

Séance 6

Système d'exploitation embarqué general purpose



Ce(tte) œuvre est mise à disposition selon les termes de la Licence Creative Commons Attribution – Pas d'Utilisation Commerciale – Pas de Modification 4.0 International.

Rappels

- Support de l'exécution de processus en **mode user**
 - Interface avec le kernel via les appels systèmes
 - Mapping des adresses virtuelles vers les physiques
- Support présent dans le kernel pour **gérer les appels systèmes**
 - Structure PROC de représentation des processus
 - Fonction d'initialisation du kernel en code assembleur
 - Routage des appels systèmes vers les fonctions kernel

Objectifs

- Design OS general purpose pour système embarqué (GPOS)
 - Portage d'un GPOS existant vers une architecture cible
 - Développement d'un GPOS embarqué (EOS)
- Détails de certains aspects du design d'un EOS
 - Structure de la mémoire et construction de l'image kernel
 - Séquence de démarrage et processus init
- Quelques exemples d'EOS modernes et récents

Porter un GPOS



General Purpose OS (GPOS)

- Un General Purpose OS (GPOS) est un OS complet
 - Gestion des processus, de la mémoire et des entrées/sorties
 - Présence d'un système de fichiers et d'une interface utilisateur
- Processus dynamiques pour exécuter commande utilisateur
 - Exécution sécurisée dans son propre espace virtuel d'adresses
 - Libération de toutes les ressources utilisées à la terminaison
- Support d'une variété de périphériques d'entrée/sortie

Clavier, souris, stockage de masse...

GPOS embarqué (EOS)

- **Système embarqué** typiquement formée d'un microcontrôleur
 - Utilisé pour monitorer une série de senseurs
 - Permet de générer des signaux pour contrôler des actuateurs
- Évolution **programmes** exécutés sur systèmes embarqués
 - Au début, structure de type super-loop ou orienté évènements
 - Petit à petit, arrivée des systèmes multi-fonctionnels
 - Systèmes embarqués de plus en plus puissants

Portage d'OS

- Portage d'un OS existant est une solution plutôt immédiate
Linux, FreeBSD, NetBSD, Windows...
- Plusieurs EOS actuels sont basés sur le kernel Linux
 - Android utilisé pour appareils avec un écran tactile
 - Version adaptée de Debian pour ARM sur la Raspberry PI
 - Tizen utilisé sur produits Samsung comme smart télé, frigo...
 - Sailfish OS dans in-vehicle infotainment (IVI), yacht...

Type de portage

- Portage de **type procédural** par configuration
 - Kernel GPOS déjà existant pour la plateforme cible (ARM...)
 - Configuration d'entêtes .h et répertoires
- **Adaptation** d'un kernel d'une architecture vers une autre
 - Redesign et ré-implémentation de composants clés
 - Beaucoup plus complexe et couteux à mettre en œuvre



GPOS embarqué

Développer un EOS

- **Adaptation GPOS** pour l'exécuter sur système embarqué
 - PMTX est un GPOS Unix-like pour PC basés sur Intel x86
 - Machine uniprocesseur 32 bits, protection et paging dynamique
 - Portage sur a simple CPU de type ARM
- Caractéristiques de **PMTX**
 - Gestion des processus, de la mémoire et des périphériques E/S
 - Système de fichiers compatible avec EXT-2 de Linux
 - Interface utilisateur sous forme d'une ligne de commandes

ARM Versatilepb VM

- Émulation d'un ARM Versatilepb VM sous QEMU

Exécution d'un EOS sur le système ARM, avec support des E/S

- Cinq **périphériques d'E/S** à supporter sur l'ARM

- Un **SDC** comme stockage de masse principal
- Dispositif d'affichage **LCD** et **clavier** forment la console
- Quatre ports **UART** comme terminal série pour les utilisateurs
- Gestion du scheduling et autres évènements avec quatre **timer**

Organisation de la SDC (1)

- SDC constituée d'**une seule partition** formatée en EXT2

Blocs de 4 Kio de taille et un groupe de blocs

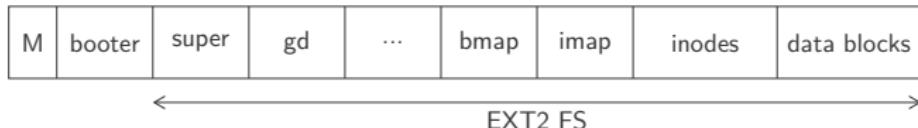
- **Système de fichiers** d'une taille totale de 128 Mio

- Secteur MBR (0) avec table de partitions et début du booter
- Suite du booter à partir secteur 2 chargera l'image kernel
- Partition du SDC montée à la racine du système de fichiers

Organisation de la SDC (2)

- **Cinq parties** principales sur la SDC

- bin exécutables binaires des commandes
- boot images kernel bootable
- dev fichiers spéciaux (pour périphériques E/S)
- etc fichier de mots de passe
- user répertoires home des utilisateurs



Arborescence répertoire source

- Code d'un EOS avec **arborescence répertoire source**

Compilation afin d'obtenir une image kernel bootable

- **Six parties** principales dans l'arborescence

- **BOOTER** contient booter niveaux 1 et 2

- **type.h, include.h, scripts mk de compilation**

Structure données kernel, paramètres du système, constante

- **kernel, fs** (système de fichiers), **driver** (driver périphérique)

- **USER** contient commandes et programmes utilisateur

Fichiers kernel

- Fichiers kernel principalement en C (2% de code assembleur)

Nombre total de lignes de code d'un kernel EOS environ 14000

- Fichiers .c organisés en **quatre parties** logiques

- **Gestion des processus** avec structures de données et fonctions

type.h, ts.s, eoslib

- **Fichiers du kernel** avec toutes ses commandes

*queue.c, wait.c, loader.c, mem.c, fork.c, exec.c, thread.c, t.c
signal.c, except.c, pipe.c, mes.c, syscall.c, suc.c, kernel.c*

- **Driver de périphérique** lcd.h, pv.c, timer.c, kbd.c, uart.c, sd.c

- **Système de fichiers** fs, buffer.c

Séquence de démarrage (1)

1 Booter le kernel de l'EOS

Le BOOTER va charger l'image kernel en mémoire depuis la SDC

2 Exécution du reset_handler pour initialiser du système

Configuration pointeur de pile P0, modes, activation MMU

3 Configuration vecteur interruptions et driver périphériques

4 Exécution du kernel_init pour démarrer le kernel

Initialiser structure données kernel, créer et exécuter P0

Séquence de démarrage (2)

- 5 Construction des **tables des pages** en mémoire

Pagination dynamique à deux niveaux avec pgdir et pgtables

- 6 Initialisation **système de fichiers** et montage sur /

- 7 Création du **processus initial** P1 et exécution de ce dernier

- 8 Fork de P1 pour démarrer **processus de login**

Écoute sur le terminal console et série pour un login utilisateur

Séquence de démarrage (3)

- 9 Gestion du **login d'un utilisateur**

Fork du processus login pour interpréteur de commandes sh

- 10 L'utilisateur **entre des commandes** via une entrée

Exécution des commandes par le processus sh

- 11 **Logout** de l'utilisateur lorsqu'il a terminé son travail

Fork d'un nouveau processus de login par le processus P1

Organisation de la mémoire

- Map mémoire de l'EOS organise les 258 Mio disponibles
Trois principales parties : kernel, données kernel, zone libre
- Mémoire découpée précisément en **huit parties**

Adresses	Description
0–2 Mio	Kernel de l'EOS
2–4 Mio	Buffer pour le LCD d'affichage
4–5 Mio	Zone de données pour 256 buffers E/S
5–6 Mio	Tables des pages niveau 2 Kmode ; 258 pgtables (1 Kio)
6–7 Mio	pgdirs pour 64 processus (16 Kio)
7–8 Mio	Réserve pour extension
8–256 Mio	Cadres libres pour pagination dynamique
256–258 Mio	Espace de 2 Mio pour E/S

Interface utilisateur

- Commande utilisateur comme **fichiers exécutable ELF**
Placés dans le répertoire `/bin` sur device root
- **Trois programmes** principaux pour pouvoir démarrer l'EOS
 - init est le processus initial
 - login gère la connexion des utilisateurs
 - sh reçoit et exécute des commandes

Processus init (1)

- Processus initial P0 handcrafted lors du démarrage de l'EOS
Création d'un fils P1 en chargeant /bin/init en image Umode
- Exécution processus init en mode utilisateur
 - De manière similaire à ce qui se passe dans UNIX/Linux
 - Exemple qui fork un seul processus de login sur la console

Processus init (2)

- Fonction parent représente le **code de P1**

Lance le processus login et attend la fin de son exécution

```
1 int parent()
2 {
3     int pid, status;
4     while (1) {
5         printf("INIT: wait for ZOMBIE child.\n");
6         pid = wait(&status);
7         if (pid == console) {
8             printf("INIT: forks a new console login.\n");
9             console = fork();
10            if (console)
11                continue;
12            else
13                exec("login /dev/tty0");
14        }
15        printf("INIT: I just buried an orphan child proc %d.\n", pid);
16    }
17 }
```

Processus init (3)

- Fonction main lance le **premier processus login**

Après avoir initialisé l'entrée et la sortie standard

```
1 int main()
2 {
3     int in, out;
4     in = open("/dev/tty0", O_RDONLY);
5     out = open("/dev/tty0", O_WRONLY);
6     printf("INIT: forks a new console login.\n");
7     console = fork();
8     if (console)
9         parent();
10    else
11        exec("login /dev/tty0");
12 }
```

Processus login

- Même programme **login** pour tous les processus de login

Chacun des processus est exécuté sur un terminal différent

```
1 int in, out, err; char name[128], password[128];
2
3 main(int argc, char *argv[])
4 {
5     // 1. Fermer descripteurs de fichier 0,1 hérités de INIT
6     // 2. Ouvrir argv[1] trois fois comme in(0), out(1), err(2)
7     settty(argv[1]); // 3. Fixer le tty name dans PROC.tty
8     // 4. Ouvrir /etc/passwd en lecture
9     while (1) {
10         printf("Login:"); gets(name); // 5. Demander login et
11         printf("Password:"); gets(password); // mot de passe user
12         // 6. Rechercher le couple (name, password) dans /etc/passwd
13         if (valid_user) {
14             chuid(); // 7. Changer uid et gid par ceux du user
15             chdir(); // Changer cwd par répertoire HOME du user
16             close(); // Fermer fichier /etc/passwd
17             exec(); // Exécution program pour user
18         }
19         printf("Login failed, try again.\n");
20     }
21 }
```

Processus sh (1)

- Typiquement exécution **processus sh** après un login

Récupère des commandes depuis le user pour les exécuter

- Deux possibilités pour l'**exécution d'une commande**

- Commande simple exécutée depuis /bin (cd, exit...)
 - Sinon fork d'un processus fils pour traiter la commande

- Possibilité de gérer des **symboles spéciaux**

Redirection E/S, pipe pour exécution enchainée...

Processus sh (2)

- Gestion du **symbole de pipe** à l'aide d'une fonction dédiée

Multiples pipes gérés récursivement, de droite à gauche

```
1 int pid, pd[2];
2 pipe(pd);           // Création pipe: pd[0] en read, pd[1] en write
3
4 pid = fork();
5 if (pid) {          // Le parent est reader
6     close(pd[1]);
7     dup2(pd[0], 0); // Redirection stdin au bout de pipe read
8     exec(cmd2);
9 } else {            // Le fils est writer
10    close(pd[0]);
11    dup2(pd[1], 1); // Redirection stdout au bout du pipe write
12    exec(cmd1);
13 }
```

Exemples d'EOS



Tizen

- EOS spécialisé pour devices embarqués très variés

Basé sur le kernel Linux et la librairie C GNU

- Outils de développement basés sur Javascript et HTML5

- Principalement pour IVI, Mobile, TV et Wearable
- Projet de la Linux Foundation, contrôlé par Samsung et Intel



<https://www.tizen.org/>

Sailfish OS

- EOS spécialisé pour les mobiles vraiment indépendant
 - Support des architectures Intel et ARM
 - Multitâche fort, respect vie privée et UI/UX améliorée
- EOS construit à partir du kernel Linux et de Mer

Possibilité d'exécuter des applications Android



<https://sailfishos.org/>

OpenWrt

- EOS **spécialisé pour router** du trafic réseau
 - Construit sur base de Linux et d'outils systèmes (BusyBox...)
 - Optimisation composants pour mémoire routeur domestique
- De **multiples packages** optionnels installables

Routeur CPE, gateway, smartphone, ordinateur de poche...



<https://www.openwrt.org/>

leJOS

- EOS firmware pour brique programmable Lego Mindstorms
 - Supporte les générations de briques RCX, NXT et EV3
 - Inclus une machine virtuelle Java
- Offre un accès aux ports I²C pour les senseurs et moteurs
Inclus des algorithmes de contrôle : PID, filtre de Kalman...



<http://www.lejos.org/>

Tock

- EOS spécialisé pour l'IoT (senseur, wearable)
 - Exécution concurrente de multiples applications méfiantes
 - Adapté pour microcontrôleurs low-memory et low-power
- Sécurité/sûreté, fiabilité et complètement **low-power**

```
1 int main() {
2     while(1) {                      // 5µA (on imix development board)
3         led_toggle(0);
4         delay_ms(5000);
5     }
6 }
```

Tock

<https://www.tockos.org/>

Crédits

- <https://www.flickr.com/photos/bfaling/5569427691>
- <https://www.flickr.com/photos/cernicalo-e/5426187612>
- <https://www.flickr.com/photos/akyamada/8038557981>