




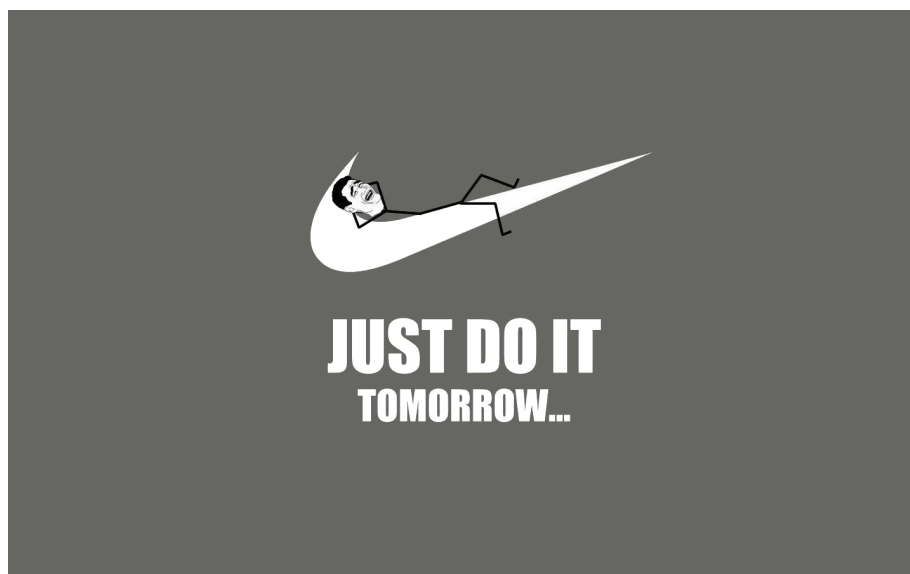
# Compte Rendu Global

## Systeme D'exploitation

Samir AKNINE, Antoine GRÉA

### ✿ Consignes & Conditions D'évaluation

- ✓ Vérification de la *compréhension* des sujets abordés.
-  Explication *rigoureuse* de la démarche et des résultats.
- Présentation et rédaction claire, précise, organisée et *synthétique*.
-  Rendu sur Spiral d'un **zip** par **binôme** nommé **SE2017.NOM1&NOM2.zip** avec un fichier **doc/rapport.pdf** de compte-rendu, un dossier de code source organisé *afin que le script fonctionne*.
-  Date limite du 2 **juin 2017** avant minuit.
- ! Tout non-respect des consignes sera synonyme de  
● **TRAVAIL NON RENDU !**



## Exercice 1 Console & Processus

### 1.1 Initiation À Linux

#### 1.1.1 Traitement De Flux

- ? Expliquez la différence entre *redirection* et *tube*. En quoi ceci est-il lié aux descripteurs de fichiers ?

#### 1.1.2 Finger

- ? Donnez *deux* différentes commandes capable de donner la liste des utilisateurs qui ont un UID supérieur à **1000** d'un système ainsi que leur UID

### 1.2 Processus

**C**eci est à faire dans le contexte de l'exercice **2** du **TP2**. Mettez bien les sources dans l'emplacement `src/killbill.c` dans l'archive.

#### 1.2.1 Fork

- ? Dans le programme suivant expliquez les conséquences des appels des fonctions **fork**, **wait**, et **exit** sur la mémoire, le pointeur d'exécution et l'état de chaque processus. Donnez des détails. *Encore plus de détails !*

```
int main(int argc, char *argv[]) {
    int a, e;
    a = 10;
    if (fork() == 0) {
        a = a * 2;
        if (fork() == 0) {
            a = a + 1;
            exit(2);
        }
        printf("%d \n", a);
        exit(1);
    }
    wait(&e);
    printf("a : %d ; e : %d \n", a, WEXITSTATUS(e));
    return(0);
}
```

- ? Réalisez un diagramme de séquence de ce programme avec chaque processus comme acteur. Indiquez clairement ce que chaque processus affiche.

### 1.2.2 Kill Bill

- ?
- Créez un programme appelé `killbill` qui est capable de tuer le zombie ainsi créé. Vous utiliserez uniquement le signal `SIGBUS` et le PID donné lors de l'exécution du zombie.

### 1.2.3 Comprendre Les Zombies

- ?
- Vous expliquerez ce qu'est un processus zombie, comment ils adviennent et ce qui est nécessaire pour les éliminer. Vous expliquerez également la raison du premier «GRRRRRR» lors de l'exécution du programme zombie et ce qui se produit lors de son élimination.

#### ⊕ Bonus

- ?
- Créez un programme `chucknorris` capable de chercher et d'éliminer tous les zombies d'un système.
- ?
- Faites le programme `killbill` en intégrant la compilation, le code C entier et l'exécution du binaire dans un script `bash` d'une ligne.

## Exercice 2 Synchronisation

### 2.1 Puzzle De Caractères

**C**eci est à faire dans le contexte de l'exercice 2 du **TP3**. Mettez bien les sources dans l'emplacement `src/puzzle.c` dans l'archive.

#### 2.1.1 Sémaphores

- ?
- Réalisez l'affichage séquentiel avec pas plus de 7 sémaphores. Comment doivent être placés les déclarations et appels de sémaphores dans votre programme ?

#### 2.1.2 Preuve

- ?
- Fournissez une preuve *mathématique* formelle que votre programme ne contient pas d'interblocage. À partir de cette preuve montrez que votre solution garantit qu'un seul processus peut utiliser l'affichage à la fois.

## ⊕ Bonus

- ❓ Réalisez une simulation de feux de circulation ou de file d'attente chez le boucher afin de vous préparer pour ces exercices types du *partiel*.

## Exercice 3 Tubes

### 3.1 Combien De Processus ?

- ❓ Créez un programme `src/count.c` qui utilise les tubes et les duplications de descripteurs de fichier pour répliquer la commande `ps -ef | wc -l`.

### 3.2 Long Long Long

**C**eci est à faire dans le contexte de l'exercice 2 du **TP4**. Mettez le code dans le fichier `src/mario.c`

#### 3.2.1 Débouche Tuyaux

- ❓ Créez le programme qui nettoie l'entrée du programme `self`. Reliez le tube de manière à rendre le programme opérationnel.

## ⊕ Bonus

- ❓ Expliquez votre interprétation de ce que fait le programme `self`. Testez votre théorie en modifiant les données d'entrée.

(Indice : le programme affiche un tracé de fonction mathématique)

## Exercice 4 Mémoire Partagée

### 4.1 À Ne Pas Oublier

#### 4.1.1 C'est Bien De Partager

- ❓ Expliquez pourquoi on a besoin de mémoire partagée entre processus. Comment est protégée la mémoire des processus ?

#### 4.1.2 Se Mettre À La Page

- ❓ Que se passe-t-il lorsque l'on dépasse la taille que l'on a alloué dans une mémoire partagée ? Expliquez cet effet.

## 4.2 Morpion

**C**eci est à faire dans le contexte de l'exercice **2** du **TP5**. Vous déposerez le code dans le fichier `src/morpion.c`

### 4.2.1 Rien Ne Va Plus

? Concevez le programme du morpion. Comment gérer les attentes et la synchronisation avant l'accès mémoire ? L'arbitre peut-il lutter contre la fraude ?

#### ⊕ Bonus

? Que se passe-t-il lorsqu'un programme demande plus de mémoire que disponible ? Expliquez ce mécanisme avec un exemple.

*Bonne Chance !*