Projet de système d'exploitation
— Rock and Roll —

Benjamin Bonneau & Maxime Flin

14 mai 2020

Sommaire :

Introduction

ecos est un système d'exploitation conçu pour une architecture Intel x86 64 bits. Il propose une interface très proche des normes POSIX et fourni

- une gestion de processus concurrent
- une gestion de plusieurs systèmes de fichiers
- une implémentation de la library standard C
- un shell avec des programmes utilitaires courants (cat, ls, etc. . .)

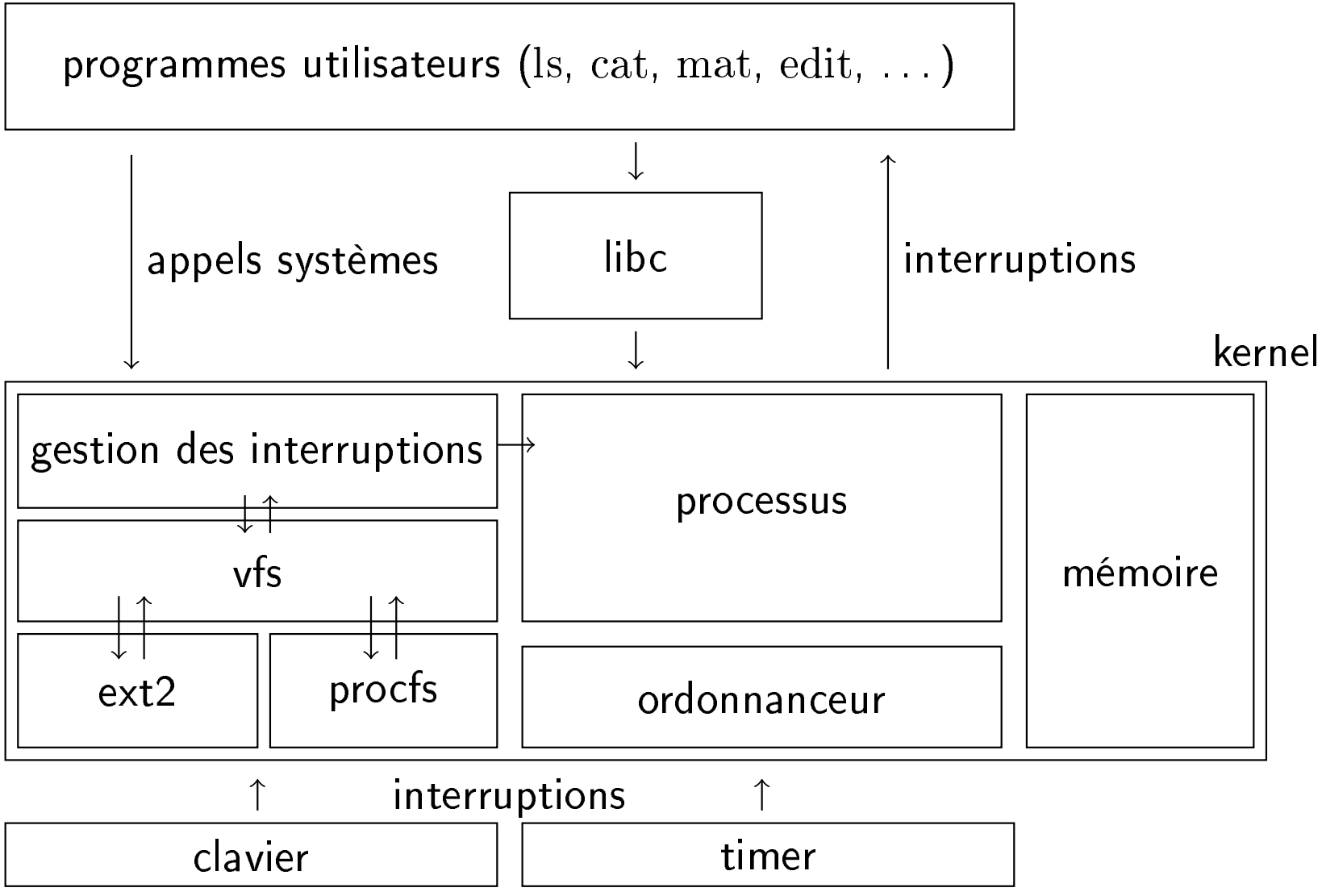

Figure – une vue globale du système

Boot

src/boot

- GRUB
  - kernel (ELF)
  - carte de la mémoire
  - affichage
- *protected mode 32 bits* vers *long mode 64 bits*
- Control Registers
- Paging
- Global Descriptor Table
- Chargement du kernel

Mémoire

Paging 4 niveaux : 48 bits d'adresse

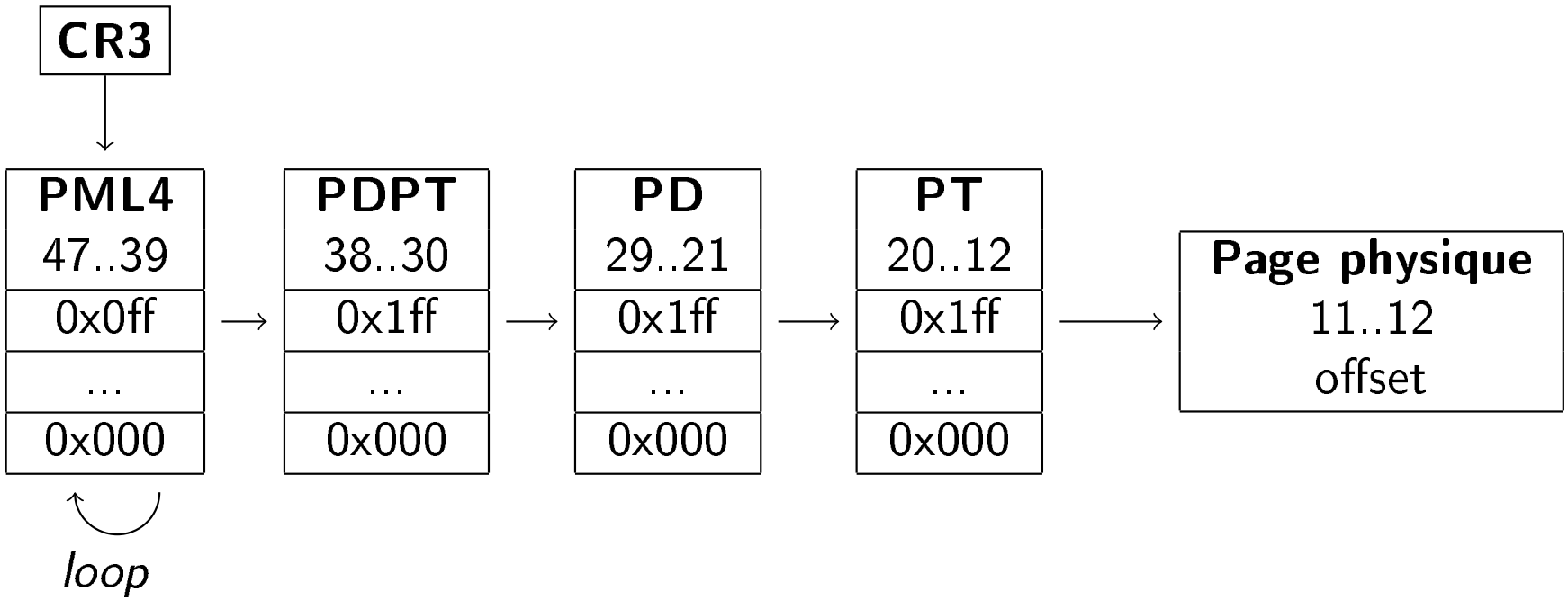

Figure – Structure du paging

Mémoire

Allocation des pages physiques disponibles

- découpage de la mémoire en blocs de 2Mo
- au niveau d'un bloc : arbre d'arité 8 (3 niveaux)
- pour gérer l'ensemble des blocs : 2 arbres d'arité 64. Blocs entièrement libres et blocs partiellement libres.

Mémoire

- chaque processus dispose de son PML4
- partie basse : userspace
- partie haute : kernel
- entrées spéciales dans la partie userspace :
  - Pages partagées (libc)
  - Pages allouées lors de l'accès (.bss, pile)
  - Pages copiées lors de l'accès (fork)

  actions effectuées lors de l'accès détecté par un page fault

Execve

- effectué par un processus auxiliaire avec privilèges de niveau 1
- gestion des fichiers spécifiant un interpréteur (#!)
- chargement de fichiers ELF
- création d'un nouveau paging
- allocation et chargement de sections depuis le fichier : .text, .data, .rodata
- marquage des pages à allouer lors de l'accès : .bss (initialisé à 0), pile
- ajout de la libc
- copie des arguments et paramètres d'environnement
- protection des pages en lecture seule

Processus

On maintient une tables de processus dans laquelle on sauvegarde pour chaque pid toutes les informations dont on pourrait avoir besoin :

- le statut du processus associé (Libre, Zombie, Endormi, Run, ...)
- l'état des registres lors de la dernière interruption
- l'identifiant du paging
- la table des descripteurs de fichiers

Pour éviter d'avoir à parcourir toute la table on a ajouté aux processus une structure de liste chainée.

Ordonnancement des processus

On a une table qui a un niveau de priorité associe une file de processus. L'ordonnanceur prend le premier processus de la file de priorité la plus élevée [1].
Le noyau reçoit toutes les 10ms des interruptions du PIT [2] qui interrompent le processus en cours. Si celui-ci travaille déjà depuis plus d'une certaine unité de temps [3], alors on laisse travailler un autre processus. De cette manière, on évite les famines.
On distingue deux processus particuliers :

- le processus `idle`, de pid 0, est un processus de priorité minimale ; c'est le processus courant lorsqu'il n'y a rien à faire. Lorsqu'un processus demande à travailler, si c'est `idle` le processus courant, alors on attends pas le tic de PIT avant de changer le processus.
- le processus `init`, de pid 1, est le premier processus lancé par le noyau au moment du boot. Il se fork en un `init1` (qui lance un shell) et attend de récupérer les processus zombies.

1. la recherche de la priorité est optimisé avec une bitmap
2. Programmable Interval Timer
3. définie dans kernel/params.h

Système de fichier

On n'interagit avec aucune device réelle, par manque de temps, nous avons décidé de simplement simuler les comportement d'un tel composant. Lors de la compilation, nous générons un fichier binaire qui représente le système de fichier et le chargeons directement dans le binaire du noyau.
Néanmoins nous avons la possibilité d'interagir avec plusieurs devices en même temps. Pour chacune de ces devices, on précise

- sa position dans le système de fichier parent (si ce dernier existe)
- le système de fichier à utiliser pour lire dans cette device
- des informations spécifique au point de montage.

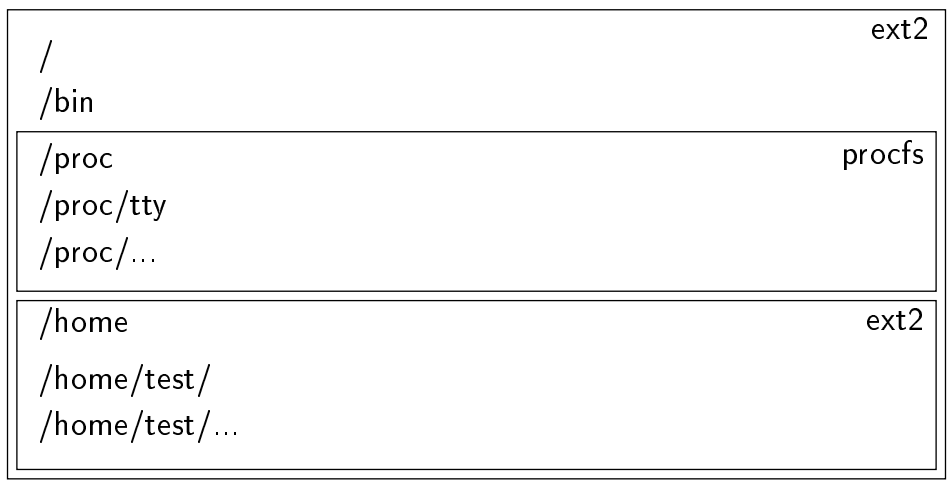

Figure – Hiérarchie des systèmes de fichiers dans notre projet

Extended Filesystem 2

Un système de fichier développé dans les années 90 pour remplacer Minix, le système de fichier supporté par les premières versions de Linux. Les données sont réparties groupes de blocks pour essayer de maintenir les blocs d'un même fichiers adjacent.

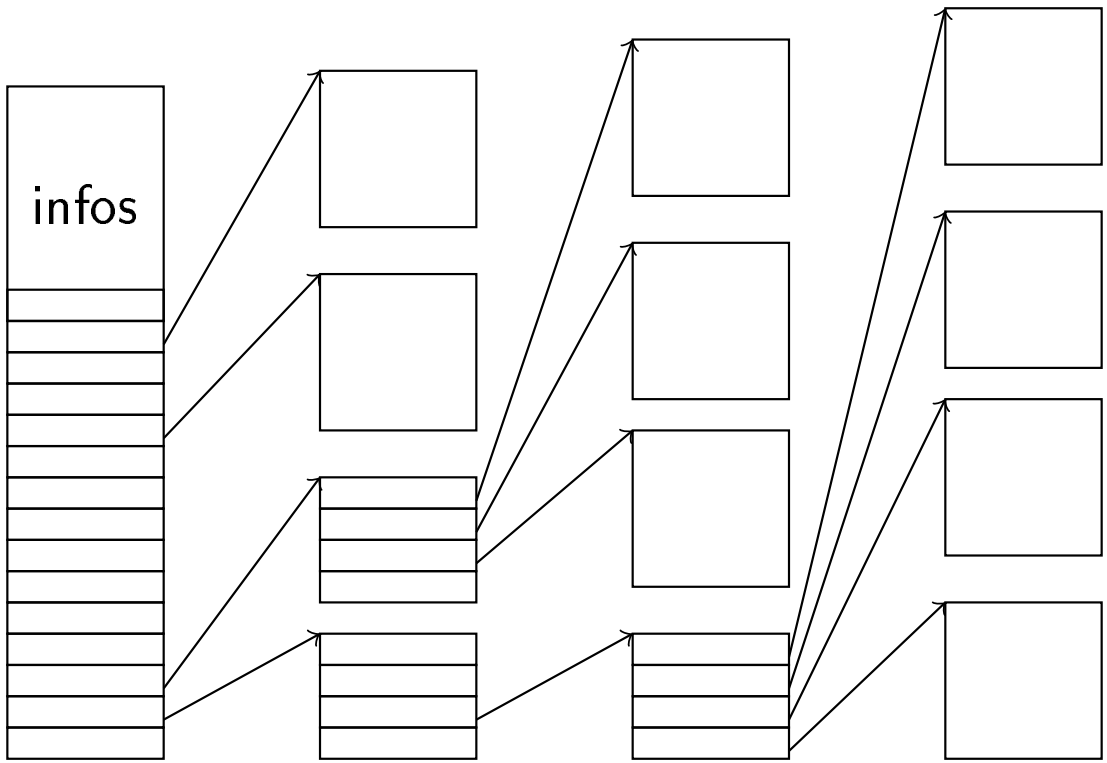

Figure – représentation d'un inoeud

procfs

procfs est un système de fichier virtuel qui permet d'obtenir des informations sur le système à travers une interface classique de fichiers. Notre implémentation de ce système de fichier est spécifique à notre système. Il se structure de la manière suivante :

- tty
- pipes
- null
- display
- un dossier pid pour chaque processus vivant
  - stat
  - cmd
  - fd

Cette interface est très pratique pour développer des outils comme ps ou lsof qui peuvent ainsi facilement accéder aux données du kernel sans avoir à ajouter des appels systèmes complexes.

Système de fichiers virtuel

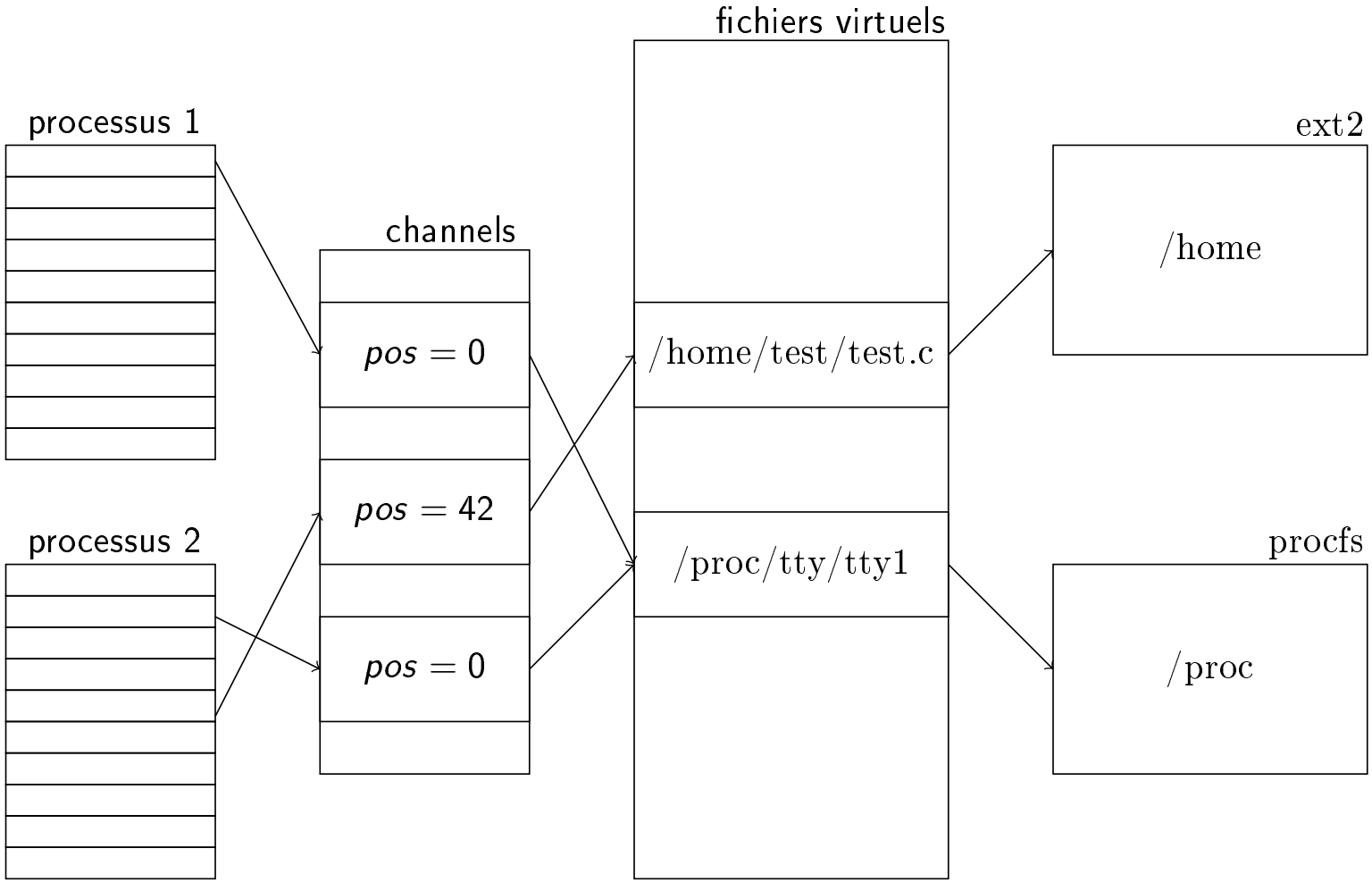

Figure – Les différentes couches d'indirections entre le fichier et le processus

Libc

Dans la libc, on trouve

- des fonctions utilitaires (strlen, malloc...)
- des abstractions au dessus des appels systèmes (opendir, fopen, ...)
- les appels systèmes, qui se sont plus que des interruptions.

Il y a également une séquence de contrôle qui permet d'initialiser la libc, de commencer son programme dans la fonction main et d'exit le programme au retour de main.
Pour éviter de dupliquer le code de la libc dans les binaires chargées dans le système de fichier, on la charge dynamiquement quand on lance le processus.
De plus, grâce aux interfaces les plus communes de la libc, du code générique respectant les normes POSIX pourrait être exécuté sur notre système.

Améliorations possibles

- Bit d'adresse 47
- Amélioration des signaux
- Gestion des flottants
- Le copy-on-write
- Gestion du hardware (devices et drivers)
- Compléter notre implémentation de ext2fs
- Toujours plus de systèmes de fichiers (tmpfs, devfs, ext3, ...)
- Ajouter des utilisateurs
- Les couches réseaux
- Une interface graphique
- Processus légers
- Mémoire partagée

Merci pour votre attention !