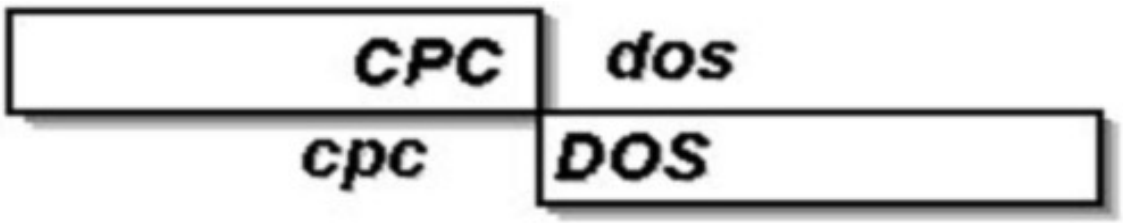


Microsf01 CpcdosC+

Manuel Doc.1.2013

(En cours - 26/03/2013)

Apprendre la programmation CpcdosC+ pour le Kernel Cpcdos OSx

- Version OS2.0.5 -

Programmation base en fichiers de commandes et interprétation en console

- Tutoriels & exemples -

Favier Sébastien 01

http://microsf01.fr.nf | http://cpcdos.fr.nf/

sebastien.ordinateur@hotmail.fr

Sommaire :

- Afficher un MSGBOX - - - - - `msgbox/`
- Fermer une fenêtre ou objet - - - - `fermer/`
- Actualiser une ou plusieurs propriétés - - `actualise/`
- Focus sur une fenêtre - - - - - - `focus/`
- Lancer le mode console - - - - - `lc/`
- Lire ou éditer le registre - - - - - `reg/`

- Configurer & Tester le système – **sys/**
  1. Résumé système (Application externe) - **/quick**
  2. Tester le Kernel - **/krnltest**
  3. Status du Cache CPU - **/cpu_cache**
  4. Level du CPU - **/cpu_level**
  5. Compatibilité du VESA - **/testvesa**
  6. Lister modes d'écran&bit - **/testecr**
  7. Initialiser le Kernel&SCI - **/ini_krnl**
  8. Tester la mémoire Etendu (XMS) - **/memtest**
  9. Nettoyer la mémoire - **/memclear**
  10. Tester le CPU - **/test**
  11. Appel interruptions DOS - **/int**
  12. Écrire & lire dans la mémoire - **/poke** & **/peek**
  13. Mémoire buffer - **/buffer**
  14.
  15.

- Variables environnement
  - Résolution d'écran & Bit & Fond bureau
  - Mémoire STACK
  - Informations DEBUG & Sortie+Affichage
  - Cadence processeur /100 %
  - Stop sur erreur
  - Informations OS & répertoires
  - Console
  -
  -

6. Exemples - - - - - - - - - - - - - Chapitre VI
    - Sortie console en Mode LC
    - Créations de fenêtres & d'objets en Mode IUG

Chapitre I - Histoire du CpcdosC+ *A REFAIRE !*

Le premier langage de programmation développé par Microsf01 était pour le 3eme Cpcdos sur Amstrad Cpc 464 il se nommait **CpcCmD** à la base il était écrite en Basic 1.0 en 2006 , ce langage ne servait pas a créer un programme , il était fait de façon à qu'il ressemble à l'interpréteur de commandes MS-DOS.

Le Second langage de programmation développé par Microsf01 était le CPCOMMAND sur Windows avec le l'OS virtuel Cpcdos 4.5 *écrit en Visual Basic 5 , Batch(ms-dos) , C++*, il servait de commutateur pour le programme cet à dire que Cpcdos 4.5 avait un noyau avec une interface préprogrammé directement dans le code , le noyau ne générai pas d'événement..

Ce langage contient des commandes qui ont la base de :

- IO (gestions de fichiers , lecture/écriture)
- Réseau (gestion de serveur IP et utilisation des protocoles de Windows avec le dossier de partage etc ...)
- Création d'application (fenêtres de Windows etc ...)

Il fonctionne avec des fichiers qui contient des commandes nécessaire au fonctionnement et démarrage de Cpcdos 4.5

Le 3eme (aujourd'hui) c'est le **CpcdosC+**

acronyme de **Cpcdos C**ommande+

initiales : CC+ ou CCP

Ce langage est utilisé dans 2 types

- CpcdosC+ pour systèmes d'opérations
- CpcdosC+ pour Jeux (Microsf01 Games 32 bit)

Nous allons l'étudier dans ce document

Chapitre II – CpcdosC+ ? *A REFAIRE !*

Un tas d'explication à la page : **http://microsf01.e-monsite.com/pages/cpcdos-os2-1.html**

Le CpcdosC+ est un langage de programmation utilisé pour le noyau NMG (Noyau Microsf01 Games) , un noyau + console pour jeux vidéo DirectX

et pour le noyau Cpcdos.

Il est principalement utilisé pour développer sous le Kernel Cpcdos pour développer avec simplicité une interface conviviale , autonome , pratique

La sortie généré par le noyau peux être 'dynamiquement' être basculé , utilisé en mode LC et/ou en mode IUG

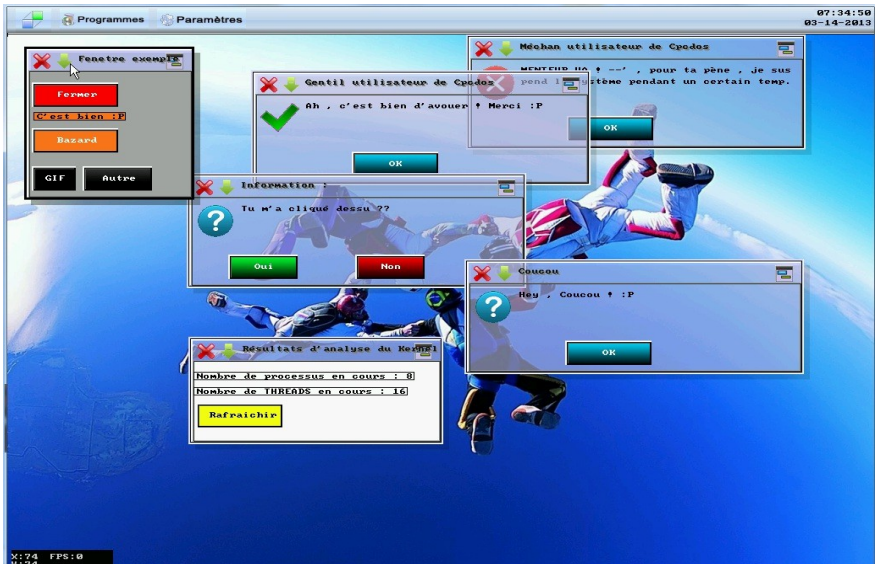
LC : Lignes de Commandes

IUG : Interface Utilisateur Graphique

le Kernel intègre une console (en LC ou dans une fenêtre en IUG) , au quelle elle peux interférer des événements , faire appelle aux services et procédures du Kernel , cet à dire que même , à partir d'une simple console , on peux créer une fenêtre (en IUG uniquement) avec des boutons , labels etc ..

et bien sûre , on peux faire appel à un fichier texte pour lancer des commandes.

Voici un exemple (en résultat) du codage avec le Kernel Cpcdos :

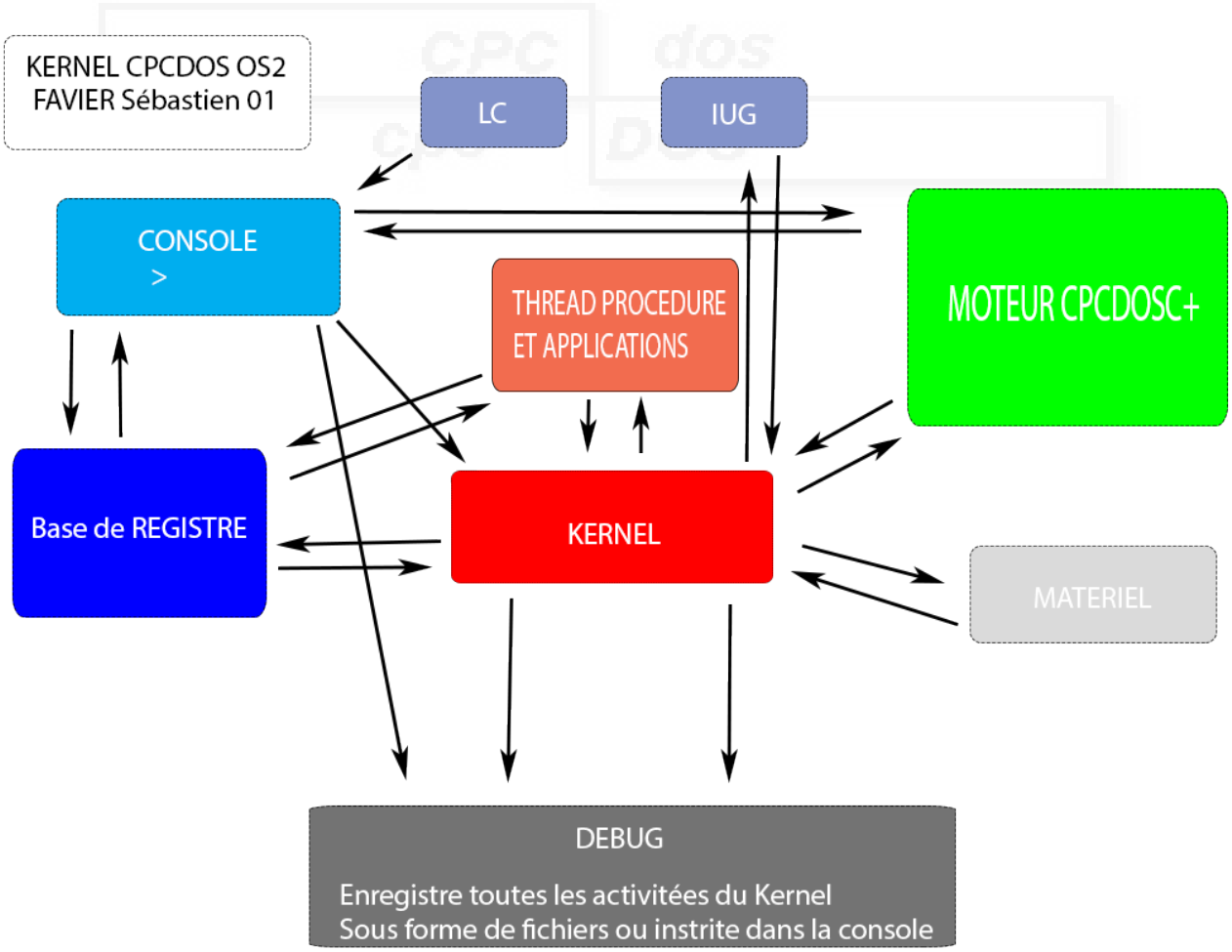


Une interface graphique simple avec un choix de résolution d'écran entre

320x200 , 320x240 , 320x400 , 320x480 , 400x300 , 512x384 , 640x350 , 640x400 , 640x480 , 720x480 , 720x576 , 800x600 , 1024x768 , 1152x864 , 1280x960 , 1280x1024 , 1600x1200 , 1920x1080 , 1920x1200 , 1920x1440 , 2048x153

et un choix de couleurs entre 8 , 16 , 32 bit .

Schéma du fonctionnement du Kernel Cpcdos OS2 :



-

LC (Lignes Commandes) :

Partie en lignes de commandes semblabe à la partie console
Il permet l'introduction de l'interface "Console" en ligne de commandes

IUG (Interface Utilisateur Graphique) :

Comme son nom l'indique , c'est tout simplement l'interface graphique où l'utilisateur interagit aux objets etc...

Console :

Partie où l'utilisateur entre ses commandes CpcdosC+
et peux être aux commandes de la partie Kernel

Base de REGISTRE :

Partie du système , il fournis et enregistre les informations et paramètres

système du Kernel.
Il est aussi liée a la Console car on peux interagir au registre via la console avec la commande *REG/*

THREAD PROCEDURE ET APPLICATION :

Partie où les information des propriétés et objets sont placée , et utilisé afin que la partie Kernel dessine
sur l'interface

MOTEUR CPCDOSC+ :

Partie du système où toutes les commandes CpcdosC+ sont analysée et éxécutées par la partie KERNEL

MATERIEL :

Partie où le Kernel gère le clavier , souris , affichage , imprimantes , USB etc...

KERNEL :

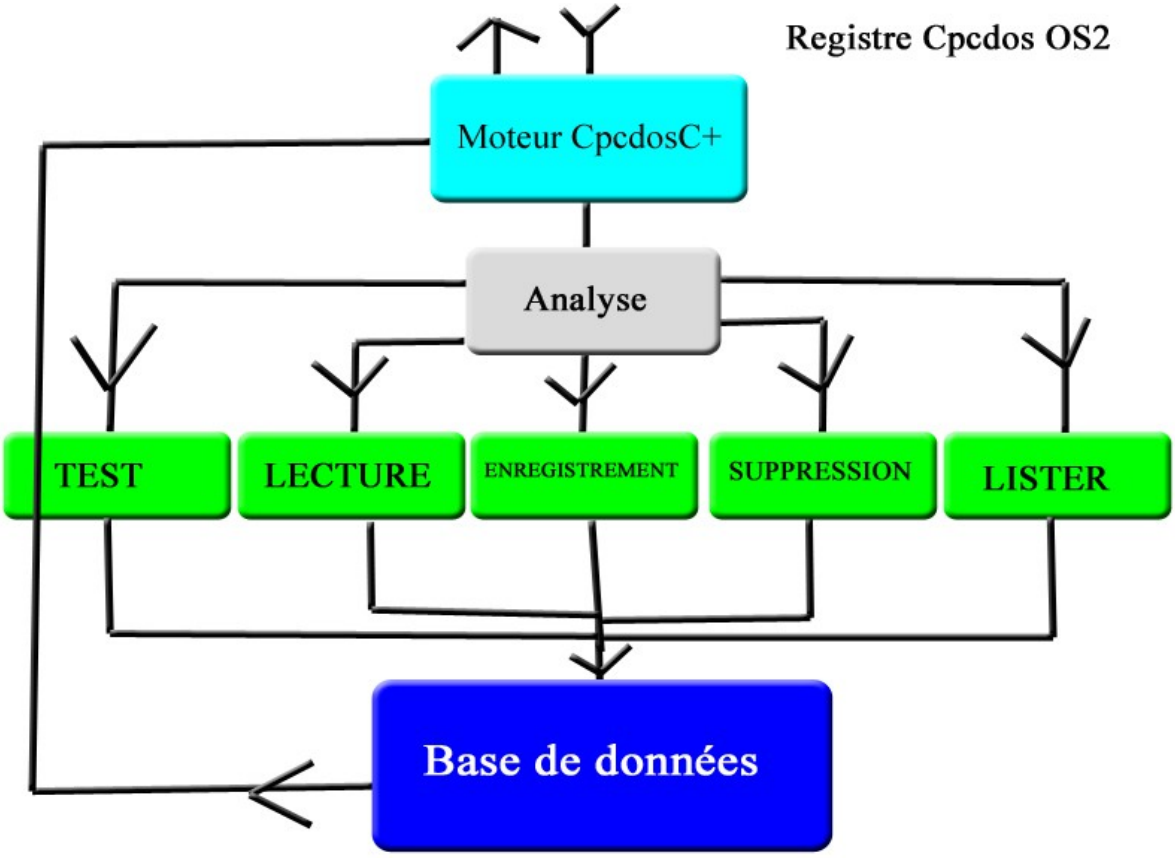
Partie NOYAU .. tout simplement celui "qui gère tout"..

DEBUG :

Partie assez importante qui enregistre ou affiche à la console , toutes le activitées du Kernel !

Nb : Le Kernel est une surcouche au dessus du DOS

Schéma du registre :



L’utilisation du code CpcdosC+ s’utilise dans 2 façons différentes !

La première dite CCB (Code Compilé Binaire)

Puis , CNB (Code Non Binaire)

C’est-à-dire que le CCB deviendra un exécutable direct au Kernel qui lui le code CpcdosC+ sera convertit Assembleur puis deviendra un fichier Binaire.

Et le CNB est comme un fichier de paramètres , indiquant juste au Kernel les fonctions d’interface avec du code de type texte , modifiable , personnalisable etc ..

Par exemple le CNB que l’utilisateur créera , pourra être un MsgBox , une simple fenêtre etc ..

Chapitre III - Requirements minimum

Avant tout , il faut faut bien évidement , un PC (et non un Mac !)

Configuration minimum requis pour le PC :

Processeur : 800 *mhz* Intel *80x86* ou Amd (AM286)

Ram : 256 mo

Carte graphique : 8 mo supportant le VGA , EGA , SVGA , VESA

Disque dure : au moins un 10 Go

Un processeur dans les allentours de

INTEL

Intel 8086

Intel 286

Intel 386

Intel (P5) : Pentium

Intel (P6) : Pentium II , Celeron , Pentium III

Intel (NetBurst) : Pentium 4

Intel (P6) Core 2 Duo

AMD

Amd Am286

Amd Am386

Amd K5

Systèmes BIOS

Phoenix , Award Software

(Energy)

Marques Systèmes

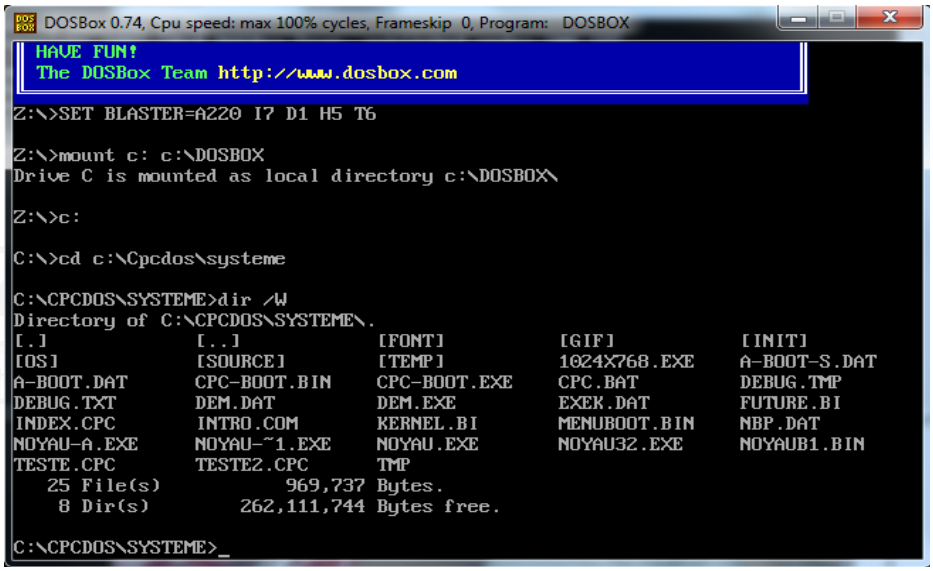
Acer , HP , Dell

(problèmes SVGA sur les PC IBM pour cause de la carte graphique interne)

Installation sur Disque dur :

{ Page réservé à l'installation du Kernel sur un disque dur }
Prochainement sur le Doc 2.2012

Installation avec DosBox :



Pour commencer , télécharger DoxBox sur cette page en choisissant votre Interface (Windows,Mac...)
http://www.dosbox.com/download.php?main=1
Puis installez-le.
Une fois installé , veuillez créer un dossier nommé « DOSBOX » dans **C:**
Puis copiez les fichier « Install.bat » , « RAR.EXE » , « KRNL_OS2.RA_ » dans **C:\DOSBOX**
Ensuite copiez et remplacez le fichier « dosbox-0.74.conf » dans **C:\Users\{nom d'utilisateur}\AppData\Local\DOSBox**

==========

NB :
Différences avec l'ancien fichier :
`cycles=auto` modifié en `cycles=max`
Puis , ajout de

```
# Pour créer le lecteur C: grâce au dossier DOSBOX puis aller au répertoire.
mount c: c:\DOSBOX
c:
```

==========

Une fois copié , lancer DosBox (raccourci qui se trouve sur le bureau)
ensuite tapez et exécutez:
`INSTALL.BAT`
Il installe et configure le Kernel par rapport à la configuration « de DosBox »
Attendez la fin de l'installation.
Une fois installé , fermez DosBox , ouvrez avec BlocNotes le fichier **C:\{*Users* ou *documents and setting*}\{*nom d'utilisateur*}\AppData\Local\DOSBox\dosbox-0.74.conf**
Allez tout en bas au paramètre **[autoexec] ,** puis à la dernière ligne , ajoutez

```
cd c:\Cpcdos\Systeme
```

Pour pouvoir travailler directement sur le Kernel dés l'ouverture de DosBox.
Ensuite vous pouvez lancer Dosbox ;)

Passez à la page suivante

Comment configurer & utiliser le Kernel ? :

Premièrement , le test :

Pour commencer , sur l'interpréteur de commandes ms-dos , allez dans le dossier
C:\Cpcdos\Systeme
tapez :

```
NOYAU.EXE sys/ /KRNLTEST #STOP#
```

« `#STOP#` » permettant de stopper le Kernel à la fin de l'exécution de la commande
Lors de ce teste , premièrement il va régler la priorité du système
Si tout fonctionne bien , vous devrez vous rendre compte que les commandes s'exécutent de plus en plus vite
Vous pourrez remarquer le temps que votre système met à exécuter (plus bas).

Il teste la rapidité du processeur , de la mémoire. Vous devrez entendre des « Beep » , ceci indique le nombres d'instructions calculées par boucles.

> Si vous avez le message « *** Tests terminées avec succès *** »
> **Tout est bon !**

Mais si votre système crash , c'est pas bon signe , redémarrez complètement votre machine et relancez le test.

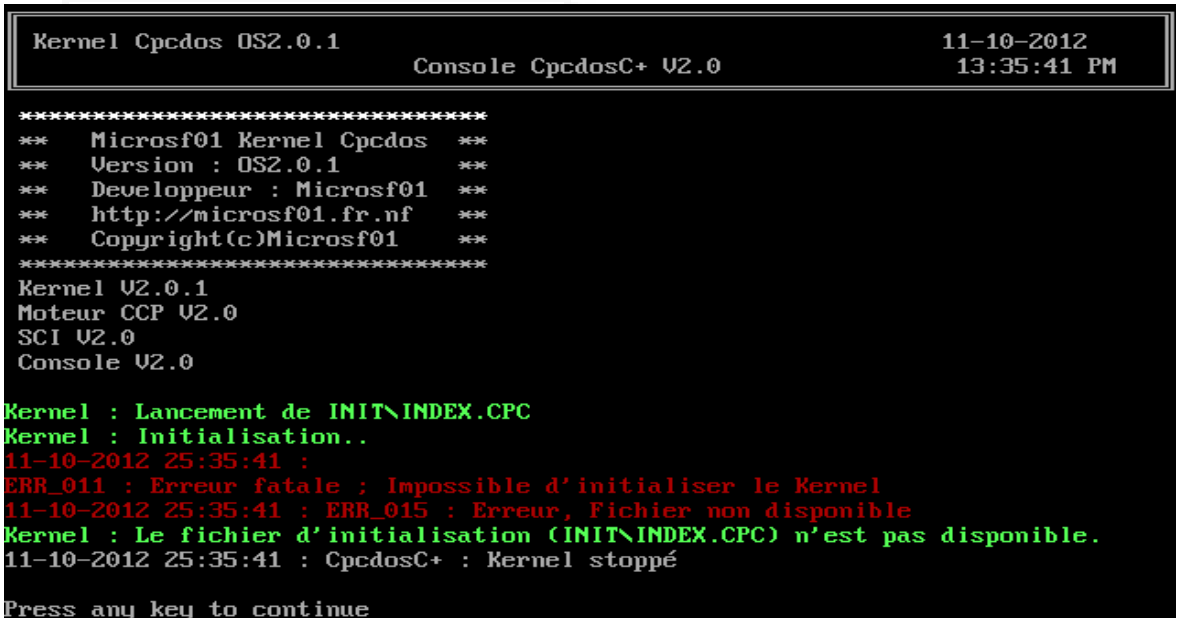
Si vous avez un message du genre « Out of stack space » , il faut configurer votre mémoire STACK (nous verrons ceci)

Si vous avez le message d'erreur « Far heap corrupt » , c'est un problème lors des tests des modes d'écran , cherchez pas , redémarrez votre machine !

Si ceci continue , si vous avez crée vous même le fichier test , enlever dans votre code , la commande « sys/ /testecr »
Puis redémarrez votre machine afin de commencer à zéro. Et allez à la page suivante.

C’est partis !

Une fois votre machine redémarré ,allez dans le répertoire **C :\Cpcdos\Systeme**
avec la commande **CD C:\Cpcdos\Systeme** , puis
tapez seulement la commande : **NOYAU.EXE**
Vous devrez avoir ceci :



Vous avez le message d’erreur 011 et 015 indiquant qu’il vous manque le fichier d’initialisation INDEX.CPC
Et nous allons le créer nous même !

Si vous ne voulez pas faire cette étape , copiez simplement le contenu du répertoire **C:\CPCDOS\SYSTEME\AUTRES** (les dossiers **INIT** et **OS**) dans le répertoire **C:\CPCDOS\SYSTEME**
Comme ça , il y aura déjà la configuration de base du Kernel.

Alors pour le créer , si là vous utilisez DosBox , utiliser simplement le NotePad de Windows
Laissez le formatage ANSI , on laisse ça de coté..
Si vous n’êtes pas sous Windows , mais complètement sous DOS , tapez la commande :

```
NOYAU SHELL/ C:\Cpcdos\Pilotes\Dos\EDIT.COM #STOP#
```

La commande SHELL permet d’exécuter une commande DOS ou d’exécuter un fichier EXE , COM ou BAT , et l’option **#STOP#** Permet simplement de ne pas exécuter le Kernel une fois la commande exécuté

Une fois uns de vos éditeurs ouvert , créez un nouveau fichier qui sera nommé **INDEX.CPC** dans le répertoire **C:\Cpcdos\Systeme\INIT**

A quoi sert ce fichier ?

Ce fichier , est un des premiers fichiers à être exécuté lors du lancement du Kernel
Il permet de lancer les zones de paramètres et ainsi lancer le fichier INDEX de l'OS
Bref , vous allez le découvrir au fur et à mesure.

Vous pouvez commencer à personnaliser votre fichier avec la commande **REM/**
Cette commande permet d'insérer une remarque personnalisé
vous pouvez mettre votre nom , date , titre.. ce que vous voulez
exemple :

```
rem/ Fichier d'indexage créer par Jean Pierre
rem/ Date : 11/11/2012
```

Tapez ceci , explications après :

```
:_KRNL_CFG:
exe/ INIT\KRNL_CFG.CPC /l:_CFG

:_KRNL_INI:
fix/ STOP_ERREUR = 1
exe/ INIT\KRNL_INI.CPC/l:INI_1

:OS :
fix/ STOP_ERREUR = 0
exe/ OS\INDEX.CPC /l:autoexec
```

Explications :

« :_KRNL_CFG: , :_KRNL_INI: , :OS: »
Ce sont des Labels , ils permettent de définir le début d'une zone exécutable.

Par exemple , vous pourrez exécuter le fichier **INDEX.CPC** à partir de **_KRNL_INI**

« exe/ INIT\KRNL_CFG.CPC /l:_CFG »
Permet d'exécuter le fichier **KRNL_CFG.CPC** à partir du label **_CFG**
(**INIT** , étant le dossier)

« fix/ STOP_ERREUR = 1 »
Permet de définir la variable **STOP_ERREUR** à 1
Cette variable à 1 permet de stopper le Kernel à la moindre erreur.

« exe/ INIT\KRNL_INI.CPC/l:INI_1 »
Permet d'exécuter le fichier **KRNL_INI.CPC** à partir du label **_INI_1**
(**INIT** , étant le dossier)

« fix/ STOP_ERREUR = 0 »
Permet de définir la variable **STOP_ERREUR** à 0
Cette variable à 0 permet de ne pas stopper le Kernel après une erreur.

« exe/ OS\INDEX.CPC /l:autoexec »
Permet d'exécuter le fichier **INDEX.CPC** à partir du label **autoexec**
(**OS** , étant le dossier)

Voilà le fichier INDEX.CPC du dossier INIT crée !
Pas la pène de tester , il ne fonctionnera pas car il manque des fichier que nous n'avons pas encore crées

Ensuite vous créez un nouveau fichier qui se nomme **KRNL_CFG.CPC**
Dans le même répertoire (**C:\Cpcdos\Systeme\INIT**)

Tapez ceci :

```
rem/ Microsf01 Cpcdos OS2
rem/ config

:_CFG:
fix/ tmp = "512"
fix/ ah = 1
fix/ PDS = 0

fix/ scr_bit = 0
fix/ scr_bas = NUL

:_CLOCK:
txt/ *** Definition Priorite systeme 100/100 ***
fix/ /io CLK = 100
exe/ INIT\INDEX.CPC/l:_KRNL_INI
```

Ces commandes suivant :

« `fix/ tmp = "512"`
`fix/ ah = 1`
`fix/ PDS = 0 »`

Ne servent a rien pour le moment , pour une version prochaine du Kernel..

« `fix/ scr_bit = 0 »`
Optionnelle , permet juste d'indiquer le Bit de couleur à 0 bit tant que le mode IUG n'est pas lancé.

« `fix/ scr_bas = NUL »`
Optionnelle aussi , juste indiquer , pas de changement de résolution d'écran tant que le mode IUG n'est pas lancé.

« `txt/ *** Definition Priorite systeme 100/100 *** »`
Permet d'afficher un message à l'écran.

« `fix/ /io CLK = 100 »`
Permet de fixer la variable **CLK** à 100
le paramètre /IO permet d'enregistrer cette variable sur le disque dur
Cette variable permet d'indiquer l'utilisation en Pourcentage du processeur.

« `exe/ INIT\INDEX.CPC/l:_KRNL_INI »`
Permet d'exécuter le fichier **INDEX.CPC** à partir du label **_KRNL_INI**
(**INIT**, étant le dossier)

Que une fois cette commande exécuté , il va lire le fichier **INDEX.CPC** à partir de **_KRNL_INI**

Chapitre IV - Commandes de bases

Allé , commençons par la programmation !

Cette partie IV sera les commandes de base que l'on utilise principalement en mode LC

J'ai ce dédié ce manuel de ce langage uniquement pour mon noyau Cpcdos OS2 , donc si vous voulez être développeur Cpcdos , vous pouvez apprendre a l'utiliser ici.

Pour commencer les commandes de bases , comme tout (enfin presque)

NB (Important) :

- *Pas de différenciation entre les commandes en majuscules/minuscules*
- *Autant de tabulation ou espace entre le début de ligne et la commande*
- *Utilisez des nom de propriétés différents*
- *Les valeurs de couleurs R.V.B doivent être de 3 caractères numériques*

  ex : Ne pas mettre « **1,50,30** » . *mais il faut mettre* « **001,050,030** » *(rajouter des zéros au début pour faire 3 caractères de chaques*
- *LC : Lignes de Commandes*
- *IUG : Interface Utilisateur Graphique*
- *CCB : Code Compilé Binaire*
- *CNB : Code Non Binaire*

EFFACER L'ECRAN

La commande qui le permet est :

```
cls/
```

(Clear Screen)

Commande compatible CCB et CNB

cette commande permet d'effacer l'écran uniquement quand le Kernel Cpcdos est en mode LC

cet a dire qu'il ne peux pas être utilisé lorsque le mode IUG est exécute !

Par contre en mode IUG , il existe un paramètre qui permet d'effacer complètement l'écran
(fonctionne aussi en mode LC)

(a utiliser avec précaution puisque j'ai fait de sorte que la commande n'alerte pas le service d'affichage ,
sinon le noyau redessinera l'interface ! mdr)

La commande qui permet l'effacement complet en mode IUG et aussi en LC , est :

```
cls/ tout
```

Exemple :

```
txt/ "hello word ! "
cls/
```

COMMENTAIRES

La commande qui permet ceci est :

```
rem/
```

Cette commande permet tout simplement au développeur d'écrire des commentaires dans son code pour se repérer etc ...

AFFICHAGE TEXTE

La commande qui permet ceci est :

```
txt/ {texte}
```

(**T**e**XT**e)

Cette commande est utilisable uniquement en mode LC , il permet l'affichage de caractères ASCII de codage DOS dans une plage de caractères allant de 0 à 255

il peux être composé de paramètres pour préciser les couleurs et positions

par exemple :

Écriture basique :

```
txt/ Hello word
```

Écriture sur une ligne en plusieurs ligne :

```
txt/ Coucou #R
txt/ Tu va bien ?
```

Sortie :

Coucou Tu va bien ?

Afficher le contenue d'une variable :

```
fix/ VAR1 = utilisateur
fix/ VAR2 = de l'ordinateur
txt/ Coucou %VAR1% %VAR2 %
```

Sortie :

Coucou utilisateur de l'ordinateur

Des couleurs ?

```
Couleurf/ 2
couleurp/ 1
txt/ Hello word !
```

Affiche Hello Word ! En bleu (1) sur vert (2)

CREER UNE VARIABLE

La commande qui le permet est :

```
fix/ {variable} = {Données}
```

(**Fix**er (Set))

Cette commande permet de créer une variable locale , (limité à 128)

Il contient une fonctionnalité avancé d'enregistrement de variables en mémoire.

```
Fix/ /io {variable} = {Données}
```

Mais la chaîne de nom de variable est limité à 8 caractères.

Il permet l'enregistrement de la variable dans sur le disque.
Ce qui permet de pouvoir redémarrer le pc et pouvoir relire un variable

Puis

```
Fix/ /supp {variable}
```

Qui lui permet de supprimer la variable enregistré sur le disque

Si cette option est omise , la chaîne peux aller jusqu'à 24 caractères.

EXECUTER UN FICHIER CPCDOSC+

La commande qui le permet est :

```
exe/ {fichier}
```

(**exe**cuter)

Cette commande permet d 'exécuter un fichier de commandes CpcdosC+ uniquement

Le paramètre « /l: »

```
exe/ {fichier} /l:
```

Ce paramètre permet d'indiquer au Kernel a partir de quelle label le code s'exécute

Exemple :

```
exe/ Fichier.cpc /l:monlabel
```

Fichier.cpc :

```
txt/ texte1
monlabel:
txt/ texte2
```

Sortie :

texte2

- - - - - - - -

Il existe 2 type de fichier exécutables CpcdosC+

de fichiers CCB (Code Compilé Binaire) et des fichiers CNB (Code non Binaire)

L'avantage du CCB c'est que le code est crypté , in-modifiable l'exécution sera certes un peux plus long mais il aura le même résultat que le CNB

Puis l'avantage du CNB c'est que le fichier est complètement modifiable
mais inconvenant c'est qu'il ne peux pas être protégé.

Pas de paramètres particuliers pour l'exécution d'un CCB , il se décrypte et se lance tout seul..

ARRÊTER L'EXECUTION D'UN FICHIER CPCDOSC+

La commande qui le permet est :

```
stop/
```

Cette commande permet d'arrêter l'exécution d'un fichiers de commandes CpcdosC+ en cours

Il devrait être utilisé dans un fichiers.

Une autre commande est semblable , ne pas confondre, il sert a stopper complètement l'exécution du Kernel sans prévenir le SCI et la procédure de vidage mémoire et arriver directement à l'interpréteur de commandes ms-dos ou FreeDos.

La commande qui le permet est :

```
stopk/
```

(**stopk**ernel)

CONDITIONS

La commande qui le permet est :

```
si {interrogé} { Condition = < > } {Opérateur} (: {Commande} :)
```

Cette commande est une instruction conditionnelle.

Exemples :

```
fix/ VAR1 = 5
fix/ VAR2 = 2
fix/ RES = /c %VAR1% + %VAR2%
si/ %RES% = 7 (: txt/ %VAR1% + %VAR2% est egale a 7 ! :)
```

EXECUTER UN FICHIER EXECUTABLE DOS

La commande qui le permet est :

```
shell/ {fichier}
```

Cette commande exécute des fichier de format MZ (*.exe .com)* et interpreteur DOS *.BAT*

Exemple :

```
shell/
```

Une commande voisine existe et remplie la même fonction :

```
dos/
```

Chapitre V - Commandes avancées

Nous allons voir ici les commandes plus approfondies , cet à dire des commandes pour contrôler & créer une interface graphique.

BASE :

Déjà , vous devez indiquer les principaux paramètres qui permet au Kernel d'interagir , de paramétrer l'interface etc...

```
rem/ Configurer le systeme
fix/ SYS_SEG = segment16
fix/ SYS_MEMTYPE = xms
fix/ SYS_COMC = 512
fix/ SYS_STACK = 4096
rem/ Carte graphique :
fix/ SCR_bas = 1024x768
fix/ SCR_bit = 16

rem/ Fond d'ecran bureau
fix/ SCR_FOND = IMAGE.BMP

rem/ Repertoires systemes
fix/ Prog = OS\Prog
fix/ Media = OS\Media
fix/ Systeme = OS\Cpcdos
```

INITIALISER UNE INTERFACE

La commande qui le permet est :

```
ini/
```

Et

```
ini;
```

Ces commandes permettent la création d'un tableau pour la création de fenêtres ou d'objets

Il faut bien sûre compléter la création avec la commande ***créer/***

ini/ est une boucle d'initialisation

Pour créer une fenêtre :

```
ini/ fenetre(
ini/ fenetre)
```

Pour créer un bouton :

```
ini/ bouton(
ini/ bouton)
```

Pour créer un label :

```
ini/ label(
ini/ label)
```

Pour créer une imagebox :

```
ini/ imagebox(
ini/ imagebox)
```

Pour créer un textebox :

```
ini/ textebox(
ini/ textebox)
```

Pour créer un :

```
ini/ (
ini/ )
```

Pour créer un :

```
ini/ (
ini/
```

Puis la commande ***ini;*** doit être utilisé uniquement dans une boucle d' INItialisation (vu ci-dessu)

il permet de remplir des valeurs dans un tableau mémoire du Kernel permettant la création d'une interface

```
ini;nom = "NOM"
ini;texte = "TEXTE
ini;type = "0 ou 1"
ini;couleur = "Rouge,Vert,Bleu"  ‘ Couleur base (genre fenêtre)
ini;couleurf = "Rouge,Vert,Bleu"  ‘ Couleur de Fond
ini;couleurp = "Rouge,Vert,Bleu"  ‘ Couleur Premier Plan
ini;tx = "Taille X"
ini;ty = "Taile Y"
ini;px = "Position X"
ini;py = "Position Y"

etc...
etc...
```

Nous allons dans les pages suivantes voir des exemples.

FERMER UN OU PLUSIEURS OBJETS/FENETRES

La commande qui le permet est :

```
fermer/
```

Cette commande permet de fermer (décharger) en mémoire toutes les propriétés associé au tableau qui permettant l'affichage graphique & interagir des événements des données de propriété.

Il cherche et ferme TOUTES propriétés au nom correspondant , tout en respectant la classe.

Par exemple , si un bouton , un label puis une fenêtre ont le même nom , ils seront tous les trois fermées.

Nb : nom en majuscules !

Exemples :

```
fermer/ MONBOUTON
fermer/ FENETRE_1
fermer/ A1
```

Il existe aussi un paramètre « drôle » , (a utiliser avec précautions) , qui permet de fermer TOUS les objets.

La commande qui le permet est :

```
fermer/ /tout
```

Si elle est exécute , vous n'aurez plus de bouton , de label , de textebox etc...

ACTUALISER UNE OU PLUSIEURS FENETRES

La commande qui le permet est :

```
actualise/
```

Cette commande permet d'actualiser une fenêtre

Exemple :

```
actualise/ MA_FENETRE_1
```

Le paramètre **/tout** permet d'actualiser toutes les fenêtres

Exemple :

```
actualise/ /tout
```

FOCUS SUR UNE FENETRE

La commande qui le permet est :

```
focus/
```

Cette commande permet de sélectionner une fenêtre et la dessiner au premier plan.

Exemple :

```
focus/ MA_FENETRE_1
```

CREER UNE FENETRE

La commande qui le permet est :

```
ini/ fenetre( )
```

Cette commande , accompagné de paramètres obligatoires , permet de créer une fenêtre graphiquement parlant de ce type (Exemple):

Barre de titre , fond de couleur , position , taille , type , déplaçable , refermable.. et un contenu

(pas de boutons femeture , j'ai pas finis)

Exemple :

```
ini/ fenetre(
    ini;nom = "FENETRE_1"
    ini;texte = "Ma petite fenetre !"
    ini;type = "1"
    ini;couleur = "087,215,186"
    ini;tx = "300"
    ini;ty = "250"
    ini;px = "MX"
    ini;py = "MY"
    Creer/
ini/ fenetre)
```

Explications !

```
ini/ fenetre(
```

Permet d'informer pour INItialiser / créer une nouvelle fenêtre

```
ini;nom = "FENETRE_1"
```

Donner au noyau le nom d'Objet de la fenêtre

Si vous nommez une autre fenêtre au mémé nom , la fenêtre sera remplacé

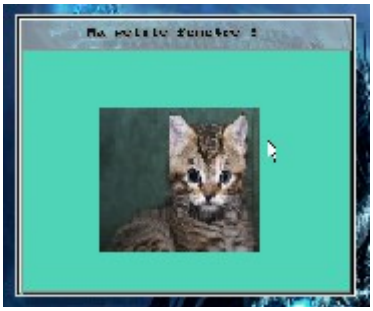
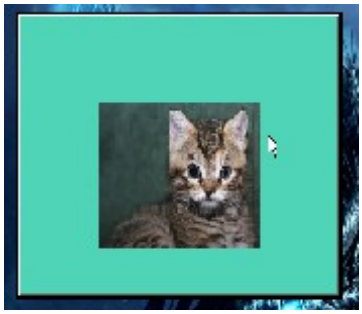
```
ini;texte = "Ma petite fenetre !"
```

Titre de la fenêtre

```
ini;type = "1"
```

Type de la fenêtre

1:Normal 2:Sans Barre de titre 3:transparente


Il y a aussi d'autres paramètres attribuables comme

Movable? (Déplaçable)

Ative? (Interaction possible)

Reductable?(Reduire)

Fermable?

Tache?(Affiché dans la barre des tâches)

```
ini;type = "1;M0A0R0F0T0"
```

La lettre + une valeur booleen (1 ou 0) si un(s) des paramètres sont omit il est activé par défaut

```
ini;couleur = "000,000,000"
```

Couleur de fond de la fenêtre en R.V.B (bien mettre 3 caractères sur chaque couleurs) /!\

```
ini;tx = "300"
```

Taille X de la fenêtre
Acronyme : TX = Taille X

```
ini;ty = "250"
```

Taille Y de la fenêtre
Acronyme : TY = Taille Y

```
ini;px = "MX"
```

Position X de la fenêtre
Acronyme : PX = Position X
"MX"
C'est juste pour que la fenêtre sois centré sur l'axe X ,vous pouvez mettre une valeur à la place
(Milieu X)

```
ini;py = "MY"
```

Position X de la fenêtre
Acronyme : PY = Position Y
"MY"
C'est juste pour que la fenêtre sois centré sur l'axe Y, vous pouvez mettre une valeur à la place
(Milieu Y)

```
creer/
```

Permettant de créer l'objet ou la fenêtre qui se trouve en mémoire dans le tableau SCI
(Attention , bien le mettre avant la prochaine création d'objet ou de fenêtre , car sinon , les valeurs risquerons d'être remplacé par les nouvelles.)
Il faut obligatoirement l'écrire juste avant la fin de boucle **`ini/ fenetre)`** sinon rien ne se passera.

```
ini/ fenetre)
```

Indique la fin de l'INItialisation de la fenêtre.

CREER UN BOUTON

La commande qui le permet est :

```
ini/ bouton( )
```

Cette commande , accompagné de paramètres obligatoires permet de créer un bouton graphiquement parlant de ce type (Exemple):

Voici un exemple : (il faut bien sûre créer une fenêtre –> voir **CREER UNE FENETRE** ci dessus).

```
ini/ bouton(
    ini;nom = "BOUTON_1"
    ini;fenetre = "FENETRE_2"
    ini;texte = "Mode TECK ;D"
    ini;type = "1"
    ini;img = "3"
    ini;couleurf = "100,100,250"
    ini;couleurp = "200,000,255"
    ini;tx = "150"
    ini;ty = "30"
    ini;px = "10"
    ini;py = "170"
    creer/
ini/ bouton)
```

Explications !

```
ini/ bouton(
```

Permet d'informer au Kernel , la création d'un objet (Bouton).

```
ini;nom = "BOUTON_1"
```

Permet de nommer la propriété.

```
ini;fenetre = "FENETRE_1"
```

Indique au Kernel , sur quelle fenêtre allons-nous créer ce bouton.
Là , il va créer ce bouton sur **FENETRE_1.**

```
ini;texte = "Mon Bouton"
```

Permet d'écrire le texte du bouton.

```
ini;type = "1"
```

(Pas d'effets sur la version Cpcdos OS2.0.1 , Laissez sur **"1"**)

```
ini;img = "3"
```

Permet de définir une image de fond sur un bouton

numéro entre 0-7

vous avez par défaut :

ini;img = "1"

ini;img = "2"

ini;img = "3"

ini;img = "4"

ini;img = "5"

ini;img = "6"

ini;img = "7"

Les images sont stocké dans la cible où vous avez définit la variable MEDIA dans OS.CPC

Par défaut , il se situe dans « **C:\CPCDOS\SYSTEME\OS\Media\IUG** »

Souces images BMP personnalisables ;)

```
ini;couleurf = "100,100,250"
```

Permet de définir une couleur en R , V , B de fond du bouton
Uniquement si l'option `ini;img = "0"`

```
ini;couleurp = "200,000,255"
```

Permet de définir la couleur des caractères du texte du bouton

```
ini;tx = "150"
```

Défini la taille de l'axe X du bouton

```
ini;ty = "30"
```

Défini la taille de l'axe Y du bouton
(par défaut&conseillé ,mettez la valeur à 30)

```
ini;px = "10"
```

Défini la position de l'axe X sur la fenêtre définit (valeur négatifs autorisés)

```
ini;py = "170"
```

Défini la position de l'axe Y sur la fenêtre définit (valeur négatifs autorisés)

```
creer/
```

Permet de créer l'interface avec les valeurs que vous avez fournis ci-dessus

```
ini/ bouton)
```

Ferme la boucle..

CREER UN LABEL

La commande qui le permet est :

```
ini/ label( )
```

Cette commande , accompagné de paramètres obligatoires , permet de créer un label graphiquement parlant de ce type (Exemple):

Voici un exemple : (il faut bien sûre créer une fenêtre –> voir **CREER UNE FENETRE** ci dessus).

```
ini/ label(
    ini;fenetre = "FENETRE_1"
    ini;nom = "LABEL_1"
    ini;texte = "Il est bien hein ? :)"
    ini;couleurf = "250,010,010"
    ini;couleurp = "000,255,100"
    ini;transparent = "0"
    ini;type = "0"
    ini;tx = "200"
    ini;ty = "20"
    ini;px = "10"
    ini;py = "15"
    Creer/
```

Explications !

```
ini/ label)ini/ label(
```

Permet d'informer au Kernel , la création d'un objet (Label).

```
ini;fenetre = "FENETRE_1"
```

Indique au Kernel , sur quelle fenêtre allons-nous créer ce bouton.
Là , il va créer ce bouton sur **FENETRE_1.**

```
ini;nom = "LABEL_2"
```

Permet de nommer la propriété. (l'objet)

```
ini;texte = "Il est bien hein ? :)"
```

Permet de définir du texte graphiquement dans le label.

```
ini;couleurf = "250,010,010"
```

Définir la couleur de fond si **ini;transparent = "0"**

```
ini;couleurp = "000,255,100"
```

Définir la couleur des caractères.

```
ini;transparent = "0"
```

Permet d'avoir les couleur d'arrières plan « entre les caractères »
la transparence arrière plan.

```
ini;type = "0"
```

Permet de définir si la taille du label est réglé automatiquement par rapport a son contenue **"0"** donc les paramètres **ini;TX** et **ini;TY** sont inutiles

ou réglé manuellement par **ini;TX** et **ini;TY.**

```
ini;tx = "200"
```

Permet de définir la taille sur l'axe X uniquement si **ini;type = "0"**

```
ini;ty = "20"
```

Permet de définir la taille sur l'axe Y uniquement si **ini;type = "0"**

```
ini;px = "10"
```

Permet de positionner sur l'axe X le label dans la fenêtre

```
ini;py = "15"
```

Permet de positionner sur l'axe Y le label dans la fenêtre

```
Creer/
```

Permet de créer l'interface avec les valeurs que vous avez fournis ci-dessus

```
ini/ label)
```

Ferme la boucle..

CREER UNE IMAGEBOX

La commande qui le permet est :

```
ini/ imagebox( )
```

Cette commande , accompagné de paramètres obligatoires , permet de créer un label graphiquement parlant de ce type (Exemple):

CREER UNE INTERFACE

La commande qui le permet est :

```
creer/
```

Cette commande permet de créer une fenêtre ou un objet dans la boucle de création d'interface à partir de valeurs complétées (d'un tableau)

Exemple d'utilisation :

```
ini/ fenetre(
   ini;nom = "FENETRE_1"
   ini;texte = "Ma première fenêtre Cpcdos !"
   ini;type = "1"
   ini;couleur = "255,255,255"
   ini;tx = "300"
   ini;ty = "250"
   ini;px = "200"
   ini;py = "150"
   creer/
ini/ fenetre)
```

Si vous ne le mettez pas avant la commande **`ini/ fenetre)`** le tableau de valeurs sera vidé Toujours dans la boucle d'initialisation. !

DEMARRER L'INITIALISATION L'OS

La commande qui le permet est :

```
demarrer/
```

Cette commande permet tout simplement de démarrer le fichier qui se trouve dans la variable %BOOTOS% qui est normalement dans le chemin « OS\INDEX.CPC »

Donc en gros, il permet juste d'exécuter le fichier d'index qui permet l'exécution de l'OS.

Paramètres disponibles :

```
demarrer/ /safe
```

Permet de charger l'OS avec des limitations de certains paramètres si l'OS ne démarre pas correctement..

Aucuns pont ou relais se fait avec la commande **IUG/** puisse que dans ce mode le Kernel arrêtera le chargement quand **IUG/** sera déceler.

LANCER L'INTERFACE GRAPHIQUE (OS)

La commande qui le permet est :

```
iug/
```

Cette commande permet d'exécuter le service IUG qui lui gère le fond d'écran, les fenêtre et objets puis l'interaction utilisateur (evènements)

Paramètres disponibles :

```
iug/ /reset
```

Peut être tapé à la console avant ou même après l'exécution de l'OS

Permet tout simplement de recharger le code/fichier qui se trouve dans la partie IUG dans l'index principal. (après le label «:IUG : » dans le fichier INDEX.CPC

```
iug/ /safe
```

Permet d'exécuter l'interface graphique de l'OS avec des limitations si l'OS ne démarre pas correctement ou des bus se produit

Il limite l'affichage à 800x600x16

Chapitre VI - Exemples :

LC (Lignes Commandes) :

IUG (Interface Utilisateur Graphique) :

Pour le premier exemple , nous allons coder notre première fenêtre + 1 bouton

Tout ce qui est après « = » Devra se trouver entre les guillemets « **"" »**

Pour commencer , le code , puis les explications après ,