# Projet : développement d'un mini système d'exploitation pour PC x86

Jérôme Ermont et Emmanuel Chaput

IRIT - Toulouse INP/ENSEEIHT

# Plan de la présentation

# Objectifs

Développer les éléments de base d'un système d'exploitation

## Ce que nous verrons

- Gestion d'entrées/sorties de base : le clavier et l'écran
- Gestion des interruptions
- Gestion des processus
- Gestion de la mémoire virtuelle pour les processus

## Ce que nous ne verrons pas

- Gestion des fichiers
- Partage de ressources et communication entre processus

# Organisation

- 11 séances encadrées
- Travail en binôme ? → non !
- Pas de cours
- Programmation en langage C
- Quelques petits bouts en assembleur (code et/ou explications fournis)
- Évaluation : code commenté du projet
- Lien du serveur Discord : `https://discord.gg/FyESdWE`

# Plan de la présentation

# Au menu

Entrées

- De l'affichage à la console
- S'il vous plaît ? Je peux vous interrompre ?
- Il y a des manières, monsieur ! Utilisez l'appel système !

- Mise en œuvre de la console
- Appel système write

juste un petit test !

# Au menu

## Le plat

- Tic Tac Tic Tac, respectez le Timer !
- Des processus ? Comment tu définis ça ?
- Alors toi, tu crées des processus et tu les détruis.
- Il faudrait organiser tout ce beau monde, non ?
  - Et hop ! Tout le monde en file !
  - Laissez un peu la place aux autres ! Revenez dans la file ! Respectez le tourniquet !

- Interruption Timer
- Ordonnancement et gestions des processus

# Au menu

Le dessert

- C'est bien fichu ici : tu peux commander depuis la table avec un clavier.

- Lecture au clavier et appel système read
- Interpréteur de commandes simple

# Avec ta fourchette !

Il nous faut des couverts !

- Compilation : GCC
  - `sudo apt-get install build-essentials`
- Exécution : QEMU
  - `sudo apt-get install qemu`
- Mise au point : GDB
  - `sudo apt-get install gdb`
  - GDB sera connecté à QEMU et permet d'afficher les problèmes potentiels

**Remarque**

Les commandes données sont pour environnement Debian et dérivés (j'utilise Ubuntu). A adapter en fonction de votre distribution Linux.
Sous Mac : Utiliser les outils `gcc` pour x86 disponible via les `macports` (paquet `i386-elf-gcc`).

# Plan de la présentation

# L'archive fourni contient

- `/boot`
  - répertoire d'entrée du système ;
  - `crt0.S` initialise le matériel et lance le programme principal du système (`kernel_start`)
- `/kernel`
  - répertoire sources du noyau
  - c'est ici que tout (ou presque) va se passer
- `/lib`
  - quelques outils utiles (par ex. : `printf`)
- `/include`
  - pour les `.h` c'est ici

# Prise en main de l'environnement

- La compilation s'effectue via la commande `make`
  - si tout va bien, résultat : `kernel.bin`
- Exécution : `make run`
  - Une fenêtre QEMU doit apparaître
  - Le système est exécuté
- Mise au point
  - `make dbg-qemu`
  - Connecter `gdb` et `qemu` : dans gdb, taper `target remote :1234`
  - Lancer l'exécution : `r` ou `cont`
  - Point d'arrêt : `b kernel_start`
  - Afficher une variable : `display`
  - `n` : Next, `s` : Step

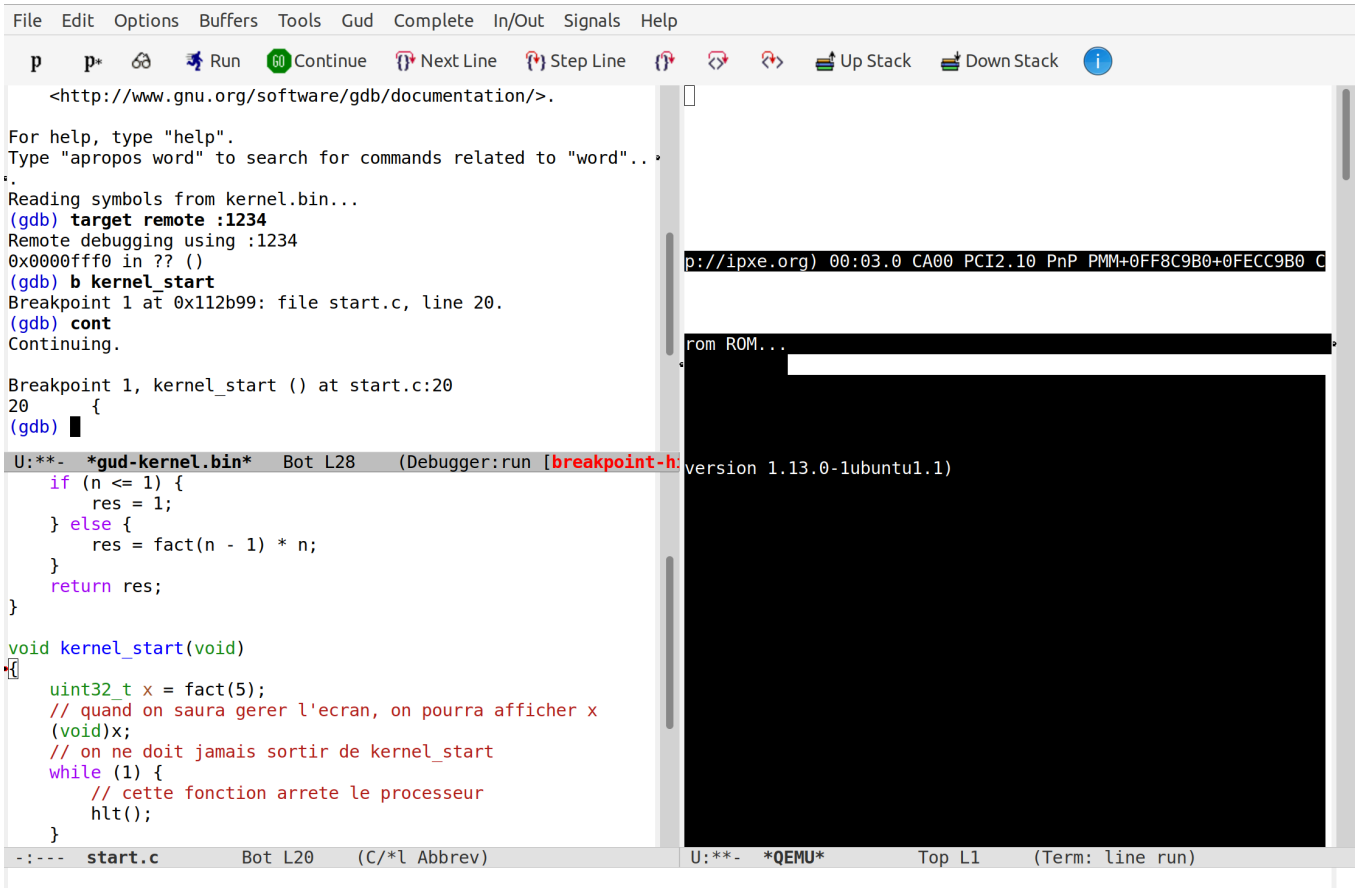
# Prise en main de l'environnement

## Un premier exercice simple

1. Ecrire une fonction qui retourne le résultat de la suite de Fibonnacci pour n=5.
2. Appeler cette fonction dans la fonction `start`.
3. Exécuter et afficher l'exécution dans `gdb`.

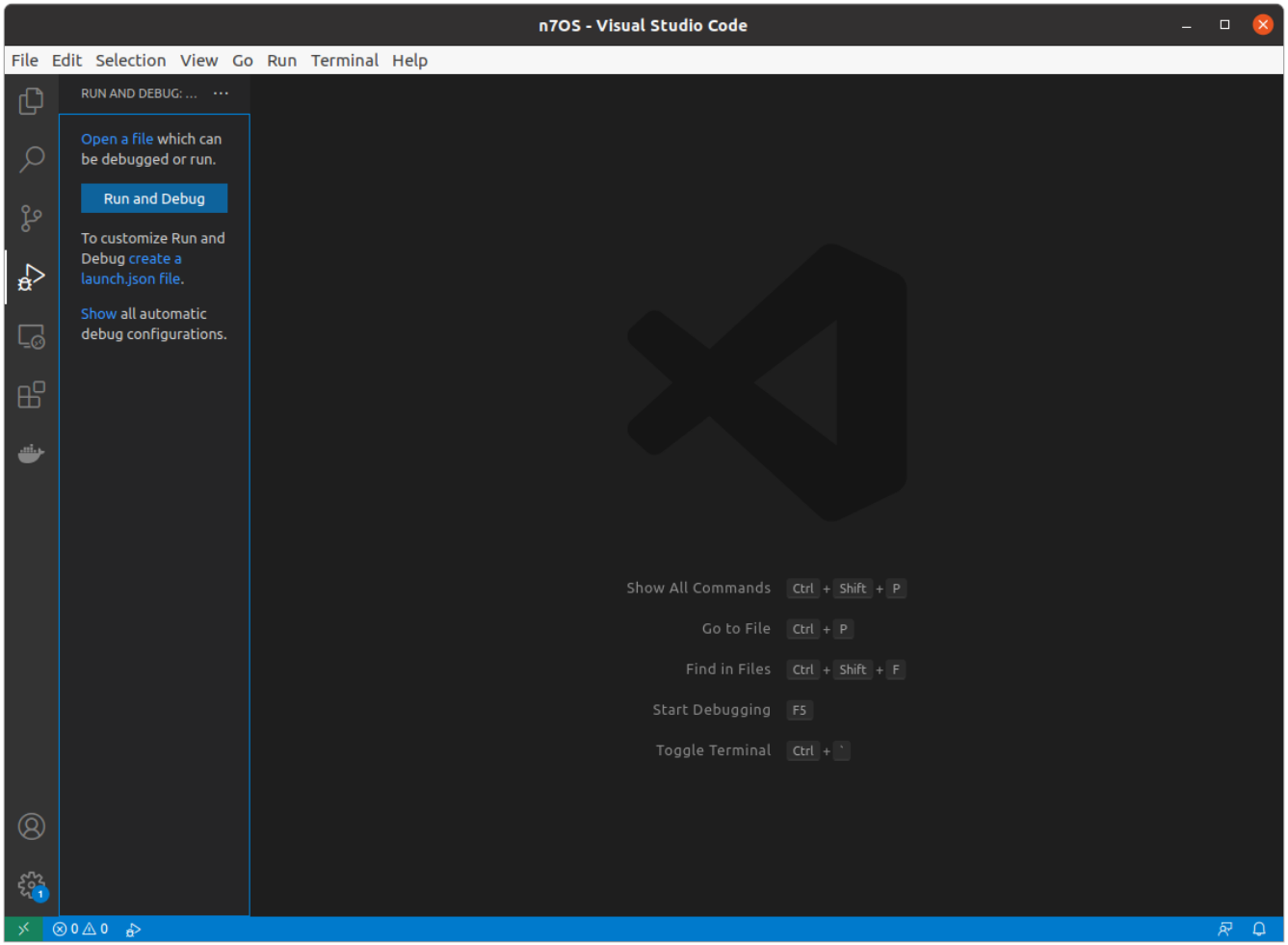
# Une version plus sympa

- Une version qui utilise `emacs`
- `make dbg` (Pour quitter CTRL-X-C)



# Pour les utilisateurs de Visual Studio Code

- Visual Studio Code dispose d'un debugger intégré



# Pour les utilisateurs de Visual Studio Code

Pour cela il faut :

1. Ajouter l'extension `Native Debug`
2. Créer un fichier `launch.json` (menu debug)
3. Configurer `launch.json` avec :

```
{
    "type": "gdb",
    "request": "attach",
    "name": "Attach to gdbserver",
    "executable": "${workspaceRoot}/kernel.bin",
    "target": ":1234",
    "remote": true,
    "cwd": "${workspaceRoot}",
    "debugger_args": ["-i=mi"],
    "valuesFormatting": "parseText",
    "autorun": [
        "dir kernel",
        "dir boot",
        "dir bin",
        "dir lib"
    ],
}
```

4. Exécuter `make vscode-dbg` puis lancer le debugger

# Plan de la présentation

# Et une console pour la une !

## Affichage à l'écran

- L'écran est composé de 25 lignes de 80 colonnes
  - Zone mémoire qui commence à l'adresse 0xB8000
  - Tableau de 25$\times$ 80 cases
- Chaque case est composée de 2 octets : le caractère à afficher et ses attributs (cli = clignotage, couleurs de fond et de texte)

| Format d'affichage | | | | | | | | Caractère | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| cli. | fond | | | texte | | | | code ascii | | | | | | | |

- Position à l'écran : 0xB8000 + (80 $\times$ ligne + colonne) $\times$ 2

# Et une console pour la une !

## Les couleurs

- 16 couleurs sont possibles

| Num. | Couleur | Num. | Couleur |
|---|---|---|---|
| 0 | noir | 8 | gris foncé |
| 1 | bleu | 9 | bleu clair |
| 2 | vert | 10 | vert clair |
| 3 | cyan | 11 | cyan clair |
| 4 | rouge | 12 | rouge clair |
| 5 | magenta | 13 | magenta clair |
| 6 | marron | 14 | jaune |
| 7 | gris | 15 | blanc |

# Et une console pour la une !

## Le curseur

- Le curseur est géré par la carte graphique
- Gestion à l'aide de 2 ports E/S : 0x3D4 (commande) et 0x3D5 (valeur)
- Utilisation des commandes `out` du processeur Intel
  - `void outb(uint8 val, uint8 port);` de `n70S/cpu.h`
  - pos du curseur : 80 $\times$ ligne + colonne
  - envoi de la position en 4 étapes
    1. envoi de la commande 15 (0xF)
    2. envoi de la valeur du poids faible de la position
    3. envoi de la commande 14 (0xE)
    4. envoi de la valeur du poids fort

# Et une console pour la une !

## La console

- `void console_putbytes(const char *s, int len);` : écrit la chaine de caractères à l'écran
  - utilisée par la fonction `printf` (pour le moment)
- seuls les caractères compris entre 32 et 127 seront affichés
- quelques caractères de contrôle à prendre en compte

| Code | Commande | Car. C | Action |
|---|---|---|---|
| 8 | Backspace (BS) | '\b' | Déplace le curseur d'une colonne en arrière |
| 9 | Horizontal Tab (HT) | '\t' | Ajoute un espace de 8 caractères |
| 10 | Line Feed (LF) | '\n' | Déplace le curseur à la ligne suivante, colone 0 |
| 12 | Form Feed(FF) | '\f' | Efface l'écran et revient à la colonne 0 de la ligne 0 |
| 13 | Carriage Return (CR) | '\r' | Déplace le curseur à la colonne 0 de la ligne courante |

# La préparation

## Travail à faire

1. Editer le fichier `console.c` et compléter le. Nous disposerons alors d'un affichage à l'écran.
2. Tester le programme en affichant le résultat de la fonction créée à l'exercice précédent.

## Remarque

Une description plus complète de la console est fournie sous Moodle

# Un peu d'aide ?


https://wiki.osdev.org/