

# Projet Système d'exploitation

Pierre Barthélemy, Mathieu Blondel

16 décembre 2018

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>File synchronisée</b>	<b>4</b>
<b>3</b>	<b>Serveur</b>	<b>5</b>
<b>4</b>	<b>Client</b>	<b>6</b>

# 1 Introduction

Réalisation d'un lanceur de commande en ligne de commande, le projet permettra de lancer des commandes à partir de plusieurs client en définissant des pipes comme entrée et sortie standard à ses commande. Les commandes sont exécuter par un unique serveur. Le programme est développer en 3 partie, une file synchronisée, un serveur et un client.

## file synchronisée

La liste d'attente synchronisée est réaliser à partie d'un espace de mémoire partagé, de deux sémaphore permettant l'accès à plusieurs processus, il ressoit les demandes des clients, se fait lire et vider par le serveur.

## serveur

Le serveur va en récupérant la demande venant d'un client via la file synchronisée, exécuter la commande voulus dans un nouveau processus avec les pipes fournis comme entrée et sortie standard.

## client

Le client créer deux pipes pour l'entrée et la sortie standard, ajoute sa demande à la file d'attente, écoute et envois ses donnée sur les pipes.

## 2 File synchronisée

La file synchronisée utilise une structure de donnée pour permettre l'accée à ses fonctions :

La structure stocke le nom de l'espace de mémoire partagé et ses droits. la liste des fonctions implémenter :

**file \_vide(void)** Créer une structure correspondant à une file vide avec des éléments de taille fixe de taille size.

Renvois NULL en cas de dépassement de capacité mémoire ou si size  $\leq 0$

sinon renvoie l'adresse d'un descripteur permettant d'y accéder.

**file \_ouvre(const char \*name, int oflag, mode\_t mod)** Ouvre un espace mémoire existant et renvoie une structure afin d'y accéder avec les autres fonctions de la librairie.

Renvois NULL en cas d'erreur.

**file \_ajout(const info \*f, const void \*ptr)** . Ajoute l'éléments pointer par ptr à la fin de la file décrit par f.

Si la file est pleine, tente de doubler la taille de la liste, en cas d'échec devient bloquant en attendant que la file se vide.

Renvois NULL si ptr == null ou en cas de dépassement de capacité mémoire, sinon renvoie ptr.

**file \_retirer(info \*f)** Défile la file s, renvoie nulle si la file est vide sinon renvoie l'adresse d'une copie de l'élément qui était au début de la file, la libération de la mémoire liée à cet élément est laissée à la discrétion de l'utilisateur.

L'opération est bloquante tant en attendant de pouvoir dépiler un élément.

**file \_est\_vide(const info \*f)** Renvois vrai ou faux si la file s est vide.

**file \_vider(info \*f)** libérer l'espace de mémoire partagé et détruit les sémaphores, renvoie -1 en cas d'erreur

La file est implémentée avec une liste flexible dans un espace mémoire partagé, la liste peut voir sa taille doubler en cas de besoin pour permettre un plus grand nombre d'éléments.

Le créateur de la file doit définir une taille aux éléments étant mis dans la file. Pour permettre l'accée à plusieurs clients, un algorithme producteur consommateur est implémenté avec 3 sémaphores stockés dans l'espace de mémoire partagé.

### 3 Serveur

Le serveur lis la file synchronisée pour avoir les différentes commandes à exécuter, quand une commande est ressus il créer un nouveau thread qui aura pour tache de s'en occuper.

Les demandes sont transmise à traver un structure regroupant diverses informations :

**argv** Liste des arguments de la commande, inclus la commande elle même

**envp** Arguments d'environement de la commande.

**tube\_in** Nom du tube à relier à l'entrée standard de la commande.

**tube\_out** Nom du tube à relier à la sortie standard de la commande.

Les tubes ressus ouvert dans l'ordre :

**tube\_in** : entrée standard permétant au client d'envoyer plus d'information à la commande.

**tube\_out** : sortie standard permétant au client de recevoir les messages de la commande.

L'ouverture des tubes est à la discrétion du client.

Dans le cas d'une commande transmise invalide, le serveur coupe le tube de retour sans rien écrire dedans. Le serveur ne se stope que dans le cas d'une erreur ou d'un signal d'arrêt par l'utilisateur. Le serveur redéfinis le comportement des certains signaux.

**SIGINT** Libère la mémoire partager avant de stoper le programme.

**SIGCHLD** Wait l'enfant afin de gérer les zombies.

**SIGCHLD** Libère la mémoire partager avant de stoper le programme.

## 4 Client

Le client est le programme qui gère l'interaction avec l'utilisateur, cela signifie qu'il passe par des entrées claviers. Le choix à été fait de prendre les demandes après l'exécution via une similitude d'interface graphique. Afin de pouvoir communiquer des pipes de communication à la commande demander par l'utilisateur le logiciel crée deux pipes à partir d'un nom type et de son identifiant afin de tenter d'obtenir des nouvelles pipes pas encore créées.