

SI4 – Projet

Virtual operating system & file system

Platform development

Description and features

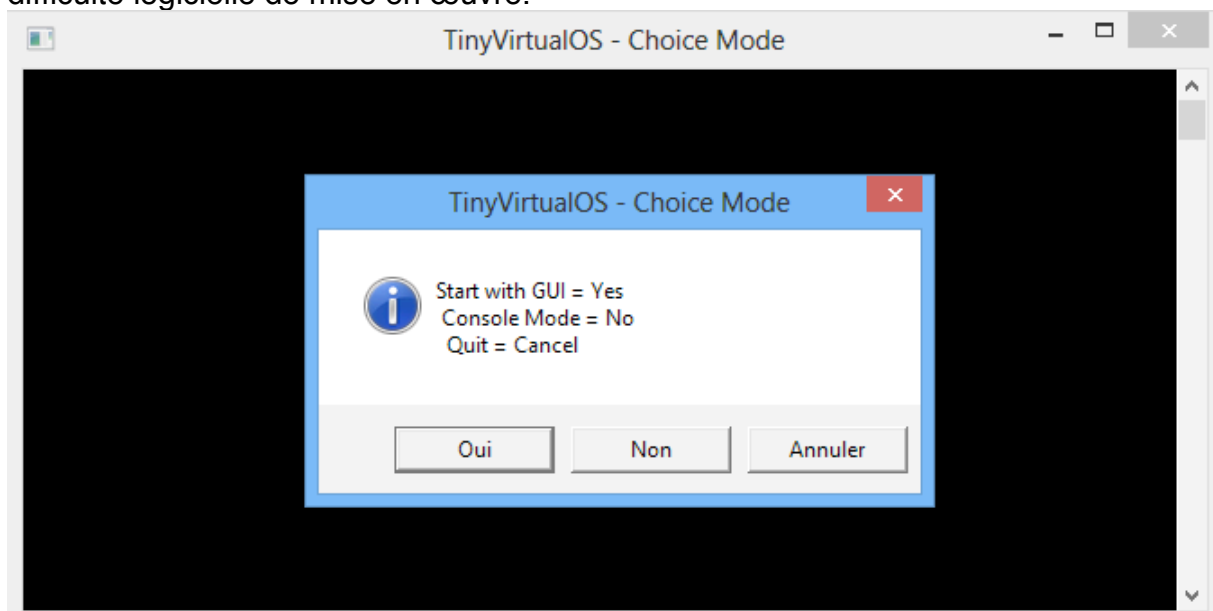
Ces quelques pages permettent de mettre en œuvre à travers la programmation en C les principes des OS et des SGF.

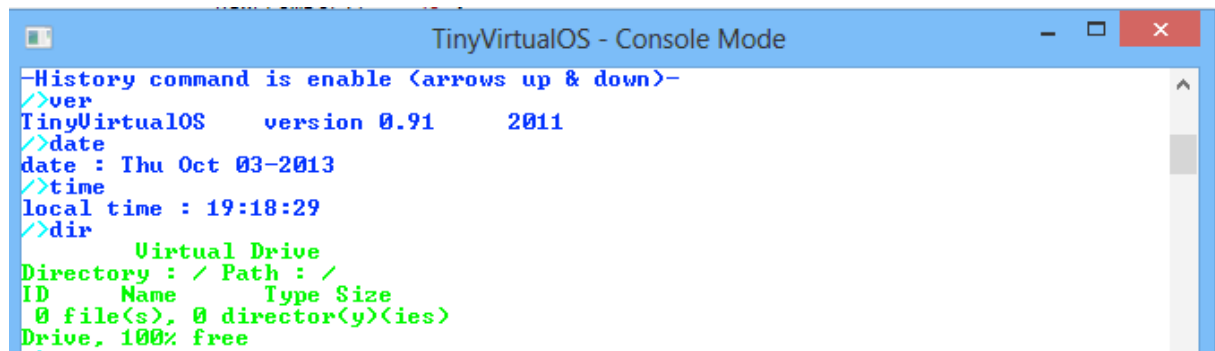
Ce projet permet de mettre en pratique toute une série d'algorithmes basés sur la mise en place d'un système virtuel. Il permet en l'occurrence ici, de manipuler la notion de structures. Dans la partie finale, la mise en place d'un OS virtuel, permet de manipuler le système de fichier qui sera construit.

Des notions de chaînage de structures sont présentes dans ce dossier, ce qui permet de mettre en évidence les arbres et les structures hiérarchiques. Il est même possible de mettre en pratique les tris de données, afin d'avoir des résultats lisibles, mais aussi pour optimiser le fonctionnement des différentes opérations.

Ce système est une version inspirée du système MS-DOS, qui a prouvé son efficacité par le passé. Certains concepts sont également empruntés à d'autres systèmes comme NTFS ou extFS.

Attention : les caractéristiques présentées dans ces pages sont susceptibles d'être modifiées si leur pertinence venait à être mise en cause, sur un « bug » ou une difficulté logicielle de mise en œuvre.





```
TinyVirtualOS - Console Mode
-History command is enable (arrows up & down)-
>ver
TinyVirtualOS    version 0.91    2011
>date
date : Thu Oct 03-2013
>time
local time : 19:18:29
>dir
    Virtual Drive
Directory : / Path : /
ID      Name      Type Size
 0 file(s), 0 director(y)(ies)
Drive, 100% free
```

1. le SGF

Le système présenté repose sur un répertoire racine, et sur une section de données. La section de données inclut des « streams », c'est-à-dire les flux de données des fichiers. Ce concept permet d'unifier la table d'allocation avec les secteurs de données.

Cette simplification permet de supprimer le bloc data qui normalement fait suite à la FAT.

Dans notre cas aucun mécanisme ne permet d'assurer la sécurité des données.

Le répertoire racine, permet 65535 entrées. L'entrée 0 est réservée au système pour spécifier ses caractéristiques. Cela permet la prise en charge de versions futures, et la compatibilité avec les versions anciennes.

La structure du répertoire racine est la suivante :

Structure rootDirectory (Seul le premier enregistrement est réservé au système)

rdID : entier // identifiant 0 à 99 réservés

rdName : chaîne de caractères[11]

rdExtension : chaîne de caractères[4] // une entrée fait 15 caractères soit 17 entrées LFN pour 255 caractères (LFN voir plus loin)

rdType : caractère // F : fichier non LFN, D : Directory, L : LFN file, d'autres besoins peuvent être implémentés. (LFN voir plus loin)

rdSize : entier // taille du fichier, la taille doit être tenue à jour

rdTime : undefined // gestion de l'heure dernier accès

rdDate : undefined // gestion de la date dernier accès

rdFirstSector : entier // ID du premier enregistrement du fichier courant dans la section de données, 0 si dossier.

rdParent : entier // Permet de pratiquer la hiérarchie des dossiers avec des fichiers. La topologie du lecteur virtuel n'est donc pas plate. rdParent = 99 correspond à la racine

rdAttrib : entier // Non utilisé pour l'instant, gestion des droits d'accès, attribut de fichier

rdParent contient le numéro de « directory » parent pour le fichier ou dossier courant. Attention, les numéros commencent à 100. Les valeurs de 0 à 98 sont réservées

pour des usages spécifiques. Les éléments avec le numéro 99 sont situés à la racine du lecteur.

rdSize : La taille du fichier doit être gérée en permanence afin d'avoir une cohérence entre la taille affichée et la taille occupée. Il est nécessaire de gérer tous les débordements.

Le point « . » entre le rdName et rdExtension ne figure pas dans la structure, il est ajouté à l'affichage du nom du fichier (sauf pour les fichiers LFN qui enregistrent ce point).

La date de modification, ainsi que l'heure, ne sont pas pris en compte ici. L'attribut du fichier pourra permettre une gestion des accès, voir de l'affichage dans la hiérarchie.

Le répertoire racine assure une structure hiérarchique par le chaînage dans un seul sens des différents dossiers. Les fichiers sont intégrés de la même manière dans ce processus.

La section de données autorise également 65535 entrées. L'entrée 0 est réservée pour des besoins spécifiques.

La section de données est décrite avec la structure suivante :

Structure dataSection

dsID : entier // identifiant d'un enregistrement du fichier qui commence à 100.

dsData : chaîne de caractères[512] // Les données du fichier concernant un « stream »

dsPrev : entier // Previous record, 0 = pas de précédent

dsNext : entier // next record, 0 = pas de suivant

La section de données permet donc un chaînage des enregistrements d'un fichier, ce qui permet au système de renvoyer correctement le contenu d'un fichier. Le chaînage se fait dans les deux sens, ce qu'il facilite les parcours.

La gestion des LFN, se fait par le processus suivant. Les entrées du répertoire racine servent à stocker le nom long du fichier. Le point qui sépare nom et extension est alors stocké. Il faut déjà remplir le champ nom puis le champ extension. Si cela n'est pas suffisant une autre entrée est créée automatiquement, avec comme parent le rdID du fichier initial. La taille d'un nom long ne peut excéder 255 caractères. Attention, il est donc possible de remplir le répertoire racine avec seulement 3855 fichiers sur le lecteur. Un nom long pouvant occuper au plus 17 entrées.

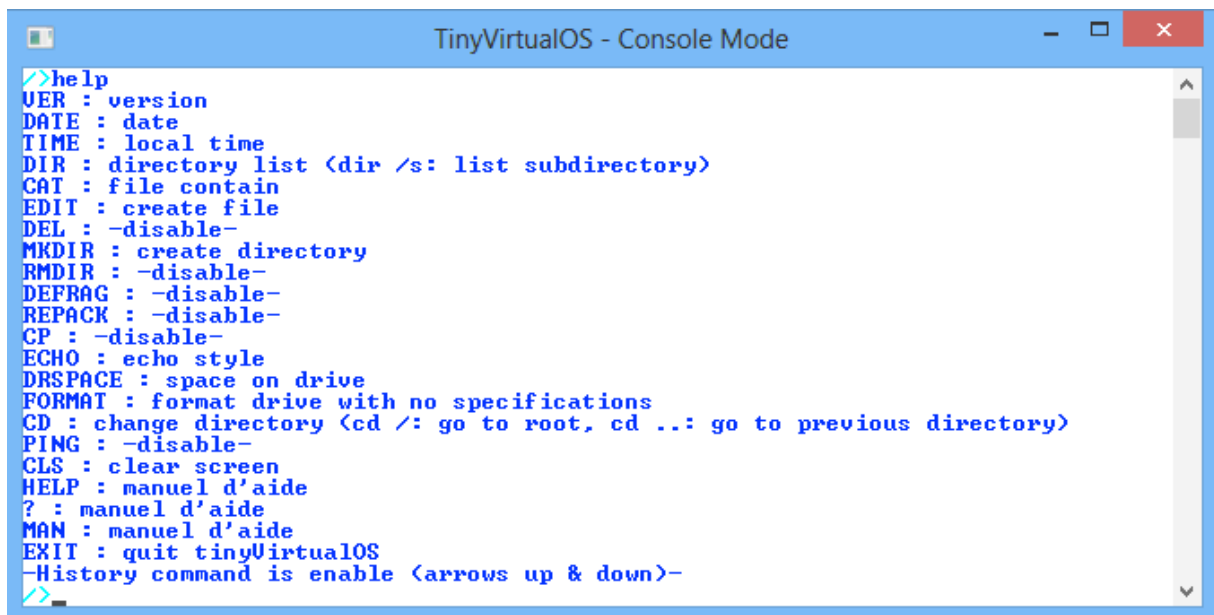
Seule la première entrée longue contient les spécifications du fichier. Les entrées suivantes ne contiennent pas d'information sur le fichier.

La taille utile de ce lecteur est de $65535 * 512 = 33553920$ octets. La création d'une hiérarchie avec les dossiers consomme de la place avec la technique présentée, il est donc préférable de placer tous les fichiers à la racine afin d'optimiser la place disponible.

2. L'OS

Le système d'exploitation virtuel autorisera des opérations utilisant le FS. Les commandes suivantes seront implémentées :

VER : version logicielle
DATE : date
TIME : heure
DIR : liste des dossiers et fichiers du répertoire courant
CAT : contenu d'un fichier
EDIT : création d'un fichier avec son contenu
DEL : effacement d'un fichier
MKDIR : création d'un répertoire
RMDIR : suppression d'un répertoire
DEFRAG : défragmentation du lecteur, au niveau rootDirectory et dataSection
REPACK : optimisation de l'espace libre, au niveau rootDirectory et dataSection
CP : copie de fichier
ECHO : affichage d'un message
DRSPACE : affiche la taille occupée et la taille disponible en nombre d'octets
FORMAT : formatage du lecteur, initialisation.



```
TinyVirtualOS - Console Mode
>help
VER : version
DATE : date
TIME : local time
DIR : directory list (dir /s: list subdirectory)
CAT : file contain
EDIT : create file
DEL : -disable-
MKDIR : create directory
RMDIR : -disable-
DEFRAG : -disable-
REPACK : -disable-
CP : -disable-
ECHO : echo style
DRSPACE : space on drive
FORMAT : format drive with no specifications
CD : change directory (cd /: go to root, cd ..: go to previous directory)
PING : -disable-
CLS : clear screen
HELP : manuel d'aide
? : manuel d'aide
MAN : manuel d'aide
EXIT : quit tinyVirtualOS
-History command is enable (arrows up & down)-
>
```

Il est possible d'inventer toutes sortes de commandes. Vous pouvez même ajouter des paramètres à vos commandes pour les rendre plus performantes.

Les commandes seront lancées dans une console à partir d'un prompt, pour simplifier le codage. L'usage d'une interface graphique se contentera d'appeler les commandes lignes déjà configurées. Un interpréteur pourra traiter le lexique, la syntaxe voir la sémantique. Un exemple d'interpréteur est disponible pour l'AP « interpréteur de fonction ».

3. Jeu d'essai

RootDirectory (tableau d'enregistrements, les attributs rdAttrib ne sont pas précisés)

rdID	rdName	rdExt	rdType	rdSize	rdTime	rdDate	rdFirstS	rdParent
0	"version ini"	"1.0"	10	9	-	-	4	5
100	"IG1"	"	D	0	-	-	0	99
101	"DAIGL"	"	D	0	-	-	0	100
102	"ALSI"	"	D	0	-	-	0	100
103	"exercice_tp"	"1.txt"	L	4	-	-	100	101
104	"exo.sauvega"	"rde"	L	4	-	-	101	101
105	"l'exercice_"	"d'als"	L	1024	-	-	102	102
106	"i_en_sql.tx"	"t"	L	0	-	-	0	105
107	"theApp"	"cpp"	F	124	-	-	104	99
108	"IG2"	"	D	0	-	-	0	99
109	"test"	"txt"	F	10	-	-	105	108

Pour rdID = 0, rdType = nombre d'élément brut, rdSize = nombre d'élément réel, rdFirstS = nombre de dossier, rdParent = nombre de fichier.

A la lecture des données, les rdParent = 99 sont à la racine. Les entrées rdType = L sont converties en fichiers standards.

Il faut faire attention pour les entrées successives L qui désignent un nom de fichier long. Des tests doivent être effectués pour voir si l'entrée suivante est bien celle d'un fichier long, afin de reconstituer le nom.

Les entrées fournies dans cet exemple, sont placées dans l'ordre de saisie. Il est nécessaire de bien comprendre qu'une réorganisation automatique est nécessaire pour simplifier la gestion des affichages.

L'arborescence est la suivante :

Root	size	date	time	attrib
+IG1		-	-	-
+DAIGL		-	-	-
->Exercice_tp1.txt	4	-	-	-
->Exo.sauvegarde	4	-	-	-
+ALSI		-	-	-
->L'exercice_d'alsi_en_sql.txt	1024	-	-	-
+IG2		-	-	-
->Test.txt	10	-	-	-
->theApp.cpp	124	-	-	-

Nous avons 4 dossiers, et 5 fichiers

La commande DIR, peut éventuellement donner une présentation similaire. Attention un fichier qui aurait été caché n'apparaîtrait pas ici.

```

C:\>dir
           Virtual Drive
Directory : / Path : /
ID      Name      Type Size
100 test.          F 55 byte(s)
1 file(s), 0 director(y)<ies>
Drive, 99% free
C:\>_

```

La commande DRSPACE peut donner le résultat suivant :

```
>DRSPACE
```

```
Taille disponible : 33552754 octets/33553920 octets soit 99 % d'espace disponible
4 dossiers, 5 fichiers
```

```
>_
```

```

C:\>drspace
Drive, 99% free
C:\>_

```

Voyons maintenant la section de données

dataSection (tableau d'enregistrement)

dsID	dsData	dsPrev	dsNext
0	"cluster dimension, stream number"	512	6
100	"2x+1"	0	0
101	"2x+1"	0	0
102	"-----512 octets-----"	0	103
103	"-----512 octets-----"	102	0
104	"-----124 octets-----"	0	0
105	"void main("	0	0

Pour dsID = 0, dsData = description des variables, dsPrev = taille du cluster, dsNext = nombre de stream.

La section de données contient bien les « streams » des fichiers. Pour une taille < 512 octets le fichier ne contient qu'un seul enregistrement. Si la taille dépasse 512 octets, il est nécessaire d'effectuer le chaînage des enregistrements.

La commande CAT test.txt donnera le résultat suivant :

```

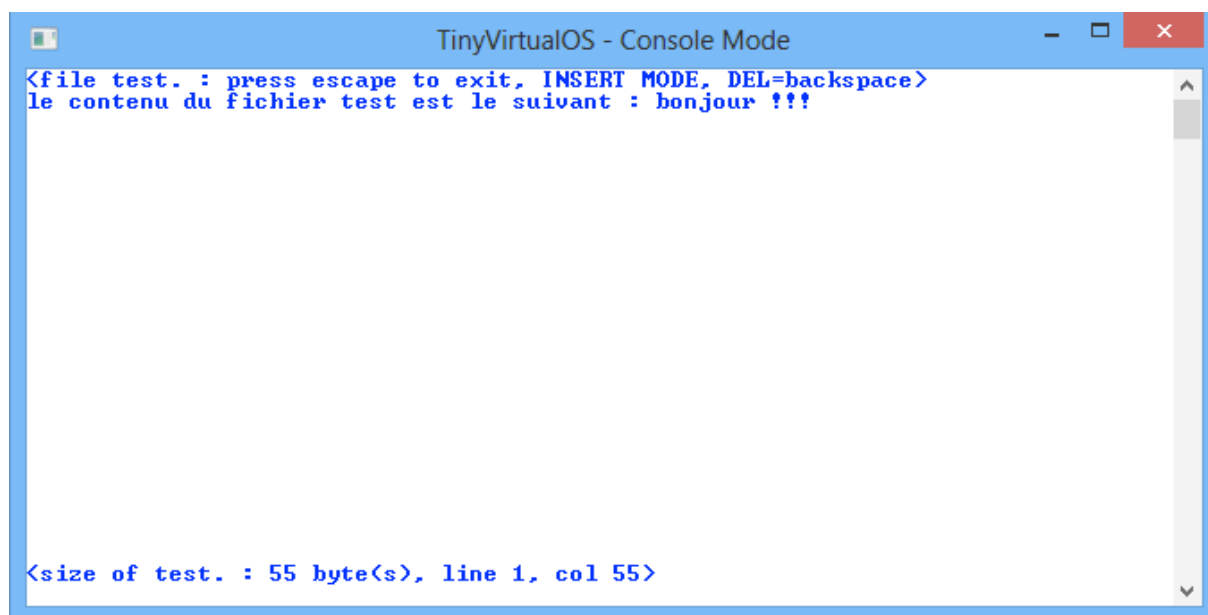
>cat test.txt
void main(
>_

```

```

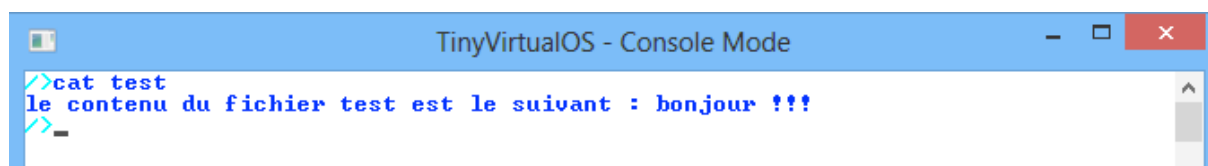
C:\>edit test_

```



```
TinyVirtualOS - Console Mode
<file test. : press escape to exit, INSERT MODE, DEL=backspace>
le contenu du fichier test est le suivant : bonjour !!!

<size of test. : 55 byte(s), line 1, col 55>
```



```
TinyVirtualOS - Console Mode
>cat test
le contenu du fichier test est le suivant : bonjour !!!
_
```

Pour les petits fichiers < 512 octets, ce qui est souvent le cas pour les fichiers des OS, la perte de place est relativement importante. Dans tous les cas des emplacements de 512 octets de données sont la taille la plus petite constatée. Dans notre cas, notre système étant expérimental, il faudrait pouvoir choisir des longueurs plus petites de 256 voir 128 octets.

Attention : Pour toutes les structures, les lignes avec rdID et dsID doivent être mises à jour afin de simplifier la gestion des opérations, et également pour être en cohérence avec la place vraiment occupée.

4. Conclusion

Ce travail dépasse largement le cadre unique du cours de SI4, il peut faire appel à des compétences transversales. Il peut faire l'objet d'un travail sérieux, afin de se familiariser avec les principes des OS et des SGF.

Présenté sous forme procédurale, il est possible de le convertir en version objet. L'ajout d'une interface graphique, associée à des commandes performantes peut en faire un outil virtuel de simulation à illustrer dans un dossier.



