

# Serveur Web Minimaliste

MARGIOTTA Adrien && DAVID Bertrand;

17 mars 2008

## Table des matières

<b>I</b>	<b>Déroulement du projet</b>	<b>1</b>
1	Objectifs	2
2	Buts	2
3	Difficultés rencontrées	2
3.1	Interception et traitement des signaux avec les handlers . . . . .	2
3.2	Fermeture des sockets pere et fils . . . . .	2
<b>II</b>	<b>Documentation technique du projet</b>	<b>2</b>
4	Fonction existOrNot()	3
5	Les handlers : interception et traitement des signaux	3
5.1	Principe du handler . . . . .	3
5.2	Utilité des handlers dans notre projet . . . . .	3
6	Gestion de l'erreur 404	3
6.1	Fonctionnement . . . . .	3
6.2	Créer sa propre page 404 . . . . .	4
7	Résolution des problèmes posés	4
7.1	Serveur de fichier . . . . .	4
7.2	Serveur d'image . . . . .	5
7.3	Générateur de page . . . . .	5

# Première partie

## Déroulement du projet

### 1 Objectifs

Nous avons pour mission d'implémenter un serveur Http minimaliste qui soit capable de faire 3 tâches pour répondre pleinement au sujet :

- Serveur de fichier
- Serveur d'image
- Générateur de page statistiques

Le langage retenu pour cette implémentation fût le langage C.

### 2 Buts

Les tenants et aboutissants de ce projet étaient de mettre en application des concepts vus en TD/TP ainsi que de s'appropriier de nouveaux concepts :

- Programmation C réseaux avec utilisation des sockets unix.
- Utilisation de certains appels systemes : `read()`, `write()`...
- Interception de signal : les handlers. Utilisés pour fermer une socket et tuer les processus zombies.

### 3 Difficultés rencontrées

#### 3.1 Interception et traitement des signaux avec les handlers

Nous avons rencontrés des problèmes dans la mise en place de handlers. Ce mécanisme avait été rapidement évoqué en cours de SE mais jamais vraiment testé. Enfin, à force de patience et perseverance, nous sommes fiers de dire que nos handlers sont pleinement fonctionnels et permettent :

- de gérer une sortie de programme "propre" en cas d'interruption brutale du programme (`ctrl+c` par exemple).
- de tuer les eventuels processus zombies.

#### 3.2 Fermeture des sockets pere et fils

Nous avouons, sans complexe, être restés bloqués un certain temps sur un problème, que nous etions presque arriver à qualifier d'insolvable. Le client n'affiché la page qu'une fois le programme quitté. Grâce à l'aide d'un de nos gourous, nous avons réussi à résoudre le problème : nous ne fermions pas les sockets ouvertes par la connexion des clients.

## Deuxième partie

# Documentation technique du projet

## 4 Fonction existOrNot()

Cette fonction teste l'existence ou non d'un fichier donné et renvoie un entier en fonction du résultat<sup>1</sup>. Cette fonction a été codé dans le but de faciliter le codage et sa lecture au moment de gérer l'erreur de type 404<sup>2</sup>(cf 6-Gestion de l'erreur 404).

## 5 Les handlers : interception et traitement des signaux

### 5.1 Principe du handler

Le concept est simple : tout événement engendrant la génération d'un signal est attrapable dans le but d'y effectuer un traitement. La structure du handler se fait en deux parties :

- Armement du signal : instant t à partir duquel le handler est à "l'écoute" d'un signal.
- Traitement du signal : série d'opérations déclenchées par la génération du signal si celui ci est armé.

### 5.2 Utilité des handlers dans notre projet

Nous avons eu besoin des handlers pour deux raisons :

- contrôler une sortie "propre" du programme en cas d'interruption brutale de ce dernier.
- Tuer les processus zombies qui peuvent apparaître.

## 6 Gestion de l'erreur 404

### 6.1 Fonctionnement

L'erreur 404 correspond à une requête sur un fichier inexistant. La manière dont une requête sur un fichier est traitée est expliqué dans l'algorithme "getFile()".

---

<sup>1</sup>existOrNot() renvoie 0 si le fichier existe et 1 dans le cas contraire.

<sup>2</sup>Erreur 404 : Page not found

---

**Algorithm 1** getFile(path : String)

---

```
BEGIN
If(!existOrNot(path))
  If(equals(path,"page"))
    //return a stats page
  Else If(existOrNot(default404)!=0)
    //generation of a standart 404 page
    write(404page);
  Else
    /* default404 is there if 404.html was created by user in current folder */
    write(404defaultPage);
Else
  //Return the page
```

---

---

**Algorithm 2** sendFile(myFile : File)

---

```
monOctet : char;
len : entier;
//char is encode on 8 bits
BEGIN
  Do
    len = readOneOctet(myFile, monOctet);
    send(monOctet)
  While(len>0)
END
```

---

## 6.2 Créer sa propre page 404

Comme indiqué dans l'algorithme getFile(), le programme prévoit le fait que l'utilisateur ait l'envie de construire ses propres pages d'erreurs. Pour se faire, nous avons juste ajouté un test : Si la page n'existe pas, alors le programme teste qu'il n'y est pas une page "404.html" qui existe auquelle cas, c'est cette dernière qui est renvoyée.

## 7 Résolution des problèmes posés

### 7.1 Serveur de fichier

Nous sommes partis du TP sur la création d'un serveur TCP car http utilise le protocole TCP. Le principe était simple puisqu'il nous a suffi de lire octet par octet le fichier voulu par la fonction fread() puis de l'envoyé octet par octet grâce à la fonction write() selon l'algorithme sendFile().

## 7.2 Serveur d'image

Nous avons conçu le serveur de fichier de telle manière qu'on est peu de modifications à faire pour le transformer en serveur d'image. Et en effet, nous n'avons eu aucune modification à faire puisque notre serveur de fichier peut également afficher des images.

## 7.3 Générateur de page

Nous avons utilisés une série de fonctions pour afficher divers statistiques :

- `getHostName()` : donne le nom de la machine hébergeant le serveur.
- `getSockName()` : donne diverses informations :
  - IP
  - Port
- `time()` : donne l'heure.