

I403C Systèmes d'exploitation avancé

Séance 3

Système d'exploitation embarqué

Sébastien Combéfis

2018–2019



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.

Objectifs

- Développement de **systèmes embarqués**
 - Définition et caractéristiques des OS embarqués
 - Design et développement d'OS pour les systèmes embarqués
- **Version de Linux** adaptée pour les systèmes embarqués
 - Caractéristiques d'un Linux embarqué*
- Développement d'**OS embarqué spécifique**
 - TinyOS pour les réseaux de senseurs sans fil*

Système embarqué (1)

- Grande présence et importance des **OS embarqué**
 - Exigences uniques et particulières causées par l'environnement
 - Stratégie de design de tels OS embarqués très différente
- Utilisation de **hardware et de software** au sein d'un produit
 - Couplage fort du système embarqué avec son environnement
 - Possibilité de contraintes de temps réel (souple/forte)
 - Possibilité de plusieurs activités concurrentes/parallèles

Système embarqué (2)

- Plusieurs **contraintes** liées à l'environnement

Vitesse de déplacement, précision des mesures, durée précise

- **Organisation** particulière d'un système embarqué
 - Interfaces de mesure et interaction avec environnement externe
 - De la simple LED qui flashe à la vision robotique temps réel
 - Addition de FPGA, ASIC ou même hardware non numérique
 - Software avec une fonction fixe et spécifique à l'application

Caractéristique OS embarqué



OS embarqué (1)

- Système embarqué simple contrôlé par **software dédié**

Ensemble de programmes directement exécutés, sans autre

- Système plus complexe nécessite un **OS embarqué**

- Version spéciale d'un general-purpose comme Linux embarqué
- Développement d'un OS dédié comme TinyOS

- Plusieurs **caractéristiques** uniques aux systèmes embarqués

Liées à espace mémoire, consommation de puissance, temps réel...

Environnement

- Système embarqué très fortement lié avec **son environnement**

Grande hétérogénéité des environnements

- Plusieurs **niveaux d'exigences** très variés

- Applications critiques avec beaucoup de fonctionnalités

Médical, navette spatiale, process automation...

- Applications critiques avec des petites fonctionnalités

Système ABS, pace maker...

- Pas d'aspect critique et fonctionnalités variées

Smartphone, smartcard, four à micro-ondes...

OS embarqué (2)

- Parfois nécessité d'avoir un **OS embarqué**

Même raison que ordinateur traditionnel, services pas tous utiles

- Un OS classique de bureau n'est **pas adapté**
 - Kernel monolithique trop riche, pas modulaire, configurable...
 - Trop d'espace mémoire, trop gourmand en temps de calcul
 - Pas conçu pour applications critiques
 - Les timings sont beaucoup trop larges

Caractéristique (1)

- Opérations à devoir réaliser sous **contraintes temps réel**
 - Correctness d'un calcul dépend du moment où il est réalisé
 - Contraintes temps réel dictées par E/S externe
- Opérations exécutées de **manière réactive**
 - Software exécuté en réponse à des évènements externes
 - Évènement pas forcément périodique, ni intervalle prédictible
 - Priorité d'exécution pour les routines et gestion pire cas

Caractéristique (2)

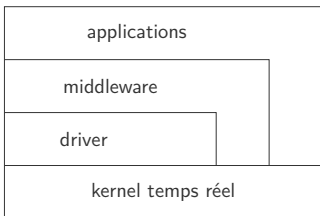
- Grande **configurabilité** du système embarqué
 - Pouvoir exécuter OS embarqué sur plusieurs systèmes
 - Configuration flexible pour n'avoir que le nécessaire
- **Flexibilité** des périphériques d'entrée/sortie
 - Pas de périphériques qui doivent être supportés par tous les OS
 - Le nombre de périphériques d'E/S est très grand
 - Ne pas intégrer drivers de périphériques lents dans le kernel

Caractéristique (3)

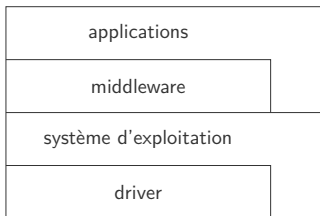
- Mécanismes de **protection** simplifiés
 - Système embarqué pour usage limité et fonction bien définie
 - Software pas testé rarement ajouté par après
 - Système embarqué fiable, peu de mécanismes de protection
- Utilisation directe des **interruptions**
 - Accès direct aux interruptions pour les processus utilisateur
 - Pas besoin de protection du système embarqué
 - Nécessité d'avoir un contrôle efficace de périphériques

Driver de périphérique

- **Drivers de périphérique** gérés comme des tâches
 - Amélioration de la prédictabilité en passant par scheduler
 - Pas de périphériques à supporter par toutes versions de l'OS



RTOS



OS standard

OS temps réel

- Système d'exploitation pour construire **systèmes temps réel**

Combinaison possible OS embarqué et temps réel

- **Trois exigences** clés pour avoir un RTOS

- Comportement lié au timing doit être prédictible

Temps d'exécution doit être borné pour tous les services

- L'OS temps réel doit gérer le timing et ordonnancement

Prise en compte des deadlines et services précis pour temps

- L'OS temps réel doit être **rapide**

Adaptation OS commercial

- Ajout de **capacités temps réel** sur un OS commercial existant
Opérations de rationalisation, nouvelles fonctionnalités...
- Souvent à partir de **Linux**, mais possibilité avec Windows...
 - OS standard plus lent et moins prédictible
 - Interfaces familières et meilleure portabilité
 - Pas optimisé pour le temps réel et les applications embarquées
 - Optimisation cas moyen et ressources affectées à la demande

OS embarqué dédié (1)

- Design **OS embarqué** dès le départ comme eCos et TinyOS

Conçus dès le départ pour des systèmes embarqués

- Plusieurs **aspects de spécialisation** des OS embarqués
 - Switch de processus et de threads léger et rapide
 - Politique ordonnancement temps réel avec dispatcher inclus
 - Petite taille
 - Rapide réponse interruptions, désactivation très brèves
 - Partitions de tailles fixes ou variables pour gérer mémoire
 - Fichiers séquentiels spéciaux pour accumuler données très vite

OS embarqué dédié (2)

- **Features kernel** pour gérer les contraintes de timing
 - Temps d'exécution borné pour la plupart des primitives
 - Existence d'une clock temps réel
 - Alarmes spéciales et gestion de time out
 - File d'attente spéciales pour temps réel (early deadline first...)
 - Retarder exécution d'un processus et le reprendre
- Parfois importance de la **prédictabilité** versus performance

Impact important sur design, surtout ordonnancement de tâches



Linux embarqué

Linux embarqué

- Version de Linux spécifique aux **systèmes embarqués**
 - Customisée pour contraintes de taille et de hardware
 - Ensemble de services software adapté au système
- Optimisation au niveau **hardware ou software**
 - Composants hardware spécifiques comme PRU
 - Drivers particuliers ou co-kernel temps réel

Taille du kernel

- Différence par rapport au **nombre de périphériques** à gérer
 - Important nombre de périphériques et de configurations
 - Support de plusieurs protocoles d'échange et communication
- Périphérique et protocole **limités** sur système embarqué
 - Dépendance forte avec le hardware disponible
 - Kernel Linux est très configurable par rapport à l'architecture

Cross-compilation

- **Compilation sur système hôte** en développement classique
Utilisation d'un compilateur qui produit le code machine
- Compilation pour une autre plateforme avec **cross-compiler**
 - Code compilé sur système hôte pour système cible
 - Compilateur spécifique pour les différentes cibles

Système de fichiers embarqué

- **Stockage permanent** sous la forme de mémoire flash

Contrairement aux disques sur système classique

- Existence de **systèmes de fichiers compacts**
 - **cramfs, squashfs** : lecture seule, compressé et en RAM
 - **jffs2** : basé sur des logs sur flash NOR/NAND
 - **ubifs** : meilleur sur gros fichier et cache d'écriture
 - **yaffs2** : nécessite moins de RAM pour stocker informations

Force du Linux embarqué

- Premier Linux embarqués apparus vers 1999

Multitude de versions développées par des compagnies

- Quatre avantages principaux d'utilisation de Linux embarqué
 - Indépendance par rapport aux vendeurs
 - Support varié du hardware et des périphériques
 - Minimisation des couts de développement et formation
 - Tous les avantages de l'open source

- OS mobile Android basé sur un **kernel Linux**

Peut être considéré comme Linux embarqué, selon la vue adoptée

- **Comparaison** entre Android et système embarqué
 - Un système embarqué doit avoir une fonction précise et fixe
 - Plateforme OS qui supporte plusieurs types d'applications
 - Système intégré verticalement et espace user Android



TinyOS

- eCos plus adaptés pour petits systèmes embarqués

Exigence stricte concernant mémoire, temps de calcul, réponse temps réel, consommation électrique...

- Niveau de rationalisation encore plus grand pour TinyOS
 - Développé pour un réseau de senseurs sans fil
 - Le système embarqué est la plupart du temps éteint
 - TinyOS n'est pas un OS temps réel

- TinyOS populaire et utilisé par plus de 500 organisations

Existence de standard open source pour TinyOS

- Inexistence d'un kernel sous TinyOS
 - OS orienté composants et pas de protection mémoire
 - Pas de processus, ni de système d'allocation de mémoire
 - Gestion d'interruptions dépend du périphérique
 - Non bloquant avec quelques primitives synchronisation

Réseau de senseurs sans fil

- Réseau de **senseurs** low powered et de petite taille
 - Connectés entre eux avec un réseau sans fil
 - Micro-electromechanical sensors (MEMS), transducers
- Existence d'un **software** compact pour gérer le senseur
Prendre la mesure et réaliser la communication
- Architecture composée des senseurs et d'un **PC hôte**
 - Station de base du réseau connectée au PC hôte
 - Senseurs jouent le rôle de relais de données (il faut du routage)
 - Possibilité d'auto-organisation en un réseau ad-hoc

But de TinyOS (1)

- Réseau de senseurs distribués comme application visée

Travail par un groupe de chercheurs de l'UC Berkeley

- Identification de six buts principaux pour TinyOS

- Permettre la forte concurrence entre plusieurs flux de données

Flux régulier de données captées et traitées

- Ressources limitées (mémoire, puissance calcul, énergie)

Utilisation efficace des ressources et communication low-power

- Adaptation aux évolutions hardware

Portabilité la plus grande possible sur différent hardware

But de TinyOS (2)

- Identification de **six buts** principaux pour TinyOS
 - Supporter une très large gamme d'applications
OS embarqué le plus modulaire possible
 - Supporter un ensemble de plateformes variées
OS embarqué plutôt orienté general-purpose
 - Robustesse du système d'exploitation
Réseau de senseurs sans surveillance pendant des mois ou années
- **Limitation du buffering** dans le réseau pour éviter latences
Et mémoire limitée et communication sur distance courte

Composant (1)

- Système logiciel pour TinyOS est un ensemble de **composants**

Petit module qui réalise une tâche ou ensemble de tâches simple

- **Interactions** entre composants et avec le hardware

De manière limitée et très bien définie

- Seul module software toujours présent est l'**ordonnanceur**

- Il n'y a pas de kernel et donc pas vraiment d'OS
- Architecture software rigide et simplifiée pour gérer le réseau

Composant (2)

- Plusieurs **composants open source** pour TinyOS
 - Besoins de base pour le réseau de senseurs connectés
 - Single-hop networking, ad-hoc routing, power management...
- **Collection de composants** standardisés forme TinyOS

La collection est l'OS avec lequel applications sont développées

- **Ordonnanceur** qui opère à travers plusieurs composants

Une seule tâche choisie car système uniprocasseur

- Algorithme par défaut est une **simple FIFO**
 - Peut mettre le processeur en sleep car power aware
 - Un slot par tâche dans la file
 - Possibilité pour l'utilisateur de remplacer le scheduler
- **Deux origines** possibles pour les tâches à exécuter
 - Placée en file comme résultat d'un évènement
 - Requête spécifique d'une tâche en cours d'exécution

Gestion des ressources

- Utilisation de **trois abstractions** pour gérer les ressources
 - Dédinée lorsque besoin d'accès exclusif tout le temps
Interruptions et compteurs
 - Virtualisée pour simuler dédiée et protection par mutex
Clock ou timer
 - Partagée à l'aide d'un composant d'arbitrage
- **Contrôle total** d'une ressource tant qu'elle est détenue
Arbitrage entre des clients coopératifs

Crédits

- <https://www.flickr.com/photos/marcus-hebel/8237627637>
- <https://www.flickr.com/photos/marcusmeissner/5543614166>
- <https://www.flickr.com/photos/39683118@N07/14977017211>