

**Rapport du devoir de système d’exploitation**

Alexandre KARAKAS - Marc CHOU

Ingénieur Informatique 1



Année 2017 - 2018

**Table des matières**

[1. Préparation du projet 1](#_Toc897729441)

# 1. Préparation du projet

Dans un premier temps, il a fallu préparer l’environnement de travail. Nous avons choisi d’utiliser l’IDE (*Integrated Development Environment*) *CLion* pour ses multiples fonctions pratiques. Ensuite, nous avons construit un dépôt de version, sur la plateforme *GitHub*, pour pouvoir partager notre code plus facilement et retourner vers d’anciennes versions de code en cas de problème. Après cette préparation, nous avons relevé les informations importantes contenues dans le sujet.

Le projet demande l’utilisation de file FIFO. Nous avons décidé de reprendre le code d’une file, dans notre cours de *Structures de données*. Il permet de manipuler plus facilement le contenu et répondre aux cahiers des charges. De plus, ce code permet de ne pas “sur-allouer” de l’espace mémoire mais, d’allouer l’espace nécessaire pour chaque file et chaque structure *passenger*.

L’implémentation du tube de communication a été un peu compliquée. Nous nous sommes posés la question de savoir si nous pouvions passer une structure dans un pipe et si nous pouvions la récupérer. Pour répondre à ce problème, nous avons fait des recherches sur internet et tester les solutions proposées. Durant ces tests, un problème récurent est survenu :

|  |
| --- |
| \*\*\* stack smashing detected \*\*\*: <unknown> terminated |

Après l’utilisation de la fonction *printf* sur certaines parties du code, nous avons pu déterminer quelle fonction posait problème. La fonction *read* était la source.

|  |
| --- |
| read(fd, &passenger2, **sizeof**(passenger)); |

Cette fonction lit jusqu’à la taille d’une structure *passenger* d’octet depuis le descripteur de fichier *fd* dans le tampon pointé par *passenger2*. Or, lorsque nous avons écrit la fonction *write*, nous avons passé l’adresse d’un *passenger*. Donc, la taille d’octet à lire est incorrecte. La solution est la suivante :

|  |
| --- |
| read(fd, &passenger2, **sizeof**(int32\_t)); |

Concernant les threads nous avons utiliser ce que nous avions vu en cours c’est-à-dire la bibliothéque pthread. Cette bibliothéque nous permet d’utiliser la fonction pthread\_create.

Après avoir creer le thread métro, le thread bus et le thread vérificateur nous sommes posés la question de savoir si il fallait créer la file du métro et du bus dans le thread ou le considérer comme une variable globale.

Dans notre projet nous les avons considérés comme des variables globales.

Pour la suite nous avons entrer en paramètre de la fonction un tableau de file qui correspond au file de chaque station noté de 0 à 7.

Puis on parcours la file de la station correspondante et on récupère les passagers.

Enfin on teste pour chaque station si le passager est arrivé à destination.