Le serveur lit cette demande dans le fifo d’entrée et traite la demande en allant chercher les informations météo dans le fichier texte. Ce petit fichier ne contient qu’un message ATIS correspondant à la météo du moment.

*Petite parenthèse d’ailleurs* concernant ce message ATIS, il faut bien entendu être réalise. Normalement ce message change en fonction de l’avion, de la météo et de la piste utilisée (pour la direction du vent). Il est bien entendu matériellement pas possible de faire varier tous ces paramètres mais normalement c’est le cas…

Donc revenons à notre serveur et au moment où il va chercher l’information météo. Il envoi ensuite au client, via le fifo individuel du client, le message ATIS avec le pid du processus correspondant afin de vérifier que le serveur fait bien son travail.

Si le serveur reçois une demande de type « OK », il met fin au transfert en supprimant le fifo du pid envoyé dans le OK, et si il reçoit un « KO » il le notifie à l’écran et sait qu’il recevra bientôt une autre ASK du client concerné comme si il s’agissait d’une demande classique.

## Le Client ou pilote

Le seul et unique rôle de client dans notre projet et de vouloir décoller. Et pour dire vrai c’est déjà pas mal mais bon ce n’est pas si facile. Si on arrive à l’image ci-dessous, c’est vraiment très bien !



Décollage d’un A380 – Le but du pilote ;)

Alors comme on veut « simuler » un véritable système avec plusieurs pilotes et une seule tour de contrôle, il faut pouvoir lancer plusieurs pilotes en une fois. Nous avons testé une situation avec 10 pilotes qui ont communiqué avec le serveur ATIS pour obtenir les précieuses informations météo.

Le pilote envoi donc une demande au serveur avec son pid dedans et il attend sagement la réponse du serveur (5 secondes de traitement). Une fois cette réponse « reçue », le pilote doit confirmer auprès du serveur s’il a bien reçu le message. Pour générer une pseudo erreur, nous avons généré un nombre aléatoire et nous pouvons donc vous affirmer qu’un pilote ne recevra pas correctement les données dans 1 cas sur 4. Explications des deux cas :

* Si le pilote à bien reçu le message ATIS, il envoie un OK au serveur et lui ne fait plus rien, le serveur s’occupe de tout.
* S’il n’a pas bien reçu le message (voir erreur ci-dessous) il envoie un KO au serveur et attend que le serveur retraite sa demande. Quand le serveur est de retour au cas de notre pilote, cela se passe comme une demande classique.

## Le serveur météo



Exemple de station météo à l’aéroport de Berlin

Le serveur météo ne se charge que de modifier le fichier texte en ajoutant les informations actuelles. Mais ça c’est la théorie alors que dans la pratique le fichier météo efface le fichier texte et le réécrit toutes les 5 minutes en modifiant la ligne ATIS que le serveur utilisera lors des demandes des clients.

Il se base sur un tableau contenant des chaines des caractères. Ces chaines de caractères étant les demandes ATIS et une petite boucle for qui toute les 5 minutes passe d’un indice du tableau vers le suivant et change la ligne dans le fichier texte. Quand on arrive à la fin du tableau réel (et non physique), on revient au début et on recommence pour un tour.

Il n’y a ici aucun intérêt de faire ceci avec un nombre aléatoire étant donné que le but du programme météo et d’écrire et de lire dans les fichiers texte avec le serveur ATIS et le serveur météo.

# Conclusion et avis personnels

Et voilà le projet et terminé et semble fonctionnel. Certaines choses peuvent surement être améliorées d’un point de vue gestion des pilotes mais il s’agit d’une question de logique aéronautique et non de programmation.

Ce projet nous a permis de constater les possibilités du C quand il s’agit de programmation système sous Linux et il va de soi qu’il faut continuer pour vraiment être attirer par la programmation système et peut être pour réussir le cours théorique.

Et non… Ce n’est pas compliquer de retrouver ses bases en C. En revanche, l’envie commence à manquer et un vrai environnement de développement en C serait le bienvenu ;)

# Avis personnels

Vive le C, même le C sous Linux. C’est un langage simple, naturel, puissant et très vieux. Certains me diront que tout est fait en java et je leurs dirai que tout est basé du C. Et c’est facile… Je n’aime pas le java, c’est pas très compliqué à voir. Et j’aimerai bien pousser encore plus loin la programmation système.

*Antoine BETAS*