Soutenance de thèse

Conception, implémentation et preuve d'un service de transfert de flot d'exécution au sein d'un noyau de système d'exploitation

Florian Vanhems – florian.vanhems@univ-lille.fr

Jeudi 2 Mars 2023

Introduction

Contexte général et portée des travaux

Sécurité informatique de systèmes critiques

Dysfonctionnement provoque la mise en péril de vies humaines ou engendre des coûts effarants

- Transport
- Énergie
- Spatial

Preuve de propriétés formelles sur le logiciel embarqué dans ces systèmes

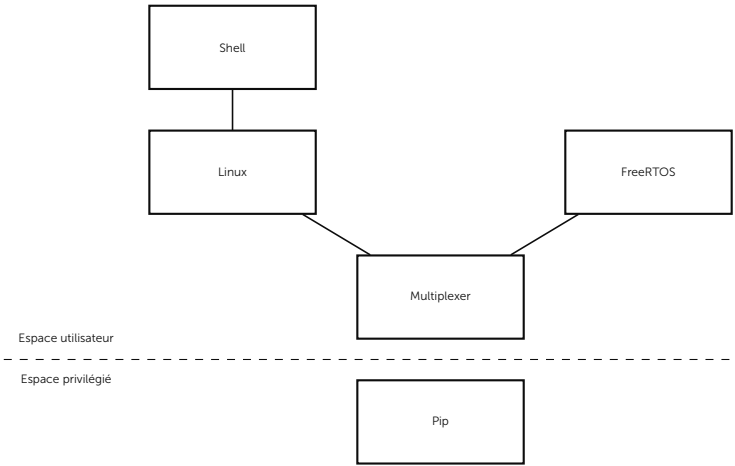
Sam Curry (@samwcyo) sur Twitter le 29 Novembre 2022

"We recently found a vulnerability affecting Hyundai and Genesis vehicles where we could remotely control the locks, engine, horn, headlights, and trunk of vehicles made after 2012.
To explain how it worked and how we found it, we have @_specters_ as our mock car thief :"

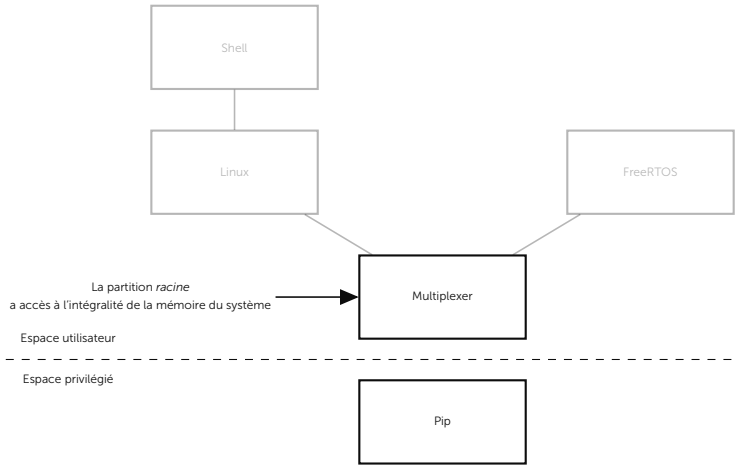
Pip, un noyau minimal aux garanties fortes

- Permet de créer des partitions de mémoire isolées les unes des autres
- Isolation rendue possible par un dispositif matériel (MMU ou MPU selon les systèmes)
- Muni d'une preuve formelle de préservation de l'isolation (bonne configuration du matériel)
- Support de familles d'architectures x86 et Armv7

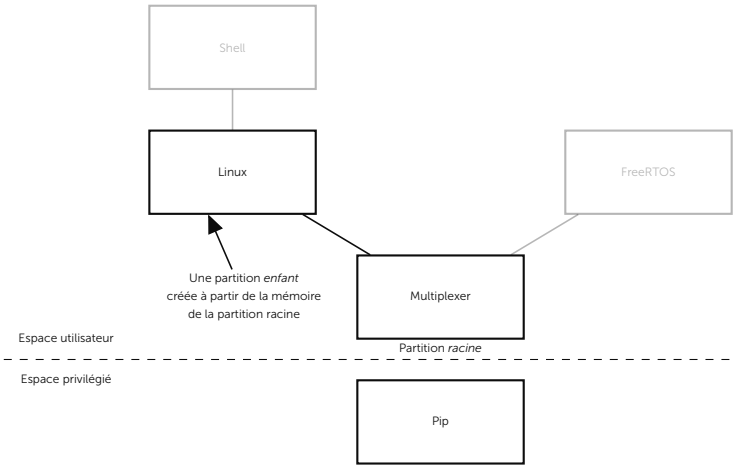
L'arbre de partition de Pip



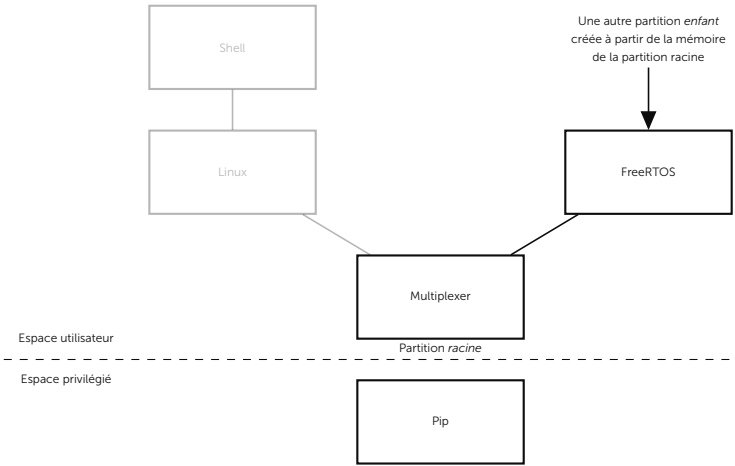
L'arbre de partition de Pip



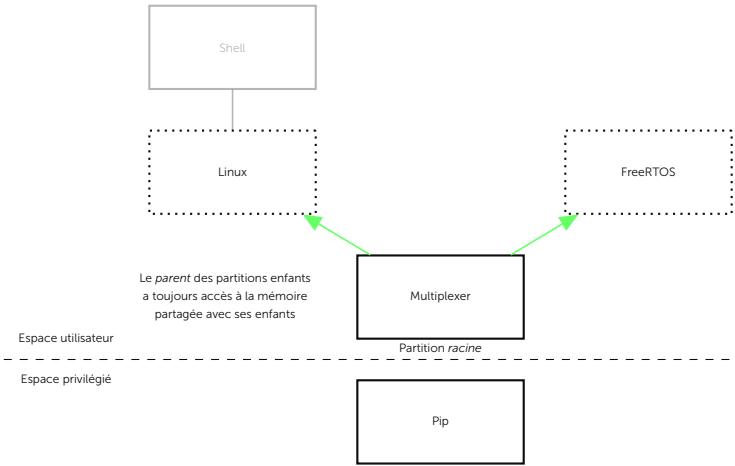
L'arbre de partition de Pip



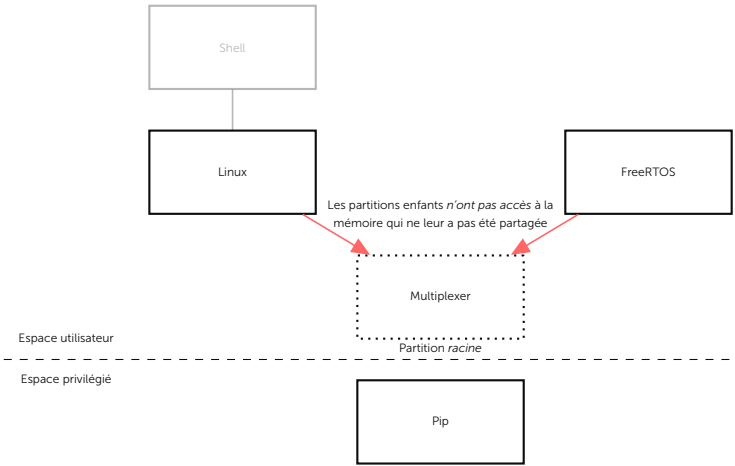
L'arbre de partition de Pip



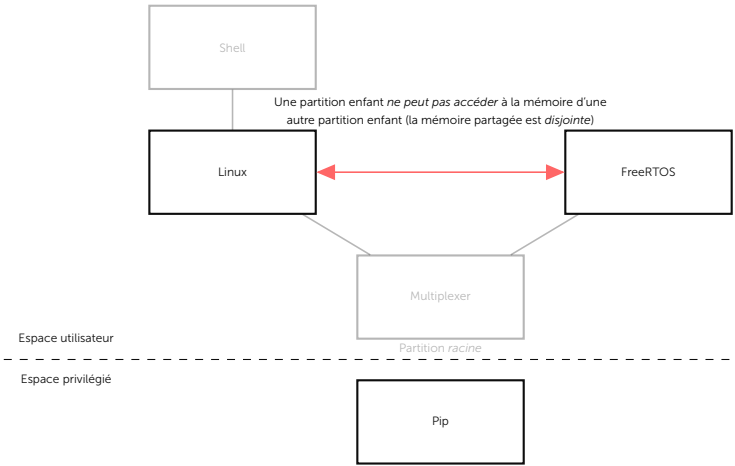
L'arbre de partition de Pip



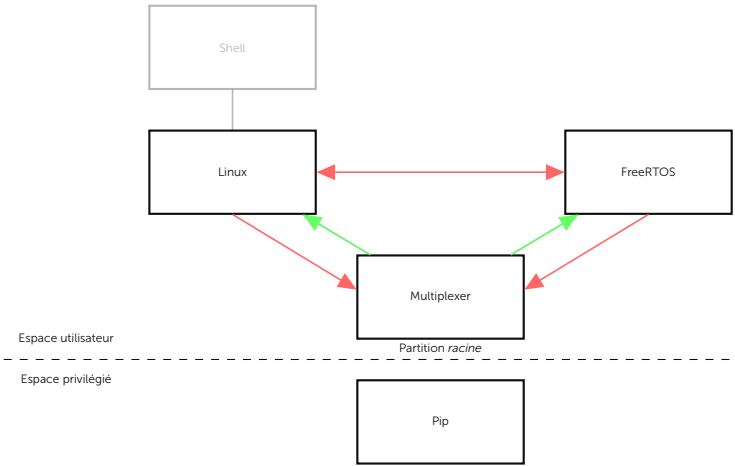
L'arbre de partition de Pip



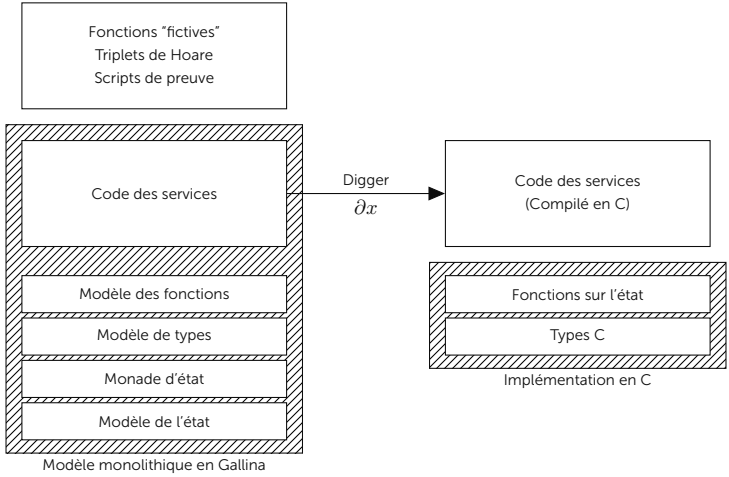
L'arbre de partition de Pip



L'arbre de partition de Pip



L'architecture de Pip



Historique des travaux sur le projet Pip

Première génération de doctorants travaillant sur les propriétés d'isolation :

- Aspect système : conception du noyau
- Aspect vérification : code des services et preuve
- Aspect embarqué : travaux sur la cohabitation d'entités différentes au sein du système

Problématique de partage du temps CPU à traiter

Contributions de la thèse

- Service de transfert de flot d'exécution entre les partitions
- Développement d'un ordonnanceur Earliest Deadline First dans une partition
- Réflexions sur l'ajout de nouvelles propriétés à Pip

1. Introduction
2. Service de transfert de flot d'exécution unifié
   - Un nouveau service pour les transferts explicites
   - Généralisation aux fautes et aux interruptions
   - Vérification formelle
   - En résumé
3. Ordonnanceur Earliest Deadline First
   - Place de l'ordonnancement dans Pip
   - Vue générale de l'ordonnanceur
   - Hypothèses & preuves
   - Résumé de la contribution
4. Réflexions sur l'établissement de nouvelles preuves
   - Proposition de nouvelle architecture
   - Perspectives liées à la nouvelle architecture
5. Conclusion

Service de transfert de flot d’exécution unifié

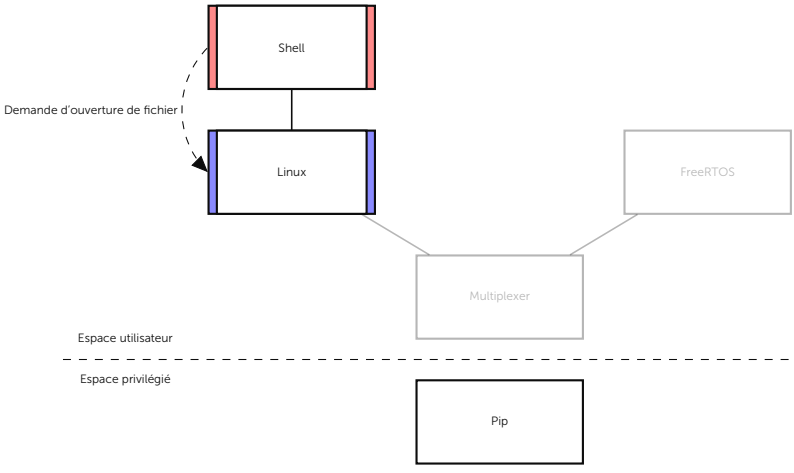
Motivations

- Généraliser la précédente implémentation du service
- Compléter la preuve de préservation de l'isolation des services fournis par Pip
- Éprouver la méthodologie de Pip

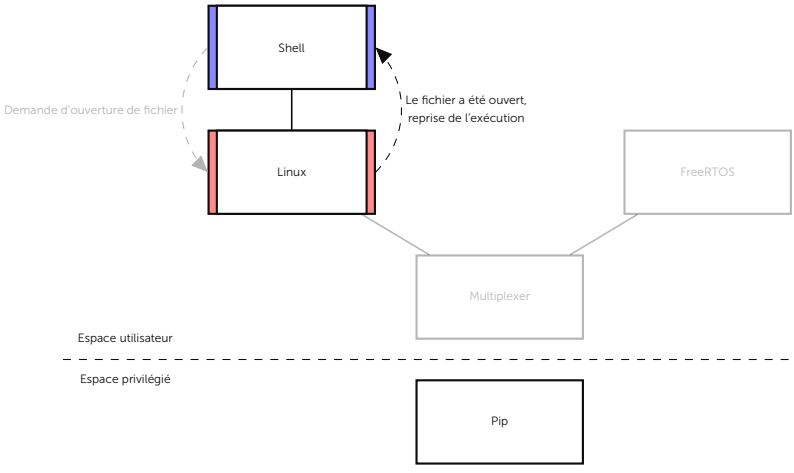
Quels objectifs ?

- Gérer les fautes et interruptions
- Minimiser le nombre de lignes de code pour faciliter la preuve de maintient de l'isolation
- Ne pas être lié à une architecture spécifique

Transferts de flot d'exécution explicites au sein de Pip



Transferts de flot d'exécution explicites au sein de Pip



Service de transfert de flot d'exécution unifié

Un nouveau service pour les transferts explicites

Idée principale : la *VIDT*

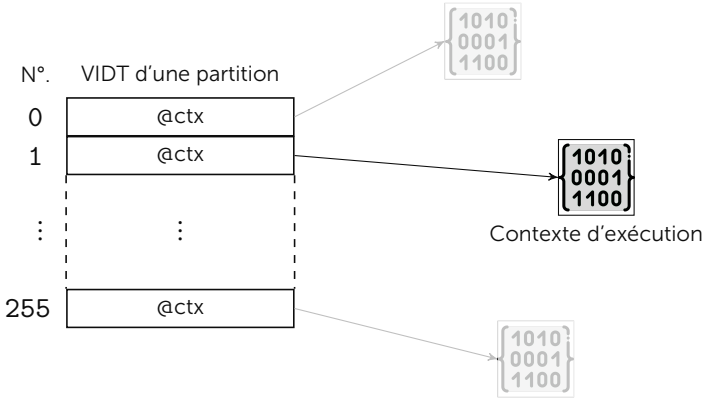


Illustration du fonctionnement du service

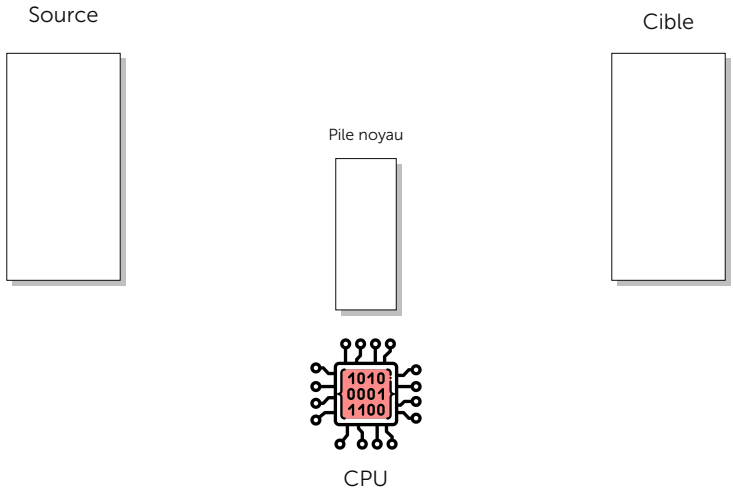


Illustration du fonctionnement du service

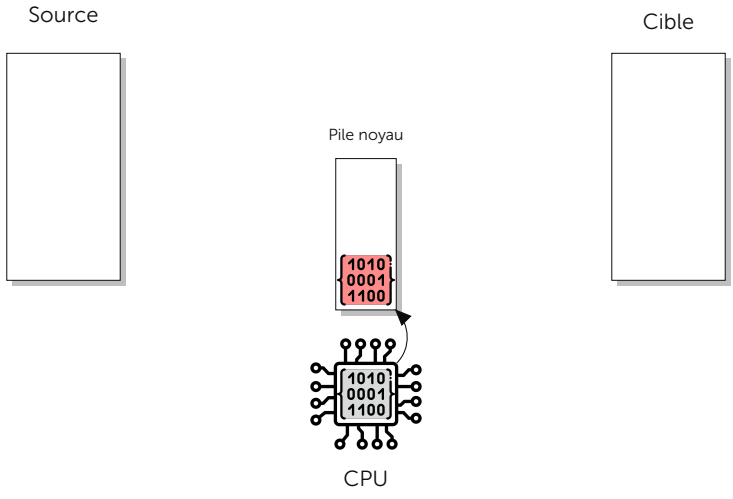


Illustration du fonctionnement du service

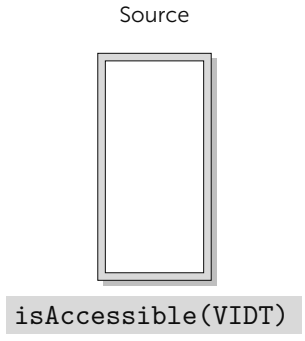


Illustration du fonctionnement du service

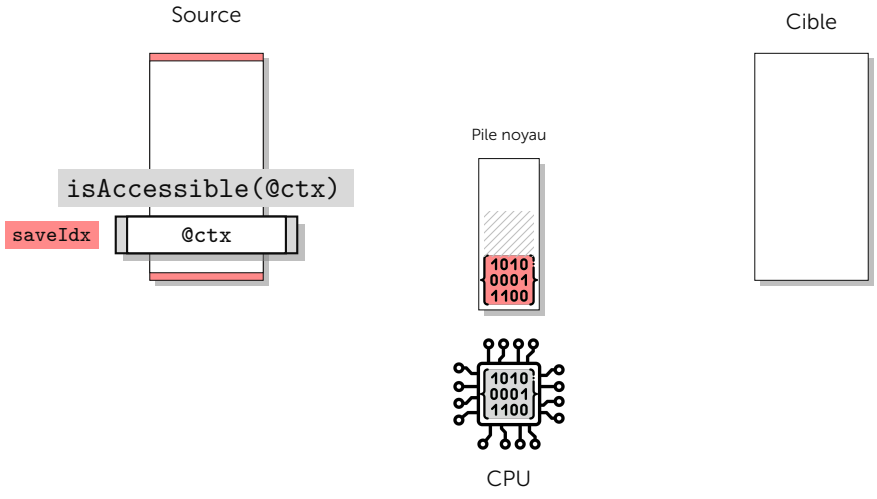


Illustration du fonctionnement du service

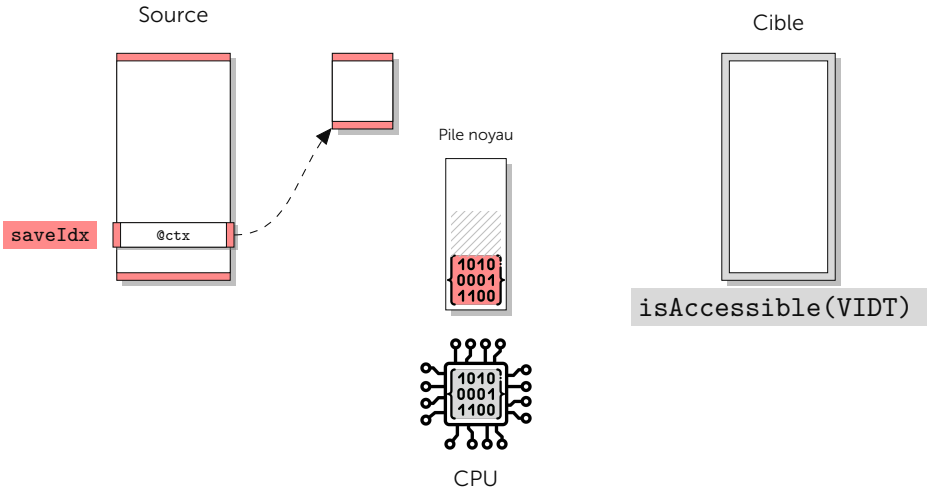


Illustration du fonctionnement du service

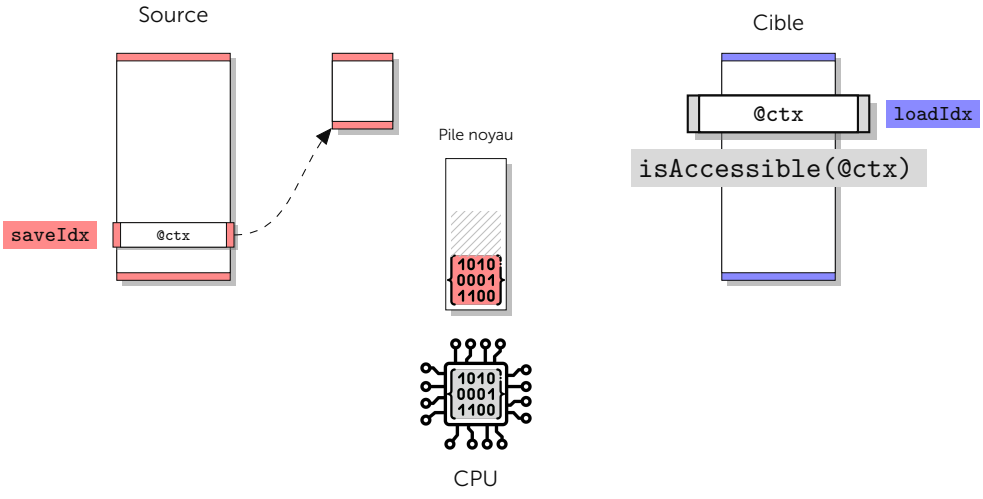


Illustration du fonctionnement du service

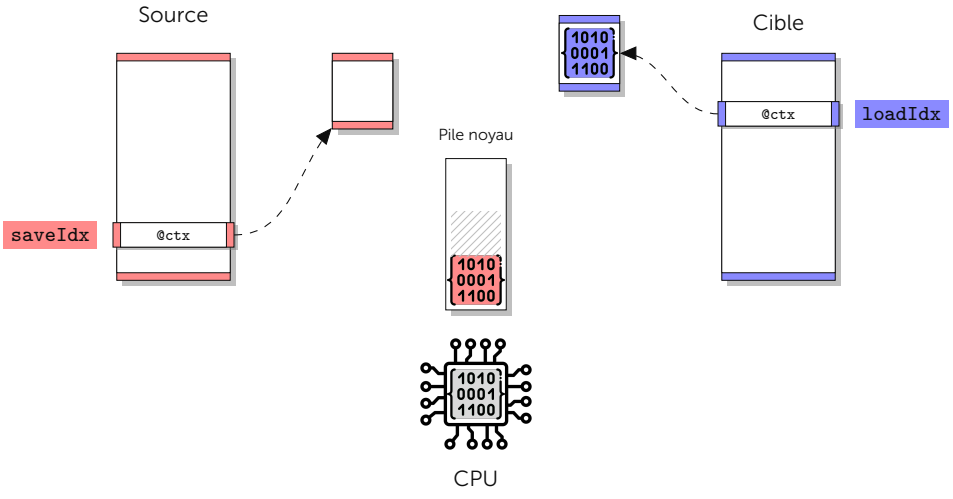


Illustration du fonctionnement du service

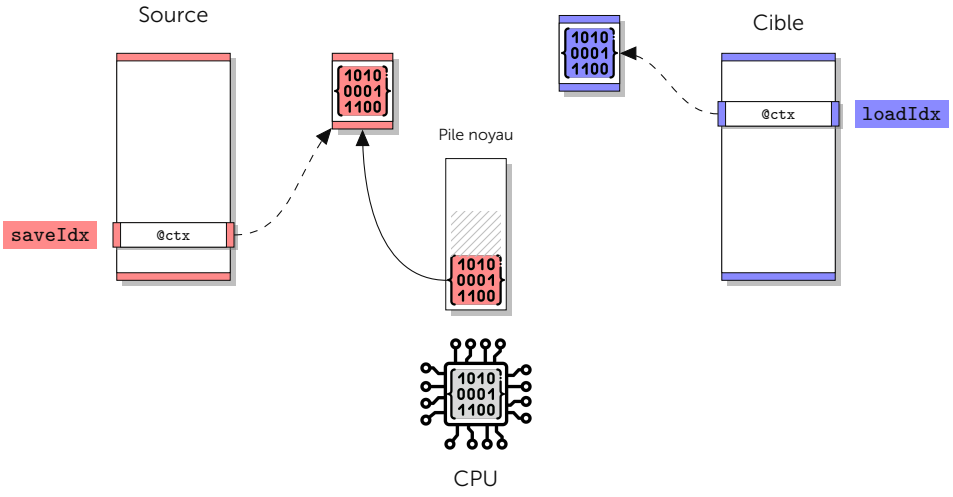


Illustration du fonctionnement du service

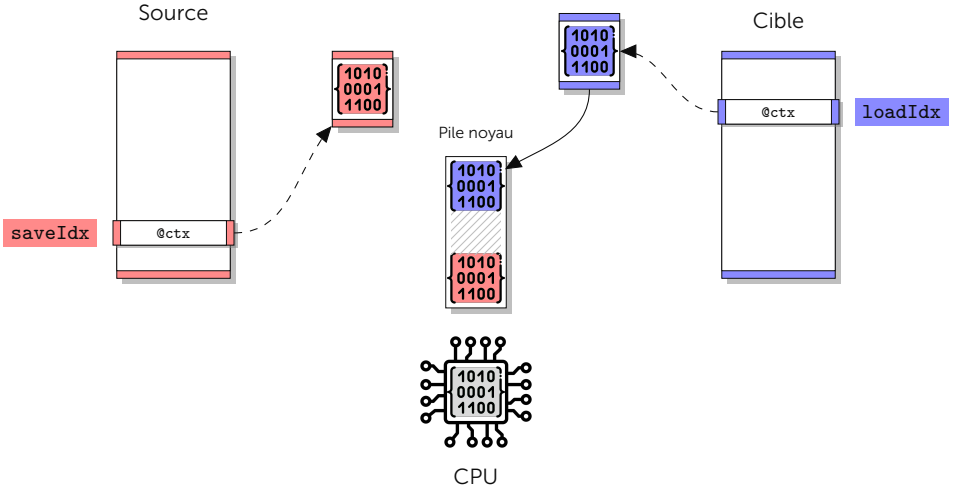
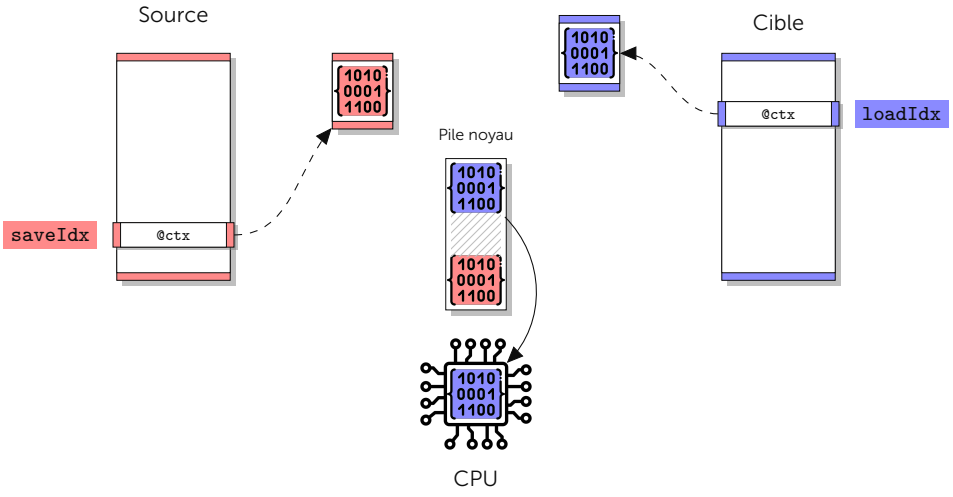


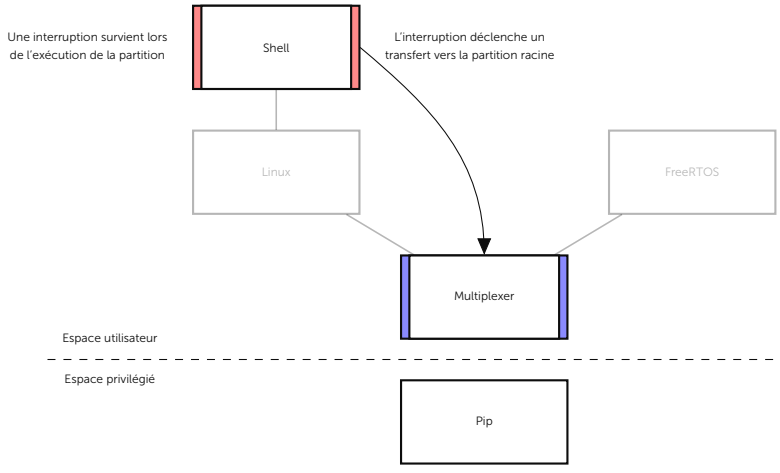
Illustration du fonctionnement du service



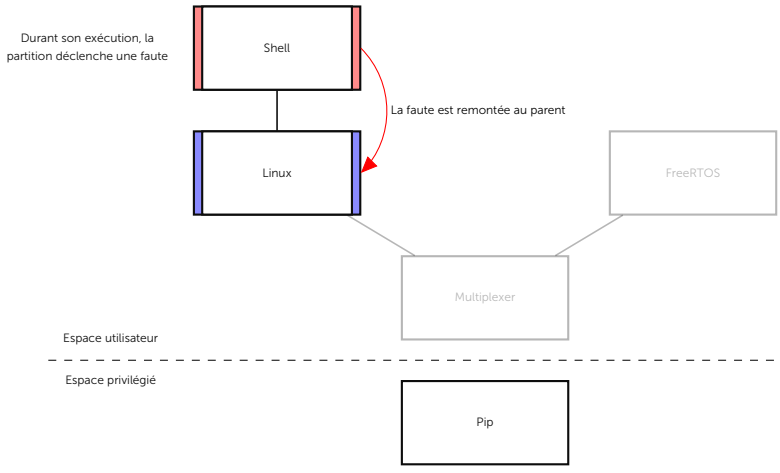
Service de transfert de flot d'exécution unifié

Généralisation aux fautes et aux interruptions

Interruptions matérielles



Fautes


Durant son exécution, la partition déclenche une faute
Shell
La faute est remontée au parent
Linux
FreeRTOS
Multiplexer
Espace utilisateur
Espace privilégié
Pip

Service de transfert de flot d'exécution unifié

Vérification formelle

Preuve de préservation de l'isolation

- Preuve complétée sur des modèles restreints
- Preuve avec des modèles plus complets en cours

Limites de la preuve

Preuve d'isolation nécessaire, mais peu surprenante pour ce service

Preuve de bon fonctionnement plus adéquate, **mais** le code de Pip est lié aux modèles d'isolation

Service de transfert de flot d’exécution unifié

En résumé

Les points clés

- Service de transfert de flot d'exécution capable de gérer
  - Transferts explicites
  - Fautes
  - Interruptions
- … en un flot d'exécution unique !
- preuve d'isolation réalisée
- support de multiples architectures

mais

- Preuve d'isolation peu surprenante
- Utilisabilité doutable, car seulement montré sur des exemples jouets

Ordonnanceur Earliest Deadline First

Motivations

- Montrer que le service de transfert de flot d'exécution est utilisable
- Utilisabilité de la méthodologie de preuve de Pip en espace non privilégié
- Mettre un autre pied dans le monde des systèmes embarqués

Ordonnanceur Earliest Deadline First

Place de l'ordonnancement dans Pip

La place d'un ordonnanceur dans Pip

Pip est un noyau minimal : ordonnancement relégué en espace utilisateur

- Multiplicité des politiques d'ordonnancement
- Ajout de logiciel dont les propriétés à prouver ne sont pas des propriétés d'isolation

Illustration sur un arbre de partitions

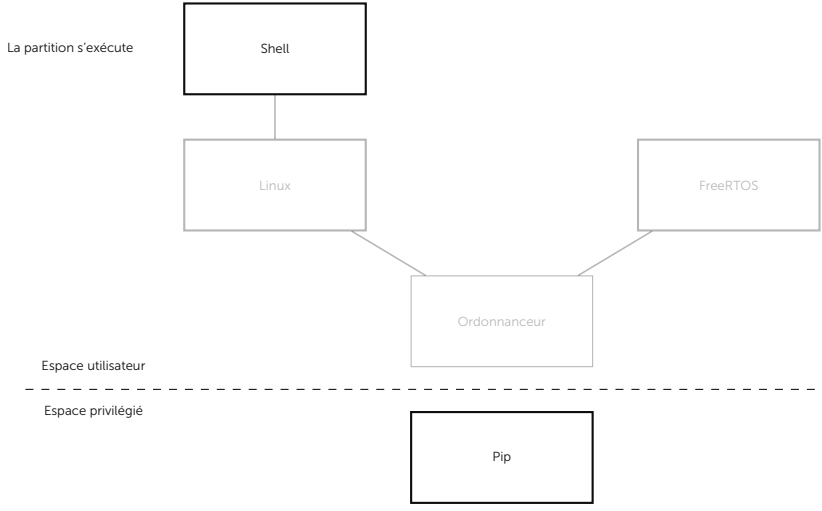


Illustration sur un arbre de partitions

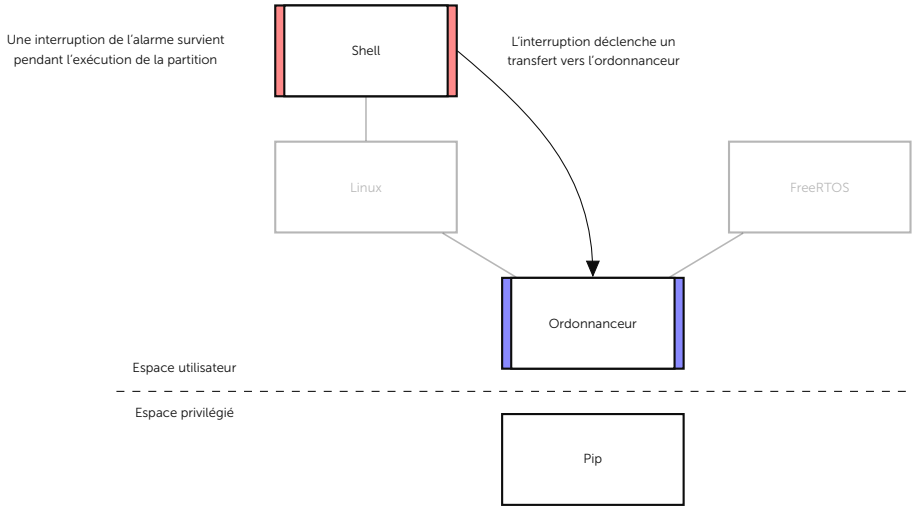


Illustration sur un arbre de partitions

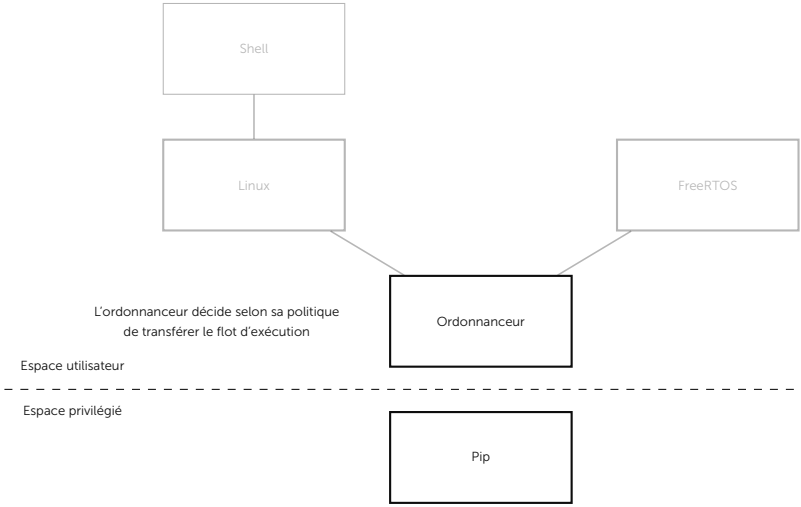
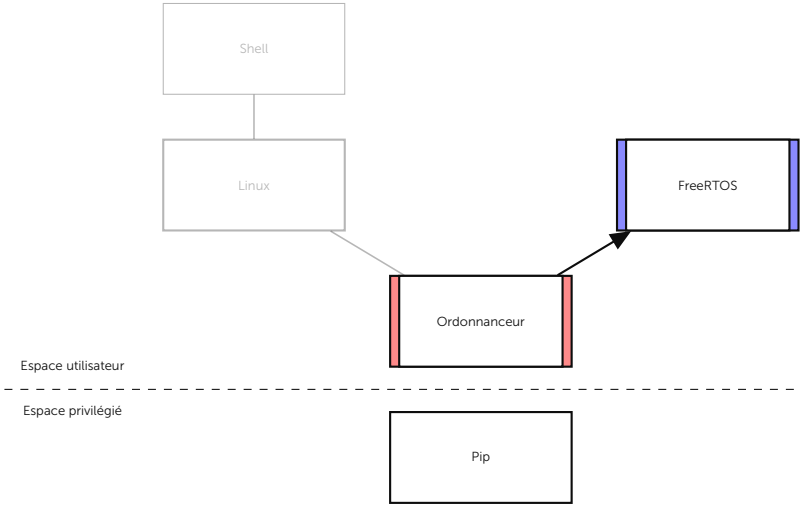


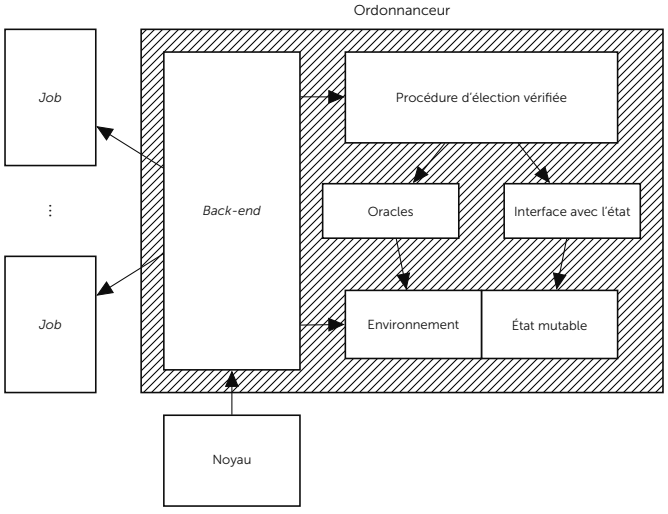
Illustration sur un arbre de partitions



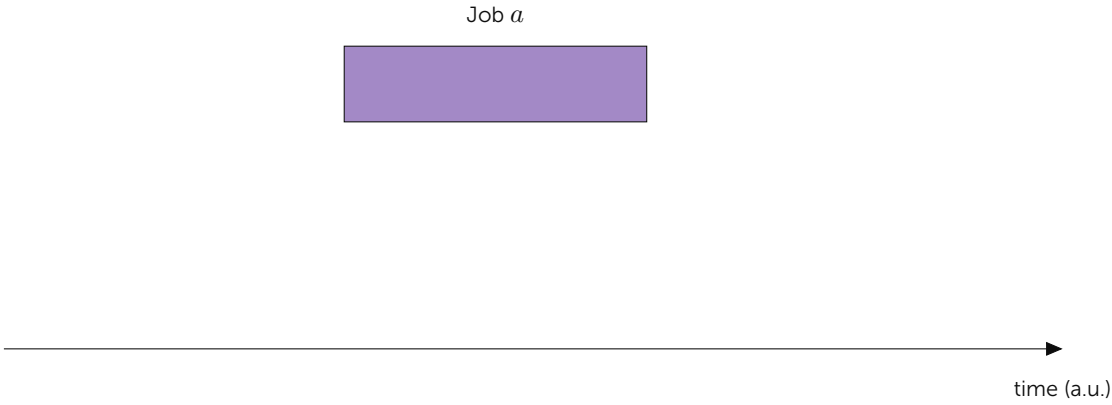
Ordonnanceur Earliest Deadline First

Vue générale de l'ordonnanceur

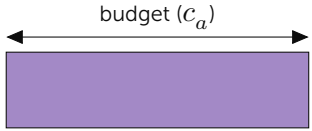
Présentation des composants de l'ordonnanceur



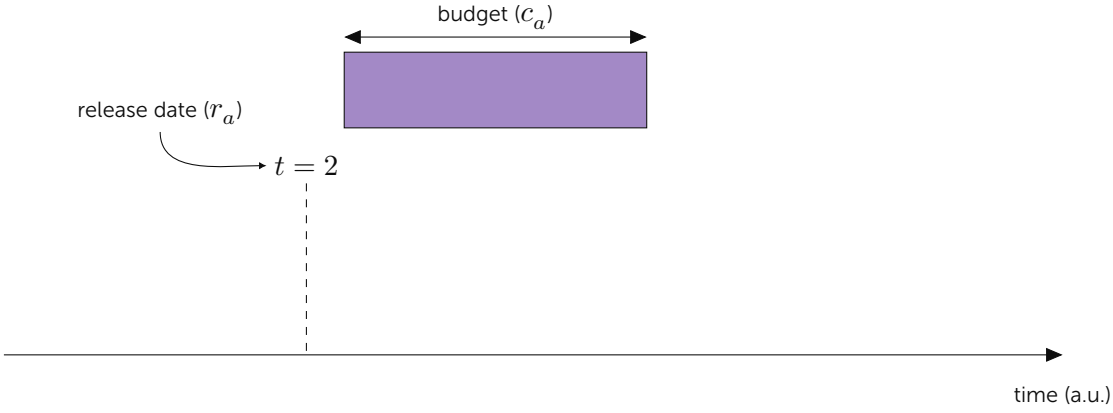
Prototype et fonctionnement de la fonction d'élection EDF



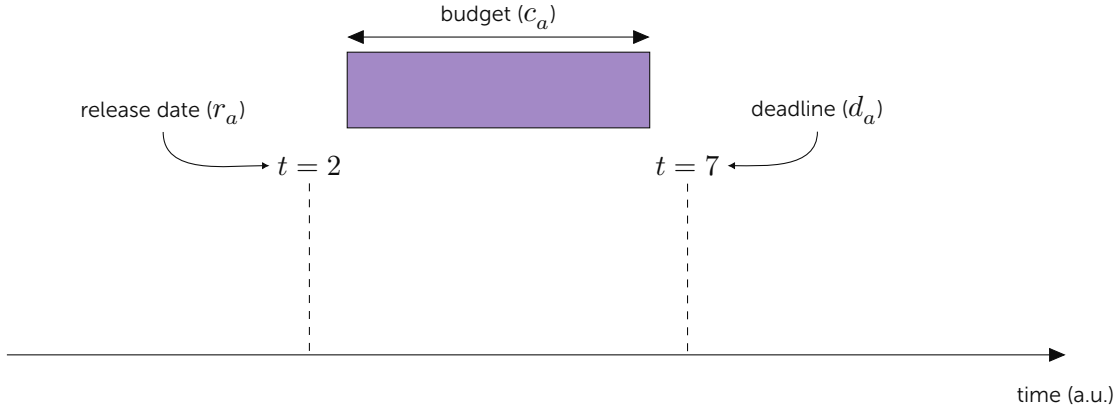
Prototype et fonctionnement de la fonction d'élection EDF



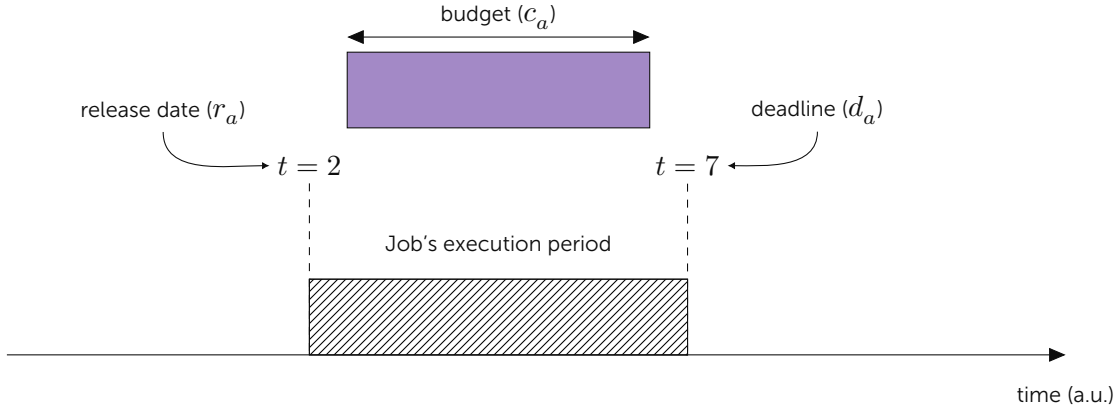
Prototype et fonctionnement de la fonction d'élection EDF



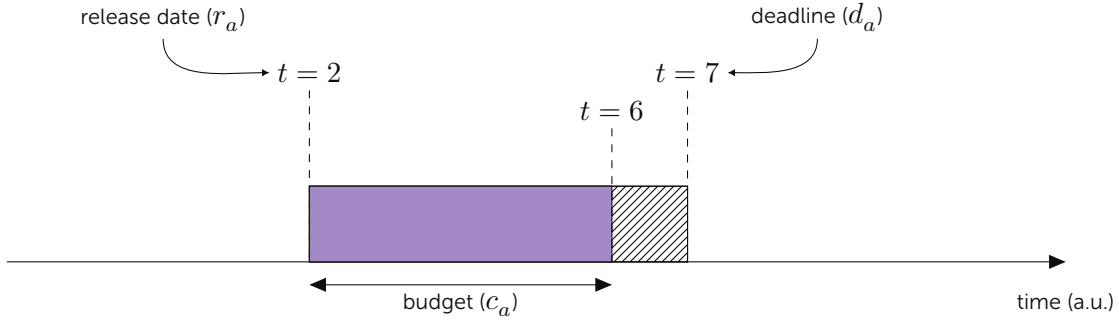
Prototype et fonctionnement de la fonction d'élection EDF



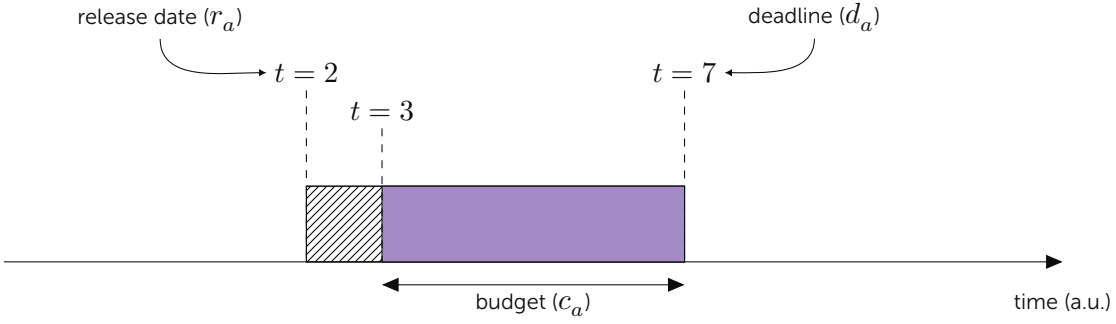
Prototype et fonctionnement de la fonction d'élection EDF



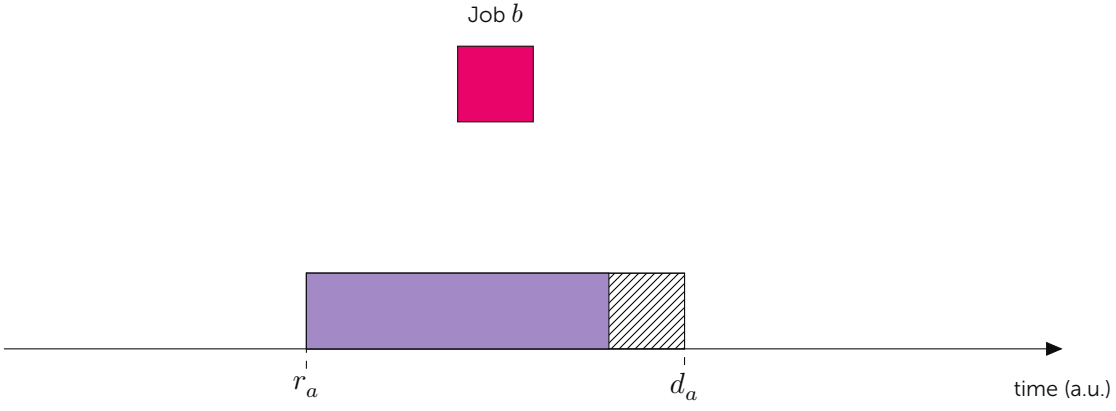
Prototype et fonctionnement de la fonction d'élection EDF



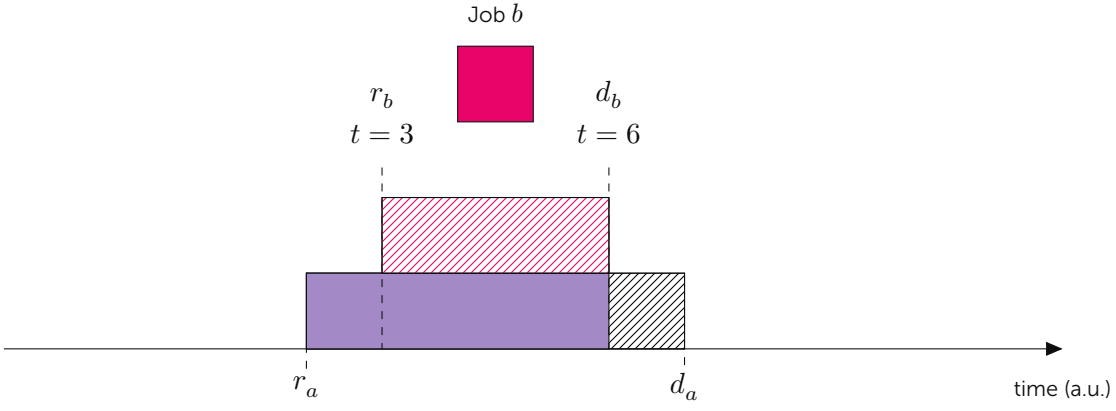
Prototype et fonctionnement de la fonction d'élection EDF



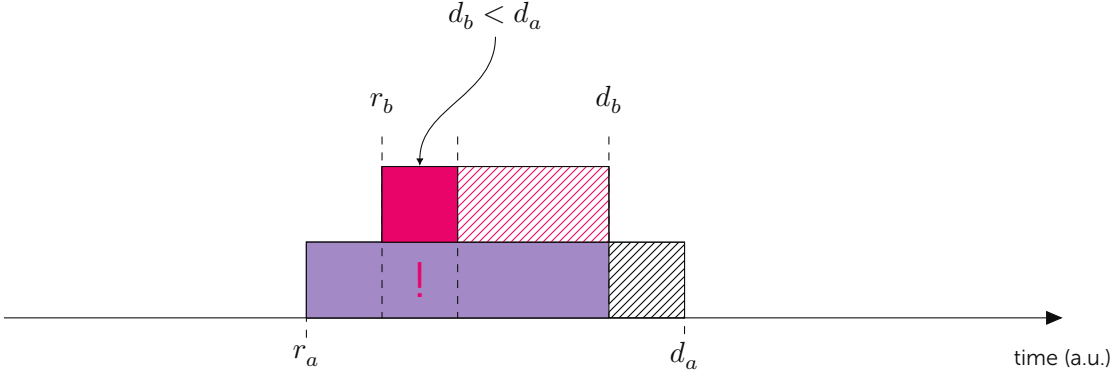
Prototype et fonctionnement de la fonction d'élection EDF



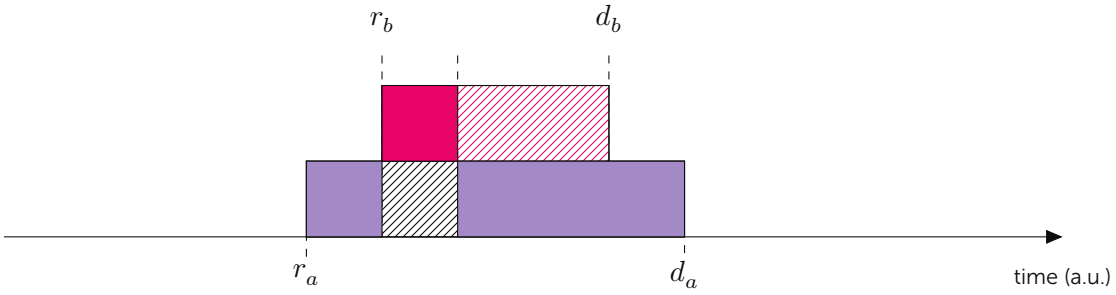
Prototype et fonctionnement de la fonction d'élection EDF



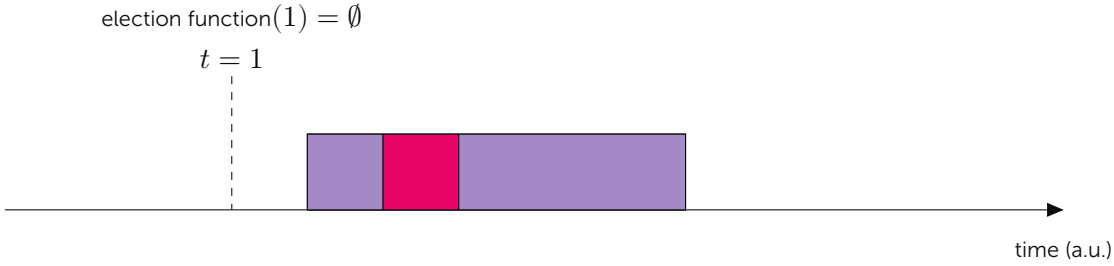
Prototype et fonctionnement de la fonction d'élection EDF



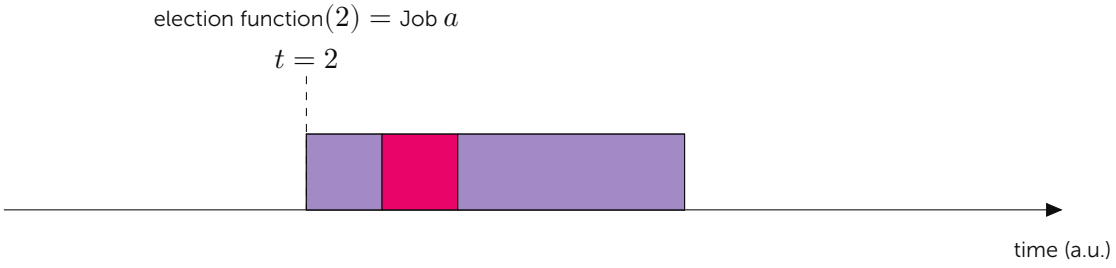
Prototype et fonctionnement de la fonction d'élection EDF



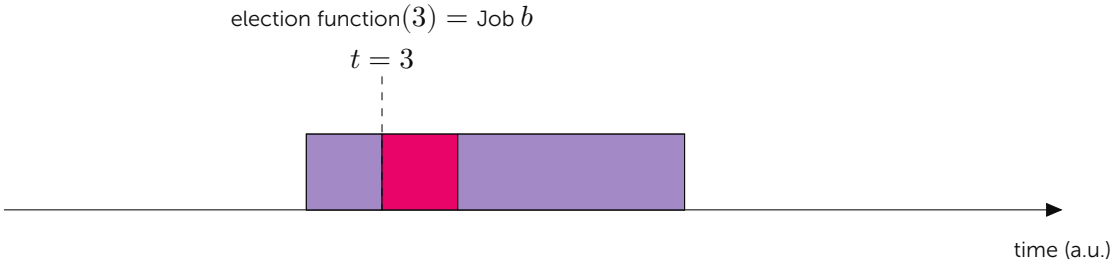
Prototype et fonctionnement de la fonction d'élection EDF



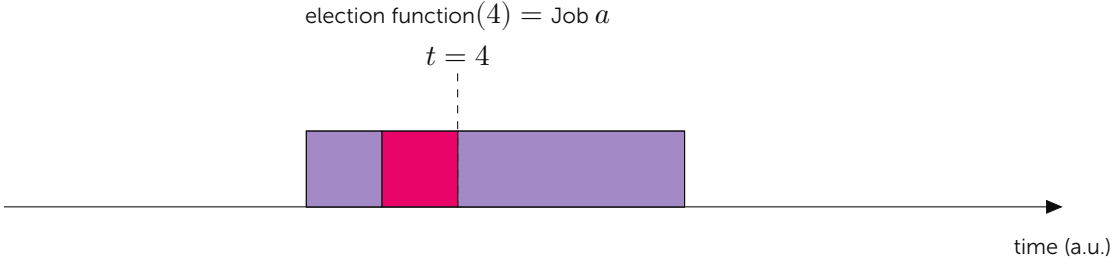
Prototype et fonctionnement de la fonction d'élection EDF



Prototype et fonctionnement de la fonction d'élection EDF



Prototype et fonctionnement de la fonction d'élection EDF



Première implémentation formellement vérifiée

d'un ordonnanceur Earliest Deadline First

ordonnançant des séquences de jobs arbitraires

Ordonnanceur Earliest Deadline First

Hypothèses & preuves

Quelles garanties sur l'ordonnanceur ?

Les ensembles *ordonnançables* de jobs sont ordonnancés de telle manière qu'aucun job ne dépasse son échéance

Propriété d'ordonnançabilité d'un ensemble de jobs

Soient deux instants distincts t, t', soit $\Gamma_{t,t'}$ l'ensemble des jobs à ordonnancer dans l'intervalle $[t, t']$. Si la somme des budgets c_j des jobs de cet ensemble est inférieure à $t' - t$, alors l'ensemble est *ordonnançable*.

Définition (Propriété d'ordonnançabilité)

$$\forall t, \forall t'.\ t < t' \implies (\sum_{j \in \Gamma_{t,t'}} c_j) \leqslant t' - t$$

Trois étapes pour atteindre la propriété de bon fonctionnement de l'ordonnanceur par raffinement

1ère étape : Modélisation de la politique d'ordonnancement Earliest Deadline First

Politique d'ordonnancement EDF

Soit j un job arbitraire et t un instant arbitraire, si le job j s'exécute à l'instant t, alors pour chaque autre job j' qui aurait pu s'exécuter à l'instant t, $d_j \leqslant d_{j'}$.

Appliquer la politique EDF à un ensemble de jobs jusqu'à un certain instant t est modélisé par la fonction :

$$\text{EdfPolicyUpTo } t$$

Propriété de bon fonctionnement de la politique EDF

$$\text{schedulable} \implies \forall j. \forall t.\ \text{EdfPolicyUpTo } t \implies \neg \text{overdue } j\ t$$

2nde étape : Conception d'une fonction mathématique d'élection

Implémentation d'une fonction d'élection (au sens mathématique) qui se comporte de la même manière que la procédure d'élection exécutable.

La prochaine étape du raffinement est de montrer que cette fonction d'élection implémente la politique EDF définie précédemment.

Fonction d'élection implémente la politique EDF

$$\forall t, \forall j, \forall s.\ \texttt{functional\_scheduler}(t) = (j, s) \implies \text{EdfPolicyUpTo}\ t.$$

De cette propriété découle la propriété de bon fonctionnement de la fonction d'élection.

3ème étape : preuve de bon fonctionnement de la procédure exécutable d'élection

Implémentation de la procédure d'élection exécutable, traductible en C, *reposant sur des primitives prédéfinies*.

La dernière étape du raffinement consiste à prouver que la procédure d'élection a le même comportement que la fonction d'élection.

Procédure d'élection se comporte comme la fonction d'élection

$$\forall t.$$
$$\{\ \mathit{env} = E \wedge s = \mathit{init}\ \}$$
$$(j, s') := \texttt{scheduler}\ (t)$$
$$\{\ \texttt{functional\_scheduler}\ (t) = (j, s')\ \}$$

De cette propriété découle la propriété de bon fonctionnement de la procédure d'élection exécutable.

Ordonnanceur Earliest Deadline First

Résumé de la contribution

Les points clés

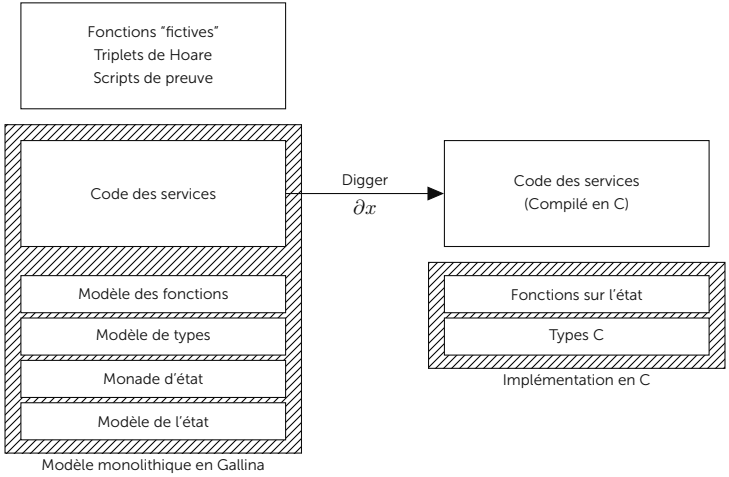
- Première implémentation formellement vérifiée d'un ordonnanceur Earliest Deadline First pour jobs arbitraires
- Démonstration de la pertinence du service de transfert de flot d'exécution
- Première utilisation du raffinement avec la méthodologie de Pip
- Première preuve en espace utilisateur avec la méthodologie Pip

Réflexions sur l’établissement de nouvelles preuves

Constat

- Besoin de montrer des propriétés non liées à l'isolation
- Code des services de Pip corrélé aux modèles d'isolation

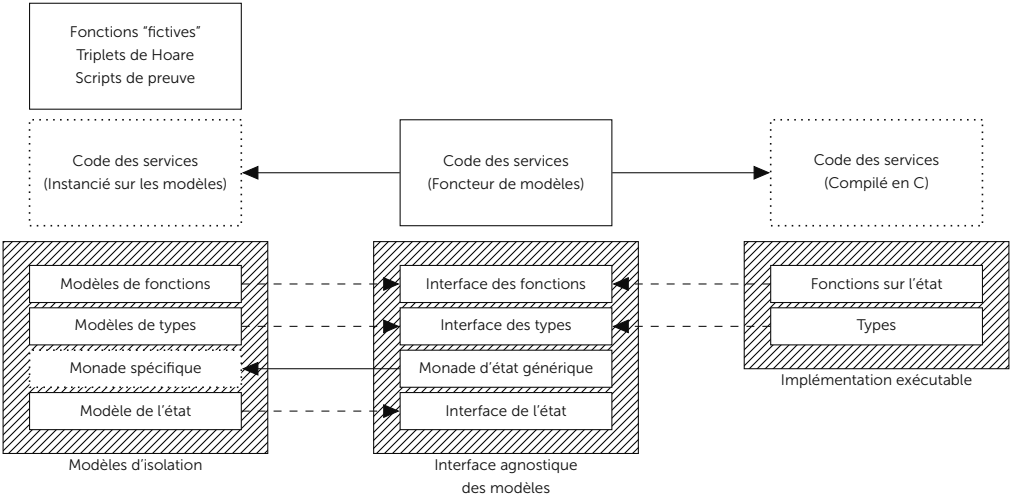
Architecture actuelle de Pip



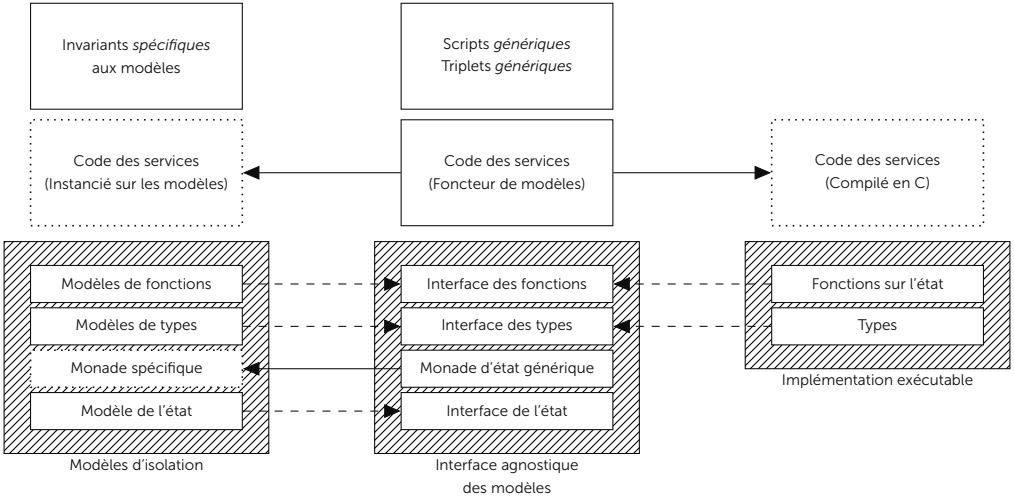
Réflexions sur l'établissement de nouvelles preuves

Proposition de nouvelle architecture

Code sous forme de foncteur de modèles



Code sous forme de foncteur de modèles



Réflexions sur l’établissement de nouvelles preuves

Perspectives liées à la nouvelle architecture

Bénéfices & *perspectives*

- Séparation nette entre le code des services et les modèles
- Permet de raisonner sur *d'autres modèles* que le modèle d'isolation
- … sans réécrire le code pour chaque nouveau modèle

Bénéfices & *perspectives*

- Séparation nette entre le code des services et les modèles
- Permet de raisonner sur *d'autres modèles* que le modèle d'isolation
- … sans réécrire le code pour chaque nouveau modèle

Ne permet cependant pas de concevoir des preuves *génériques* réduisant l'effort de preuve

Conclusion

Besoins de réponses aux problématiques de partage du temps CPU dans Pip

- Service de transfert de flot d'exécution dans Pip
  - transferts explicites, fautes et interruptions
  - au sein d'un unique service
  - preuve d'isolation réalisée
- Implémentation formellement vérifiée d'un ordonnanceur EDF
  - fonction d'élection formellement vérifiée
  - Exécution dans une partition de Pip
  - mais fonction d'élection non liée à Pip
- Preuve de concept d'architecture permettant de montrer d'autres propriétés

Des questions ?

Merci pour votre attention

Attributions

- Template LaTeX beamer adapté de minflat-beamer (CC BY-SA 4.0)
- Icônes d'artistes Freepik et NT Sookruay gratuites pour un usage personnel avec attribution