Université Kasdi Merbah d'Ouargla

Faculté des Nouvelles Technologies de l'Information et de la Communication

Département d'Informatique et des Technologies de l'Information

Mémoire en vue d'obtention d'un :

MASTER ACADEMIQUE

Domaine :Informatique

Spécialité : Informatique industrielle

par : DJERROUDI Linda

Thème :

Développement d'un driver de périphériques sous l'environnement RTLinux

Soutenu publiquement le : 05/06/2016

Devant le jury composé de :

Mr.	MAHDJOUB	Med Bachir	Université de Ouargla	**Président**
Mr.	BEKKARI	Fouad	Université de Ouargla	**Examinateur**
Mr.	HERROUZ	Abdelhakim	Université de Ouargla	**Rapporteur**

Année universitaire :2015/2016

« *Aucun homme n'est sorti de sa maison pour la recherche de la science sans qu'Allah ne lui facilite un chemin vers le paradis* »

Le prophète Mohammed SAW

Dédié à

ma mère et mon père dont le soutien m'a permis d'avancer

...

Remerciments

Je tiens premièrement à remercier le bon dieu de m'avoir accordé santé et consentement, ensuite je remercie mon encadreur Dr. Abd-Elhakim Herrouz pour la confiance qu'il m'a accordé dès le premier jour et qu'il a sans cesse renouvelé. Son intérêt profond pour la recherche de la connaissance et de la compréhension a été un vecteur d'envie d'aller toujours plus loin, je remercie également les membres du jury présidé par Mr Mahdjoub Med Bachir et examiné par Mr Bekkari Mahdjoub d'avoir accepté de corriger ce document.

je salut mes collègues du parcours universitaire, particulièrement ceux qui ont partagé leurs connaissances.

Finalement j'aimerais surtout remercier mes parents, mon frère pour son aide inconditionnelle, ma sœur et grand-mère, qui ont toujours été confiants de mes compétences et m'ont poussé sans cesse à me dépasser tout en me soutenant et surtout en me supportant.

Merci

Table des matières

Table des matières **v**

Table des ftgures **viii**

Liste des tableaux **ix**

Introduction générale **1**

1 Les systèmes temps réel **3**

1.1 Introduction 3
1.2 Définitions et concepts de base 3
1.2.1 Le temps réel 3
1.2.2 Les systèmes temps réel 4
1.2.3 Structure d'un système de contrôle 5
1.2.4 Application temps réel 6
1.2.5 Le déterminisme 6
1.2.6 La préemption 8
1.2.7 Niveaux des contraintes temporelles 8
1.3 Différences entre système d'exploitation temps réel, exécutif temps réel, application temps réel et système en temps réel 8
1.4 Les systèmes d'exploitation 9
1.4.1 Définition 9
1.4.2 Objectifs du système d'exploitation 9
1.4.3 Les systèmes multiutilisateur 9
1.5 L'évolution des systèmes d'exploitation 10
1.5.1 Le traitement par lots (batch processing) 10
1.5.2 Vers la multi-programmation 10
1.5.2.1 Le couplage de deux machines 11
1.5.2.2 Systèmes à entrées/sorties simultanées 11
1.5.3 Systèmes à temps partagé (multitâches) 13
1.6 Définition d'une tâche 13
1.7 le multitâche et ses concepts 14
1.7.1 Zone critique et exclusion mutuelle 15
1.7.1.1 zone critique 15
1.7.1.2 exclusion mutuelle 15
1.7.2 Ordonnanceur 15
1.7.3 Changement de contexte 16

1.7.4 Priorité des tâches 16
1.7.5 Communication entre tâches 17
1.7.5.1 Problématique 17
1.8 Conclusion 18

2 Les systèmes d'exploitation temps-réel **19**
2.1 Introduction 19
2.2 Les systèmes d'exploitation temps réel 19
2.3 Les techniques des RTOS 20
2.3.1 Utilisation du temps concret 20
2.3.2 L'ordonnancement 21
2.3.3 L'ordonnancement préemptif grâce au RTM . . .22
2.3.3.1 Caractérisation d'une tâche 22
2.3.3.2 Test d'ordonnançabilité 22
2.3.3.3 Les appels systèmes 23
2.3.4 Les normes POSIX 23
2.4 Caractéristiques de l'offre des SE temps réel 24
2.4.1 Fiabilité 25
2.4.2 prédictibilité 25
2.4.3 Performance 25
2.4.4 Flexibilité 25
2.5 Principaux SE temps réel 26
2.6 Conclusion 28

3 Linux et le temps réel **29**
3.1 Introduction 29
3.2 Les qualités de linux 29
3.3 Architecture du noyau Linux 31
3.4 Un Linux temps réel 31
3.5 Traçage du noyau Linux 36
3.6 Conclusion 36

4 Mise en oeuvre **37**
4.1 Introduction 37
4.2 Traçage de RTLinux 37
4.2.1 L'outil de trace : Cyclictest 38
4.3 L'installation de RTLinux 38
4.4 Développement sous RTLinux 40
4.5 Structure d'un module 41
4.6 Test du port parallèle sous RTLinux 41
4.6.1 Utilisation du port parallèle 41
4.6.2 Le code du port parallèle 42
4.6.2.1 **lpt−irq.C**[12] 42
4.6.2.2 **rt−irq−gen.C**[12] 42
4.7 Conclusion 43

Conclusion générale **44**

Bibliographie **46**

A Liste des Abréviations **49**

Table des figures

1.1 composantes d'un système temps réel 5
1.2 Le batch processing 10
1.3 Couplage de deux machines 11
1.4 Systèmes à entrées/sorties simultanées 12
1.5 les systèmes multitâches 13
1.6 les différents états d'une tâche 13

2.1 vue des composants d'un RTOS : le noyau et autres composants d'un système enfouis 20
2.2 L'execution des processus en pseudo-parallélisme 21
2.3 Différents services fournis par le noyau 24

3.1 insertion de module sur le noyau linux 32
3.2 Représentation de l'architecture de fonctionnement du temps réel sur un système d'exploitation basé sur l'architecture micro-noyau 33
3.3 Architectures basées sur Adeos utilisées par RTAI et Xenomai 34
3.4 Représentation de l'architecture de fonctionnement du temps réel sur un système d'exploitation GNU/Linux 35

4.1 Trace de RTLinux 38
4.2 Capture d'écran qui présente le Dual boot de Linux et RTLinux 40
4.3 Capture d'écran du code C du port lpt—irq 42
4.4 Captures d'écran du code C du port rt—irq—gen 42

Liste des tableaux

1.1 les transitions d'une tâche 14

Introduction générale

En contexte industriel il y a souvent besoin d'un comportement temps réel tels que le pilotage matériel (moteurs, ...), l'acquisition de données (capteurs, ...), piles protocolaires (GPRS, VoIP, ...) ... etc. D'autant plus, les développeurs et utilisateurs de systèmes temps réel sont confrontés à des besoins spécifiques et se retrouvent face aux dilemmes suivants :

- Choisir des technologies propriétaires qui ne s'adaptent pas toujours à leurs exigences ;
- Ou opter pour des systèmes temps réel ne bénéficiant pas d'une standardisation suffisante et parvenant difficilement à suivre le rythme des évolutions technologiques.

Cependant, il existe une troisième voie : l'**open source**, avec le système d'exploitation Linux en tête, qui présente de nombreux avantages grâce notamment à sa robustesse et sa communauté plus que généreuse.

En effet, l'open source assure la redistribution sans royalties, la disponibilité du code source et la possibilité de réaliser un développement dérivé de ce code source.
D'autre part, Linux est stable, efficace et les principales architectures sont supportées : **x86**, **Power PC**, **MIPS**,. . . En plus, il y a un nombre de plus en plus important de logiciels disponibles. En cas de problèmes, l'aide rapide est toujours assurée par la communauté Internet des développements Linux.
C'est pourquoi, la fiabilité légendaire de Linux en fait un candidat idéal pour les systèmes temps réel.

Malheureusement, Linux n'est pas nativement un système temps réel. Le noyau Linux fut en effet conçu dans le but d'en faire un système généraliste donc basé sur la notion de temps partagé et non de temps réel.Et comme tout problème en informatique celui-ci a poussé les informaticiens de toute catégorie amateurs et concepteurs de systèmes à chercher les méthodes et solutions possibles.

Ce travail propose des solutions techniques pour l'adaptation et la mise en oeuvre de Linux dans l'univers des systèmes temps réel et ce, afin d'accéder véritablement au développement de solutions industrielles complexes.

Ce mémoire est organisé en quatre (04) chapitres. Nous consacrerons le chapitre 1 aux concepts généraux sur les systèmes d'exploitation généralistes et temps réel servant de point de départ pour le chapitre 2 qui abordera les notions des Systèmes d'exploitation Temps Réel distinctement , ainsi qu'une énumération de quelques RTOS (propriétaires et ouverts) les plus populaires dans le dommaine industriel.

Le chapitre 3 portera sur la présentation du système d'exploitation Linux et son implémentation du temps réel à travers des différentes architectures et les modifications en ligne (PREEMPT RT).
Le quatrième chapitre quant à lui, met en avant la mise en oeuvre de RTLinux pour le développement des applications temps réel.

Enfin, en guise de conclusion (4.7), nous indiquerons quelques remarques sur ce travail en concluant par un bilan et nous exposerons les perspectives pour de futurs travaux.

Chapitre 1

Les systèmes temps réel

1.1 Introduction

Nous allons procéder , dans ce chapitre ,à la présentation et définition plus ou moins détaillées des concepts utilisés dans les systèmes d'éxploitation génériques et temps réel ainsi qu'un petit historique des SE. Pour cela, il introduit des définitions, et aborde la notion de multitâche et les principes qui en découlent. Ce chapitre sera ensuite complété par son successeur qui détaillera davantage les notions du temps réel.

1.2 Définitions et concepts de base

Dans cette section, nous présenterons et définirons différents concepts qui seront utilisés dans ce mémoire.

1.2.1 Le temps réel

Le temps réel est un concept un peu vague et chacun y va de sa définition. On pourrait le définir comme :

1. la définition canonique d'un système temps réel de [7]. se porte comme suit :

   "Un système en temps réel est un système dans lequel l'exactitude des calculs dépend non seulement de l'exactitude logique du calcul mais également du temps où le résultat est produit. Si les contraintes de synchronisation du système ne sont pas rencontrées, celui-ci sera alors non-fonctionnel."

2. D'autres ont ajouté :
   "Un système est dit Temps Réel lorsque l'information après acquisition et traitement est encore pertinente". Cela veut dire que dans le cas d'une information arrivant de façon périodique, le temps

d'acquisition/traitement doit être inférieur à la période de rafraîchissement de cette information. Pour cela, un **RTOS** doit être déterministe et préemptif pour donner la main durant le prochain tick à la tâche de plus forte priorité prête. On peut en théorie dans ce cas prédire l'évolution du système Temps Réel.

1.2.2 Les systèmes temps réel

Un système temps réel est un système informatique prenant en compte les contraintes temporelles. Ces contraintes sont alors aussi importantes que l'exactitude des résultats obtenus, et un résultat sera invalidé s'il n'est pas obtenu dans les délais convenables. On dit alors qu'un système temps réel doit respecter un déterminisme logique (exactitude du résultat) et temporel.

Un retard = le fait de rater une échéance =erreur du système.

Rappelons que lors de l'exécution d'un programme, celui-ci peut être interrompu par un évènement extérieur. Il est de la responsabilité du système d'exploitation de minimiser le temps entre la prise en compte de l'évènement et l'exécution du programme chargé de traiter celui-ci. C'est cette durée qu'on appelle temps de ***latence***.
Quant à l'***échéance*** c'est la date à laquelle la tâche doit avoir terminé son exécution.

On distingue deux types de systèmes temps réel selon l'importance des contraintes temporelles à remplir :

Les systèmes temps réel à contraintes souples (SRT), pour lesquels les évènements traités en dehors des délais ou simplement perdus n'ont pas de conséquence catastrophique sur la bonne marche du système. Par exemple, les systèmes multimédia – notamment la diffusion en flux – pour lesquels il est possible de perdre quelques secondes de vidéo sans que le fonctionnement général du système ne soit mis en péril ;

Les systèmes temps réel à contraintes strictes (HRT), qui correspondent à des systèmes dont aucune échéance ne doit être dépassée, devants répondre à des sollicitations ou évènements, internes ou externes, dans un temps maximum nécessitant alors un déterminisme temporel fort. Par exemple, les systèmes de contrôle des centrales nucléaires ou encore les systèmes embarqués en aéronautique entrent dans cette catégorie.

Un système temps réel est, en général constitué de deux parties :

— **Le système contrôlé** : processus physique qui évolue avec le temps, appelé aussi procédé ou partie opérative ;
— **Le système contrôlant** :appelé aussi "partie commande" .c'est un système informatique qui prend régulièrement connaissance du système contrôlé par le biais de capteurs, calcule la prochaine action à réaliser sur la base des dernières mesures puis applique une consigne au processus commandé par le biais d'actionneurs [23](voir figure 1.1).

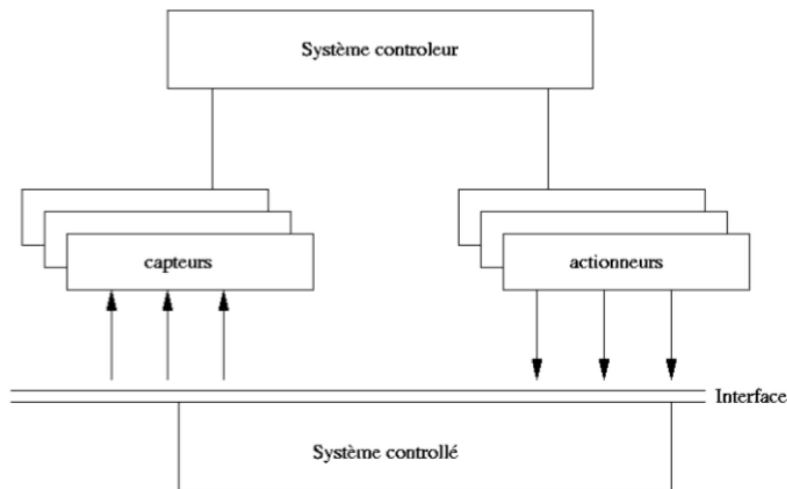


Figure 1.1 – composantes d'un système temps réel

1.2.3 Structure d'un système de contrôle

Le système de contrôle consiste en une structure matérielle munie d'un ensemble de logiciels permettant d'agir sur le système contrôlé. La structure matérielle correspond à l'ensemble des ressources physiques (processeurs, cartes d'entrée/sortie, ...) nécessaires au bon fonctionnement du système. L'architecture matérielle peut être :

—monoprocesseur : tous les programmes s'exécutent sur un seul processeur en parallélisme apparent ;
—multiprocesseur : les programmes sont répartis sur plusieurs processeurs partageant une mémoire commune ;
—réparti : les programmes sont répartis sur plusieurs processeurs dépourvus de mémoire commune et d'horloge commune.Ils sont reliés par un médium de communication par lequel ils peuvent communiquer par échange de messages.

La partie logicielle du système de contrôle comprend :

- D'une part, un système d'exploitation appelé exécutif temps réel chargé de fournir un ensemble de services que les programmes de l'application peuvent utiliser durant leur exécution. Il doit en particulier :
  - —adopter une stratégie pour partager le temps processeur (notamment dans un système multitâches) entre les différentes activités en attente d'exécution en favorisant celles qui ont les délais critiques les plus proches ;
  - —gérer l'accès aux ressources partagées ;
  - —offrir des mécanismes de synchronisation et de communication entre les activités du système de contrôle.
- D'autre part, afin de garantir un fonctionnement correct du procédé, des programmes sont implantés par l'utilisateur et traduisent les fonctions que doit réaliser le système de contrôle pour la prise en compte de l'état du procédé et sa commande. Ces programmes se présentent sous la forme d'un ensemble d'unités d'exécution appelées **tâches**.

1.2.4 Application temps réel

Une application temps réel est une application pour laquelle le respect de contraintes temporelles lors d'un traitement est aussi important que le résultat du traitement. Elle repose généralement sur un déterminisme à deux critères : logique, c'est-à-dire que les mêmes données traitées doivent donner le même résultat, et temporel, c'est à dire qu'une tâche donnée doit absolument être réalisée dans les délais impartis, autrement dit respecter l'échéance.

1.2.5 Le déterminisme

On a cité plus haut la notion du déterminisme qui est crucial dans un système informatique quelconque mais dont l'importance prend de l'ampleur dans un système temps réel.
Le déterminisme d'un système est la propriété qui nous permet de prédire son comportement dans toutes ses situations d'exécution. De façon générale, cette propriété est extrêmement importante dans tous les secteurs de l'informatique. Si le système se met à dérailler de façon imprévisible, il n'est pas déterministe. Prenons exemple d'un guichet automatique (qui n'est pas un système temps réel, mais est un système informatique tout de même) doit être déterministe. Son comportement doit avoir été prévu afin de ré- pondre adéquatement à chaque situation qui se présente. Un tel système informatique doit pouvoir réagir de façon adéquate lorsque l'utilisateur dérobe à la séquence normale des opérations , à savoir :

—Entrer la carte ;

—Entrer son NIP ;

—Choisir une opération. . .

et le concepteur doit même pouvoir dire dans quel état doit se trouver le système après une séquence donnée d'opérations quelconques.
Dans un système en temps réel, le déterminisme inclut la capacité de prédire le temps d'exécution maximal des tâches impliquées [6]. Il faut savoir à l'avance que telle ou telle séquence d'opération se fera à l'intérieur de tel ou tel délai.
Prenons l'exemple d'un réservoir hermétique muni d'un capteur de pression et d'une soupape de sécurité. Le délai entre la lecture d'une pression trop élevée et l'ouverture d'une soupape de sécurité doit être connu pour savoir si les catastrophes seront toujours évitées. Ici, tous les processus requis, tant matériels que logiciels, doivent avoir été minutés soigneusement avant la mise en marche du système, et dans plusieurs cas de figure.
Il est souvent impossible de connaître les délais exacts ; il est important dans ce cas de disposer d'une borne supérieure estimée sur le délai, et de savoir si ce temps maximal de réaction est inférieur au temps que prend la catastrophe pour se produire.
On a donc les relations suivantes entre les délais impliqués, dans le cas des traitements relatifs à une situation donnée :

- **t** : délai de traitement exact (la plupart du temps inconnu et variable)
- **Test** : délai de traitement maximal estimé
- **T** : temps avant que ne se produise une catastrophe (explosion du bassin ?). C'est la contrainte de temps.

Pour affirmer qu'un système est déterministe (qu'il respecte ses délais), il faut qu'en tout temps :

t < Test et **Test < T**

C'est la seule manière de s'assurer que **t < T** et de garantir que les situations critiques seront prises en charge à temps. Ces calculs sont souvent difficiles à réaliser ; les résultats obtenus dépendent non seulement des algorithmes utilisés dans les traitements, mais de pratiquement tout ce qui a contribué ou contribue au système :

- les algorithmes utilisés ;
- le langage de programmation ;
- le compilateur et l'optimisation du code ;
- le temps d'exécution des services du système d'exploitation ;
- les délais imposés par les mécanismes d'entrées/sorties ;
- la rapidité d'accès à la mémoire principale ;
- la rapidité du processeur ;
- etc.

1.2.6 La préemption

Dans le domaine informatique, la préemption est l'acte d'interrompre temporairement une tâche en cours d'exécution sans nécessiter sa coopération, dans le but de la laisser terminer son exécution plus tard. Ce genre d'interruption est en général effectué pour donner du temps d'exécution à une tâche de plus haute priorité et sera effectué par l'ordonnanceur du système s'il s'agit d'un ordonnanceur préemptif. On appelle ceci un **changement de contexte**.

1.2.7 Niveaux des contraintes temporelles

Un système temps réel se compose de plusieurs tâches (défini ultérieurement). Pourtant, toutes ces tâches ne relèvent pas du temps réel. On peut distinguer trois niveaux de contraintes temporelles :

•**Sévère** : les tâches temps réel critiques doivent absolument respecter les échéances prévues ;
•**Ferme** : les tâches temps réel non critiques peuvent répondre avec un certain retard sachant que la cohérence du système en sera dégradée ;
•**Souple** : les autres tâches, qui n'ont pas de contrainte de temps, n'auront qu'à répondre au mieux.

Dans un avion, l'altimètre sera une tache très critique, tandis que la jauge sera seulement critique et que la gestion du bar sera une tache sans contrainte de temps réel.

1.3 Différences entre système d'exploitation temps réel, exécutif temps réel, application temps réel et système en temps réel

• **Système d'exploitation temps réel** : Système d'exploitation complet intégrant les services nécessaires afin de concevoir et exécuter des applications temps réel, par exemple : *QNX*, *RTLinux*, *LynxOS* et *WindowsCE* (voir 2.5 pour plus de détails).

• **Exécutif temps réel** : Basé sur un système d'exploitation quelconque, ajoute les services nécessaires pour concevoir et exécuter des applications temps réel par exemples : *VxWorks* et *VRTX* (voir 2.5).

• **Application en temps réel** : Application logicielle satisfaisant à plusieurs des caractéristiques énumérées plus haut,par exemple : Contrôleur d'usine et Programme d'échantillonnage.

• **Système en temps réel** : Expression pour décrire l'ensemble matériel et logiciel requis pour l'exécution d'une application temps réel (inclus l'application, le système d'exploitation (et dans certain cas l'exécutif

temps réel), le réseau, les périphériques, les robots, la chaîne de production, les postes…),par exemple une chaîne de production automatisée

1.4 Les systèmes d'exploitation

1.4.1 Définition

Un système d'exploitation est un ensemble de programmes assurant la gestion pour le partage des ressources attachées à la CPU (dont la CPU), le cœur de ces programmes est le **noyau** qui regroupe les fonctionnalités les plus importantes (noyau et système d'exploitation se retrouvent souvent confondus)

1.4.2 Objectifs du système d'exploitation

• interagir avec le matériel et permettre l'utilisation des composants de bas niveau ;

• fournir l'environnement d'exécution pour les applications utilisateur ;

• établir une couche de « protection » entre l'utilisateur et le matériel (ce n'est pas le cas de tous les SE) comme le mode utilisateur et noyau dans les systèmes de type UNIX.

1.4.3 Les systèmes multiutilisateur

la notion de multiutilisateur fait référence au fait que plusieurs applications appartiennent à des utilisateurs différents.Ces applications peuvent être exécutées de façon concurrente et indépendante :

- **Concurrence** : partage de l'accès aux ressources
- **Indépendance** : chaque application n'a pas besoin de savoir ce que les autres sont en train de faire .

le SE doit minimiser les inconvénients liés à ces contraintes, en particulier en termes de temps de réponse ,il doit aussi être capable de fournir un mécanisme d'authentification des utilisateurs, un mécanisme de protection vis-à-vis de programmes buggés éventuellement gênants, un mécanisme de protection contre les programmes malveillants et un mécanisme d'accounting pour assurer un partage équitable des ressources (mode interactif).

1.5 L'évolution des systèmes d'exploitation

1.5.1 Le traitement par lots (batch processing)

Les machines avant 1956 n'avaient pas de système d'exploitation, en effet c'étaient des machines géantes à programme enregistré sur ruban perforé, par la suite traduits en binaire et placés en mémoire par un mécanisme appelé « ordres initiaux », lui-même enregistré sur une mémoire externe en lecture seule.
Plus tard furent introduits les sous-programmes, conservés dans une bibliothèque sous forme de rubans ; ceux-ci devaient être physiquement copiés avant exécution, sur le ruban contenant le programme principal. Les ordres initiaux devaient donc assurer ce qu'on appelle aujourd'hui l'**édition de liens**, c'est-à-dire fixer les adresses d'appel et de retour des sous-programmes pour garantir leur exécution correcte. Mais l'exploitation de l'ordinateur restait sous contrôle manuel.
Compte-tenu de l'important investissement que représentait à l'époque l'achat d'un ordinateur, il n'est pas étonnant que l'on ait recherché les moyens de rendre son exploitation plus efficace. Les programmes des utilisateurs furent ainsi regroupés en lots (en anglais batch), pour les traiter en série, sans transitions. Une nouvelle fonction apparut, celle d'**opérateur**, un technicien chargé de préparer les lots d'entrée à partir des programmes fournis par les utilisateurs, de surveiller l'exécution des travaux, et de distribuer les résultats de sortie. L'enchaînement des travaux était réalisé par un programme appelé moniteur, ou système de traitement par lots, qui fut la première forme de système d'exploitation, voir la figure 1.2 .

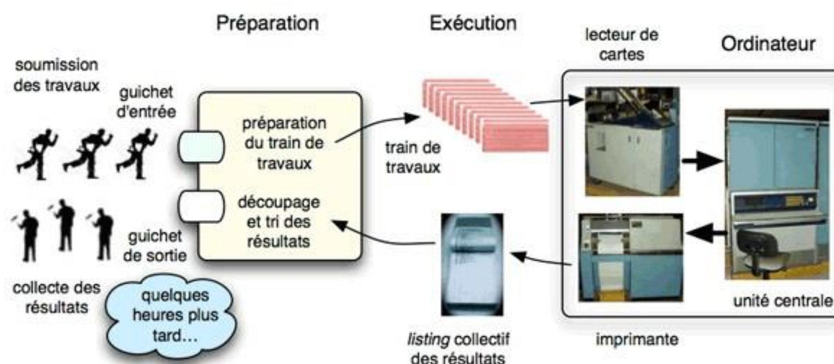


Figure 1.2 – Le batch processing [24]

1.5.2 Vers la multi-programmation

Le mode d'exploitation précédant offrait à l'utilisateur une maitrise totale sur l'exécution du travail (programme), par ailleurs il faisait objet de sous-exploitation de l'unité centrale (le processeur) :
1. en cas de bascul vers l'utilisation d'un dispositif d'entrée/sortie (im- pression d'une ligne par ex);

2. pendant le temps de lecture d'une carte.

Dans ces deux cas l'unité centrale reste oisive [20] Plusieurs modifications furent alors apportées pour remédier à ce problème :

1.5.2.1 Le couplage de deux machines

Pour augmenter le taux d'utilisation de l'unité centrale, on eut l'idée de remplacer le lecteur de cartes et l'imprimante par des bandes magnétiques, organes beaucoup plus rapides. Un calculateur, auxiliaire, beaucoup plus simple que le calculateur principal, est utilisé pour constituer une bande d'entrée à partir des cartes soumises par les utilisateurs et pour imprimer le contenu de la bande de sortie.La fonction d'opérateur évolue pour prendre en compte ces changements.

La figure 1.3 schématise ce mode d'exploitation (la phase de préparation est inchangée par rapport à la figure 1.2).

Notons que la recherche de l'efficacité conduit à augmenter le nombre de travaux dans un lot, ce qui accroît encore le temps d'attente des utilisateurs.

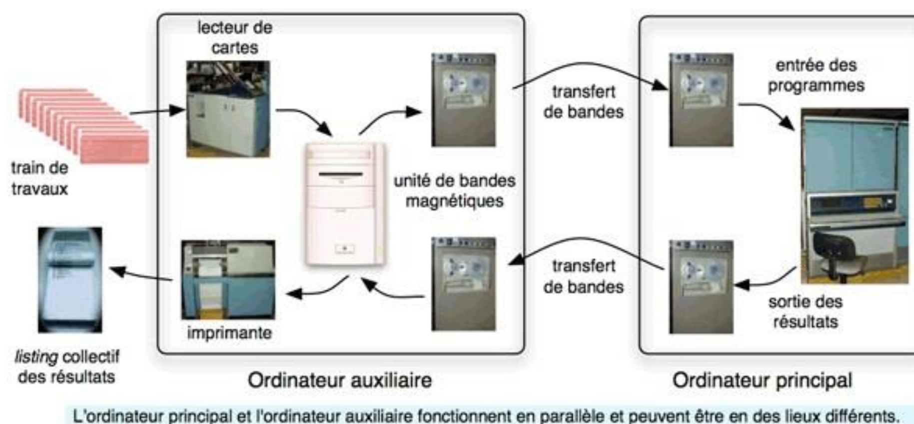


Figure 1.3 – Couplage de deux machines [24]

1.5.2.2 Systèmes à entrées/sorties simultanées

Deux évolutions matérielles ont permis de simplifier le schéma de la figure 1.3 et d'améliorer encore son efficacité :

- l'invention des canaux d'entrée-sortie, permettant d'exécuter en parallèle des entrées-sorties et des opérations de calcul,
- la généralisation des disques magnétiques, organes d'accès rapide, pour le stockage des données.

Le système d'exploitation peut alors coordonner trois activités simultanées, la lecture des travaux à exécuter, l'exécution proprement dite et l'impression des résultats.Ces trois activités communiquent par l'intermédiaire de zones de mémoire intermédiaire, ou tampons, stockées sur disque pour avoir une taille suffisante. Ce dispositif, schématisé sur la figure 1.4 , portait le nom de SPOOL (Simultaneous Peripherals Operation On Line). Il est toujours utilisé aujourd'hui pour gérer les impressions sur les ordinateurs personnels.[24]

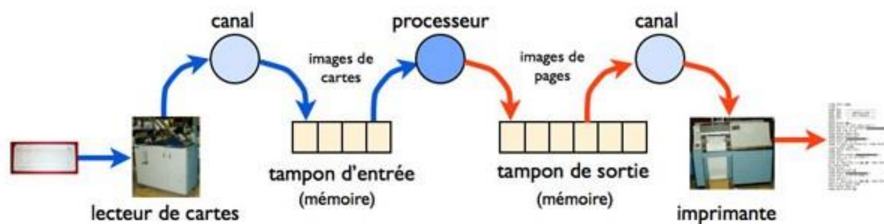


Figure 1.4 – Systèmes à entrées/sorties simultanées [24]

Dans les premiers systèmes multiprogrammés, la mémoire était divisée en zones de taille fixe, dans lesquelles étaient chargés les programmes des utilisateurs. Plus tard, les limites de ces zones, en nombre devenu variable, furent ajustables dynamiquement grâce à l'invention des registres de base, ce qui permit une meilleure utilisation de la mémoire [24].
Les programmes sont donc exécutés cycliquement par le SE qui leur alloue les ressources nécessaires (disque, mémoire, fichier,...) pendant leur tranche de temps d'exécution.

Jusque-là, un seul programme utilisateur était chargé en mémoire à un instant donné mais l'introduction des disques magnétiques a ultérieurement induit la capacité de chargement d'un ou plusieurs programmes en mémoire et donc d'économiser le temps perdu lors de la lecture du code directement à partir des cartes, de plus , cette évolution a permis de lancer l'exécution d'un programme utilisateur prêt à s'exécuter quand le programme utilisateur est en attente d'entrée-sortie dans les canaux d'entrées-sorties, donc favoriser le taux d'utilisation de l'unité centrale davantage.
plus tard les concepteurs ont réfléchi à faire résider plusieurs programmes en mémoire et ont conçu quelques aménagements matériels et logiciels dans le but d'exécuter tous ces programmes sans pour autant affecter la rapidité/temps d'exécution aux yeux de l'utilisateur , c'est les premiers pas vers la multiprogrammation.[20]

Dans les premiers systèmes multiprogrammés, la mémoire était divisée en zones de taille fixe, dans lesquelles étaient chargés les programmes des utilisateurs. Plus tard, les limites de ces zones, en nombre devenu variable, furent ajustables dynamiquement grâce à l'invention des registres de base, ce qui permit une meilleure utilisation de la mémoire.[24].

Les programmes sont donc exécutés cycliquement par le SE qui leur alloue les ressources nécessaires (disque, mémoire, fichier,...) pendant leur tranche de temps d'exécution.

1.5.3 Systèmes à temps partagé (multitâches)

Il s'agit d'une amélioration de la multiprogrammation orientée vers le transactionnel. Un tel système organise ses tables d'utilisateurs sous forme de files d'attente (figure). L'objectif majeur est de connecter des utilisateurs directement sur la machine et donc d'optimiser les temps d'attente du SE (un humain étant des millions de fois plus lent que la machine sur ses temps de réponse cadencés à la microseconde), chaque tâche est prise en charge comme le schématise la figure1.5

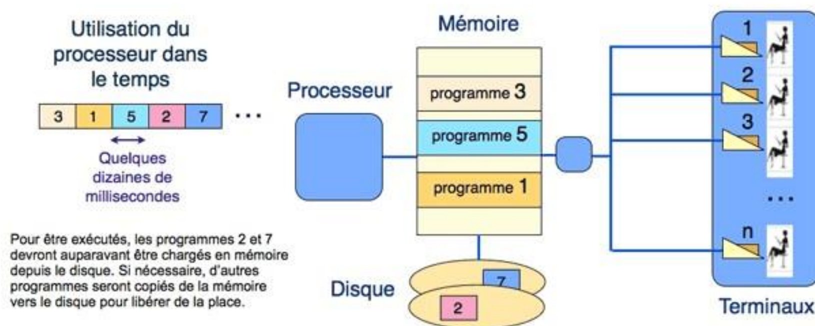


Figure 1.5 – les systèmes multitâches [24]

1.6 Définition d'une tâche

Une tâche est un programme qui s'exécute comme s'il était le seul à utiliser le CPU . Il possède une priorité, du code à effectuer, une partie de la mémoire et une zone de pile.
Une tâche peut être dans différents états comme le montre le schéma de la figure 1.6.

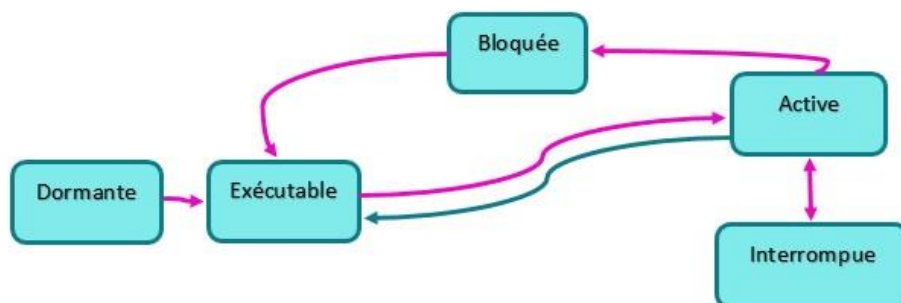


Figure 1.6 – les différents états d'une tâche

1.Dormante : La tâche est en mémoire mais n'est pas considéré par l'Ordonnanceur,

2. Exécutable : La tâche est prête à être exécutée mais n'a pas le CPU, 3.Active : La tâche est exécutée par le CPU,

4.Bloquée : La tâche est en attente d'un signal ou d'une ressource pour poursuivre,

5.Interrompue : La tâche est Suspendue par une interruption.

Le tableau 1.1 montre dans quel cas une tâche peut passer d'un état à un autre :

Etat d'origine	**Nouvel Etat**	**Evènement provoquant la transition**
Dormante	Exécutable	La tâche est activée
Exécutable	Active	L'Ordonnanceur donne le CPU à la tâche
Active	Bloquée	La tâche est en attente d'une ressource(bloquée par un sémaphore)
Bloquée	Exécutable	La ressource dont la tâche était en attente est libérée
Active	Exécutable	L'Ordonnanceur donne le CPU à une autre tâche (préemption)
Active	Interrompue	Une interruption stoppe la tâche
Interrompue	Active	Le traitement de l'interruption est terminé

Table 1.1 – les transitions d'une tâche

1.7 le multitâche et ses concepts

Un SE multitâches permet l'exécution simultanée de plusieurs tâches. Généralement, pour un système monoprocesseur (CPU unique avec exécution séquentielle des instructions), l'exécution simultanée est simulée : l'utilisateur a l'impression que chaque tâche possède exclusivement le CPU, mais c'est l'ordonnanceur qui répartit le CPU sur les tâches, donnant à chacune la chance d'exploiter le CPU pendant une **tranche de temps** minime et fixe.
Le processus de conception pour les applications temps réel implique de diviser le travail à effectuer entre différentes tâches chacune responsables d'une partie du problème.

1.7.1 Zone critique et exclusion mutuelle

1.7.1.1 zone critique

Une zone critique peut être une structure de données, une ressource partagée, une section de code au timing délicat, une zone de mémoire. Une zone critique ne peut être interrompue une fois commencée. Il faut donc désactiver les interruptions à l'entrée de la section critique et les réactiver à la sortie. Une seule tâche a le droit d'accéder à une zone critique. Elle doit être protégée pour que les autres tâches ne puissent modifier les données ou l'état du matériel. La technique des sémaphores que nous verrons plus loin doit donc être utilisée.

1.7.1.2 exclusion mutuelle

Lorsqu'une ressource n'est pas partageable entre plusieurs tâches, un mécanisme doit en assurer l'accès exclusif. Un tel mécanisme est appelé exclusion mutuelle : lorsqu'une tâche utilise la ressource, aucune autre tâche ne peut l'utiliser.
Une imprimante peut être utilisée par plusieurs processus (par exemple, par Word, par Excel, et d'autres logiciels) mais son accès est exclusif. Sinon, on pourrait avoir un mélange de fichiers si plusieurs processus se mettent à imprimer en même temps. L'exclusion mutuelle peut être réalisée de différentes façons :

— Le verrouillage matériel est réalisé à l'aide d'un drapeau binaire qui indique si la ressource est libre ou utilisée. Chaque tâche va tester le drapeau avant d'utiliser la ressource, si elle est déjà utilisée par une autre tâche, elle va se mettre en attente, verrouillage du CPU permet d'éviter l'accès concurrent à une ressource non partageable. La distribution du CPU d'une tâche à l'autre est interdite pendant cet accès.
— Les services du noyau tels que les sémaphores et les mutex seront développés plus loin.

1.7.2 Ordonnanceur

L'Ordonnanceur est le composant du noyau responsable de la distribution du temps CPU entre les différentes tâches. Il est basé sur le principe de la priorité : chaque tâche possède une priorité dépendant de son importance (plus une tâche est importante, plus sa priorité est élevée). Il attribue le processeur à la tâche exécutable (non dormante et non bloquée) de plus haute priorité. Lorsque qu'une tâche T2 plus prioritaire que la tâche active T1 passe de l'état bloqué à l'état exécutable, T1 est suspendue et l'Ordonnanceur attribue le processeur à T2. C'est le principe de la préemption. La préemption permet un temps de réponse des tâches optimum.

1.7.3 Changement de contexte

Quand l'ordonnanceur doit transférer l'usage du processeur d'une tâche à l'autre, il opère un changement de contexte. Le contexte représente l'état du processeur à un moment donné.
La tâche suspendue doit pouvoir continuer son exécution sans être affectée, la première opération à effectuer est donc la sauvegarde de l'état du processeur au moment de la suspension. Le noyau possède pour chaque tâche non dormante un espace mémoire réservé à cet effet appelé task control block (TCB)[21]
Les données temporaires utilisées par la tâche suspendue doivent être préservées lors des opérations de la nouvelle tâche active. Ces données sont organisées sous forme de pile contenant le contexte des appels de sous routines en cours (adresse de retour, valeur des registres) ainsi que les paramètres et les variables temporaires de ses sous-routines.
Une tâche peut être suspendue à tout moment et l'usage du processeur transféré vers une autre tâche susceptible d'appeler des sous-routines et d'allouer des variables temporaires,chaque tâche possède sa propre pile. Le temps requis pour un changement de contexte est déterminé par le nombre de registres à sauvegarder et à restaurer par le CPU.

1.7.4 Priorité des tâches

chaque tâche est attribuée une priorité est fonction de son caractère critique. Plus une tâche est importante, plus sa priorité est élevée. Il existe deux types de priorité de tâches :

— **Priorité statique** : la priorité de chaque tâche n'évolue pas durant l'exécution. Chaque tâche reçoit une priorité fixe lors de sa création. Toutes les tâches et leurs contraintes temporelles sont connues à la compilation.

— **Priorité dynamique** : La priorité de chaque tâche peut évoluer durant l'exécution. Cette caractéristique des RTOS permet d'éviter l'inversion de priorité.

L'inversion de priorité est un problème qui survient lorsqu'une tâche est suspendue dans l'attente d'une ressource contrôlée par une tâche moins prioritaire. Par exemple, s'il existe deux tâches **T1** et **T2** avec **T1** plus prioritaire que **T2**. **T1** attend qu'un événement se produise dans **T2** (par exemple la libération d'un sémaphore). Dans ce cas, la priorité effective de **T1** est réduite à celle de **T2** car elle est en attente d'une ressource détenue par **T2**.
Pour résoudre ce problème, il suffit d'élever temporairement la priorité de **T2** en la rendant égale à celle de **T1** pendant le laps de temps qu'elle utilise la ressource en ensuite de restaurer sa priorité normale.

1.7.5 Communication entre tâches

1.7.5.1 Problématique

Le transfert correct de données entre tâches dans un programme temps-réel soulève des problèmes plus importants que dans le cas d'un programme unique communiquant avec des routines d'interruption car :

- —Chacune des tâches peut voir son exécution suspendue après chaque instruction afin de transférer l'usage du processeur à l'autre tâche ;
- —Les changements de contexte ne peuvent être évités qu'en inter-agissant avec l'ordonnateur.

Pour communiquer entre différentes tâches, le noyau fourni des **services** destinés à :

- —Assurer la synchronisation des tâches communicantes ;
- —organiser le transfert de données d'une tâche à l'autre.

Des **objets du noyau** sont les entités constructives d'un système tems réel. Les objets les plus communs dans un noyau RTOS sont :

1. **Sémaphore** : Le concept de sémaphore est utilisé pour contrôler l'accès à une ressource. Lorsqu'une tâche accède à une ressource non partageable, le sémaphore à l'entrée de celle-ci devient bloqué et le reste tant que la tâche a relâché la ressource. Il empêche ainsi tout autre tâche a accéder à cette ressource.
   Dans la signalisation ferroviaire, il ne doit y avoir qu'un train au maximum par tronçon de voie ferrée. Lorsqu'un train entre dans ce troncon, aucun autre train de peut y entrer tant que le premier train ne l'a pas quitté. Un sémaphore peut être généralisé au cas où une ressource est accessible par n tâches simultanément. Dans ce cas, le sémaphore peut prendre n + 1 états qui seront représentés par un compteur.
   Un sémaphore possède une valeur entière s qui représente le nombre de tâches qui accèdent à la ressource qu'il surveille. Sa valeur est positive ou nulle et est uniquement manipulable à l'aide de deux opérations : **wait(s)** et **signal(s)** :
   (a) **wait(s)** est l'action d'attente. Si le sémaphore est libre (s supérieur à 0), la tâche continue son exécution et la valeur de s est décrémentée. Si le sémaphore est bloqué (s = 0), la tâche est placée à la fin de la file d'attente du sémaphore.
   (b) **signal(s)** est l'action de signalisation. Si le sémaphore est libre, il reste libre et sa valeur est incrémentée. Si le sémaphore est bloqué et qu'aucune tâche n'est en attente, il devient libre. Si le sémaphore est bloqué et qu'au moins une tâche est en attente, alors la première tâche de la file d'attente du sémaphore est débloquée.

2. **Mutex** : Un sémaphore qui ne peut prendre que deux états (libre et bloqué) est appelé **sémaphore binaire** ou **mutex**. Dans ce cas, une seule tâche peut avoir accès à la ressource.
3. **Les ftles de communication** Une file d'attente est un objet permettant la communication synchrone ou asynchrone de valeurs entre des tâches. Les caractéristiques d'une file de communication sont les suivantes :
   La capacité maximum de la file (représentant le nombre maximum de valeurs écrites et non encore lues) et la taille de chaque valeur sont fixées.
   Une tâche en attente de données depuis la file est suspendue par l'ordonnanceur (état bloqué).

1.8 Conclusion

Le temps dans un système est très important encore plus s'il s'agit d'un système temps réel. Alors il faut savoir le gérer et c'est sur ce point là que se sont basés les concepteurs système pour la gestion de cette ressource rare et critique.
Un système temps réel est un système dans lequel la validité des résultats dépend non seulement de leurs exactitudes mais aussi de leurs livraisons au bon moment.
Ce type de système est très important dans les environnements industriels, car il répond aux exigences spécifiques de ces derniers. En plus, ces systèmes offrent une solution logicielle très acceptable vis à vis des résultats obtenus.

Chapitre 2

Les systèmes d'exploitation temps-réel

2.1 Introduction

La plupart des appareils domestiques (machines à laver, téléphones, téléviseurs, lecteurs CD/DVD) que nous utilisons contiennent désormais de nombreux logiciels enfouis dedans.
Si des erreurs dans ces logiciels ne sont pas dommageables, il n'en va pas de même pour les logiciels qui contrôlent des centrales nucléaires ou des trajectoires (avions, fusées, voitures) : dans ces systèmes, une défaillance logicielle conduit souvent à des dégâts importants ou des pertes en vies humaines.
On parle alors de **systèmes critiques**, et il convient donc, lors du développement de telles applications, de s'assurer qu'elles satisferont un certain nombre de propriétés, notamment des propriétés de sûreté (comme l'absence de défaillance grave).

Dans ce chapitre nous aborderons les caractéristiques qu'offrent les **SE** temps réel pour aboutir à ces propos, nous creuserons en profondeur dans les fonctionnalités apportées aux SE classiques afin qu'ils supportent le temps réel.

2.2 Les systèmes d'exploitation temps réel

Un système d'exploitation temps réel (RTOS) est un programme qui prévoit l'exécution de façon opportune, gère des ressources du système et fournit une assise cohérente pour développer des applications [21]. les application conçues sur un RTOS peuvent être diverses, s'étendant d'une application simple pour un chronomètre numérique à une application beaucoup plus complexe pour la navigation d'avion.
un Bon RTOS, doit donc être **flexible** pour satisfaire aux ensembles d'exigences des différentes applications.

Par exemple, dans quelques applications, un RTOS comprend seulement un **noyau**, qui est le logiciel principal et qui fournit une **logique minimale**, l'**ordonnancement** et des **algorithmes de gestion des ressources**. tout les RTOS ont un noyau.
D'autre part, un RTOS peut être une combinaison de modules divers, y compris le noyau, un système de fichiers, les protocoles réseaux , et d'autres composantes spécifiques aux applications déstinées , voir figure 2.1

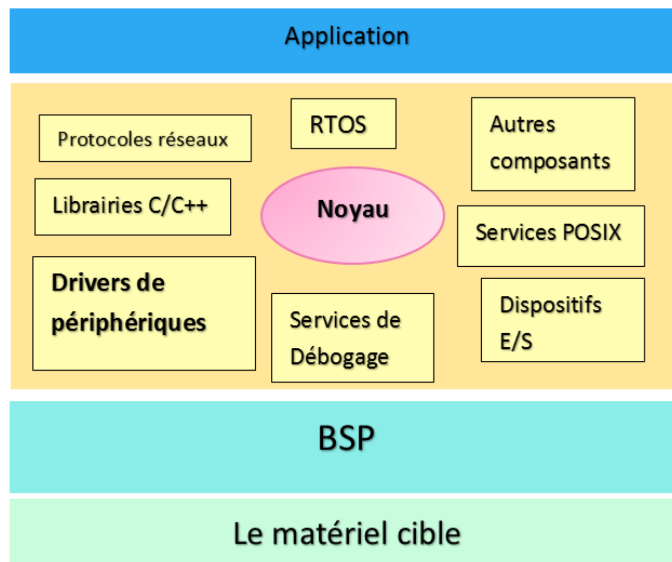


Figure 2.1 – vue des composants d'un RTOS : le noyau et autres composants d'un système enfouis

2.3 Les techniques des RTOS

comme cité dans le chapitre1 Un système temps réel interagit avec un environnement extérieur souvent complexe et en évolution, Il doit pouvoir interagir avec différents types d'éléments matériels Et respecter des échéances temporelle Afin de garantir une fiabilité importante, pour cela un SE temps réel doit pouvoir assurer :

2.3.1 Utilisation du temps concret

au sein d'une application ou d'un système temps réel if faut pouvoir manipuler le temps concret (horloge) ,Le temps réel (ou temps concret) sera utilisé de plusieurs façons :

—Soit en définissant la date à laquelle une action doit être commencée

—Soit en définissant la date à laquelle une action doit être finie.

Il peut être nécessaire de pouvoir modifier ces paramètres en cours d'exécution et de pouvoir préciser les actions à prendre en cas de faute temporelle. Cependant dans le monde réel les périphériques et l'environnement du système évoluent simultanément (en parallèle ou concurrence), ce qui constitue le défi des RTOS : la multitude de tâches de périodes différentes (tâches de supervision, d'archivage, de l'interface graphique, d'asservissement) engendre un problème de concurrence pour l'accès au CPU, la mémoire...
Pour résoudre ce problème on "simule" l'exécution concurrente des processus par la mise en œuvre du **pseudo-parallélisme** :comme il est montré dans la figure 2.2 le parallélisme est apparent à l'échelle de l'utilisateur mais le traitement sur le processeur (unique) est fait séquentiellement en tirant profit des entrées/sorties réalisées par les processus

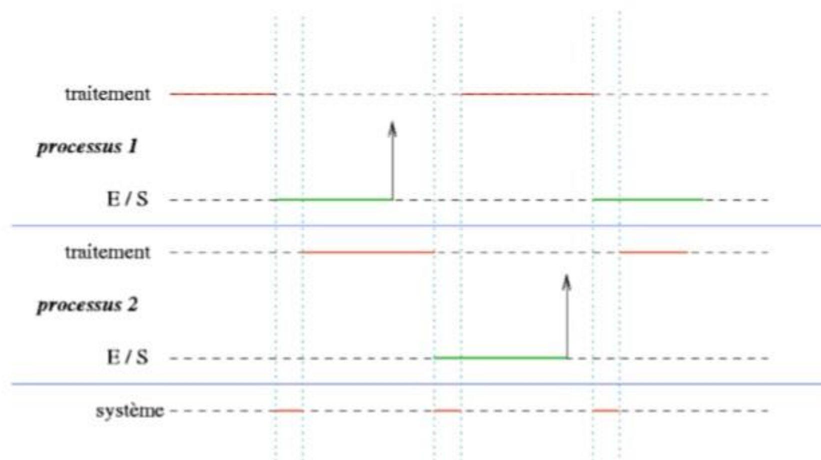


Figure 2.2 – L'execution des processus en pseudo-parallélisme

Un autre problème se pose, celui du Respect des échéances temporelles pour cela le RTOS fait appel à l'**ordonnanceur**

2.3.2 L'ordonnancement

La limitation des ressources (en particulier du processeur) conduit à bloquer des processus (ils ne peuvent progresser du fait de manque de ressource). Afin de respecter en permanence les échéances, il faut gérer efficacement la pénurie et tenter de favoriser les processus dont l'avancement est le plus « urgent » ou le plus prioritaire de la ressource. Une politique d'ordonnancement "temps réel" est alors obligatoire.
Un ordonnancement consiste à définir un ordre sur l'utilisation des ressources du système afin de respecter les échéances temporelles. - **L'ordonnanceur**(scheduler) comme cité au chapitre 1 est le processus système qui gère l'ordonnancement des tâches et processus avec des algorithmes dédiés à ce propos, c'est le coeur du noyau , voilà comment il fonctionne :

1. mis en œuvre périodiquement et dès qu'une tâche se termine
2. l'activation périodique de l'ordonnanceur est provoquée par une horloge dédiée (pas celle de la CPU), elle génère une interruption à chaque tick (milliseconde)
3. à chaque interruption, le noyau gère tous les timers, en déclenchant éventuellement les actions liées aux timers qui viennent d'expirer, gère les budgets de temps d'exécution des tâches et met à jour la liste des tâches prêtes et invoque l'ordonnanceur

Deux algorithmes classiques d'ordonnancement permettent de réaliser :

1. RTM (RaTe Monotonic) : algorithme à priorité fixe (une tâche a une priorité fixe qui est inversement proportionnelle à sa période) pour processus cycliques (le processus le plus prioritaire est celui de plus petite échéance) ; c'est l'algorithme le plus utilisé
2. EDF (Earlest Deadline First) : algorithme à priorité dynamique pour processus cycliques (le processus le plus prioritaire est celui de plus petite échéance). L'ordonnanceur choisit d'exécuter le processus prêt de plus haute priorité Au sein d'une même classe de priorité, le choix peut se faire par temps partagé (Round Robin) ou par ancienneté (gestion FIFO)

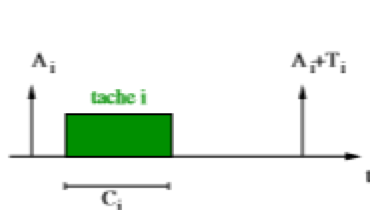
2.3.3 L'ordonnancement préemptif grâce au RTM

— Préemption : capacité à interrompre une tâche au prot d'une autre

— Priorité fixe : chaque tâche a une priorité d'exécution invariante

— Règle : exécution de la tâche de plus haute priorité non-bloquée à chaque appel à l'ordonnanceur

2.3.3.1 Caractérisation d'une tâche



— **Ai** : date d'activation de la tâche i (prête)

— **Ti** : période d'activation de la tâche i

— **Ci** : durée de la tâche i

— **Pi** : priorité de la tâche i

2.3.3.2 Test d'ordonnançabilité

— **Condition suffisante** [3] : N tâches périodiques et indépendantes sont ordonnançables si :

$$\cup = \sum_{i=1}^{N} \frac{\mathbf{Ci}}{\mathbf{Ti}} < N\left(2^{1/N} - 1\right)$$ [1]

Avec $$\mathbf{P}i = \frac{1}{\mathbf{T}i}$$

Compte tenu que cette condition n'est que **suffisante**, il ne faut pas écarter des ensembles de tâches qui ne satisfont pas ce test. Des travaux [10] ont présenté une condition **nécessaire et suffisante** d'ordonnançabilité.

— **Condition nécessaire et suffisante** : Sachant que le temps de réponse d'une tâche ti est borné supérieurement par :

$$\mathbf{RT}i = \sum_{j=1}^{i} [\mathbf{RT}i/\mathbf{P}j] * \mathbf{P}j + \mathbf{C}i$$

L'ordonnancement de l'ensemble des tâches est faisable si et seulement si [11] :

$$\forall i 1 \leq i \leq n$$
$$\mathbf{RT}i \leq \mathbf{P}i$$

2.3.3.3 Les appels systèmes

C'est un service parmis tant d'autres (voir figure2.3) fourni par le noyau,c'est des fonctions exécutées par le noyau au nom d'une tâche utilisateur, ils permettent d'accéder de façon « sûre » à des ressources privilégiées grâce à un changement de contexte d'exécution (1.7.3), après cela il :

1.sauvegarde du contexte d'exécution de la tâche
2.passage de la CPU en mode privilégié « noyau »
3.une fois la fonction terminée
4.le noyau exécute un « retour d'exception »
5.la CPU repasse en mode « normal »
6.l'ordonnanceur est appelé et la tâche de plus haute priorité prend la CPU

2.3.4 Les normes POSIX

POSIX est une famille de normes techniques définie depuis 1988 par l'Institute of Electrical and Electronics Engineers *(IEEE)*, et formellement désignée par IEEE 1003. Ces normes ont émergé d'un projet de standardisation des interfaces de programmation des logiciels destinés à

1. coefficient de Liu et Leyland

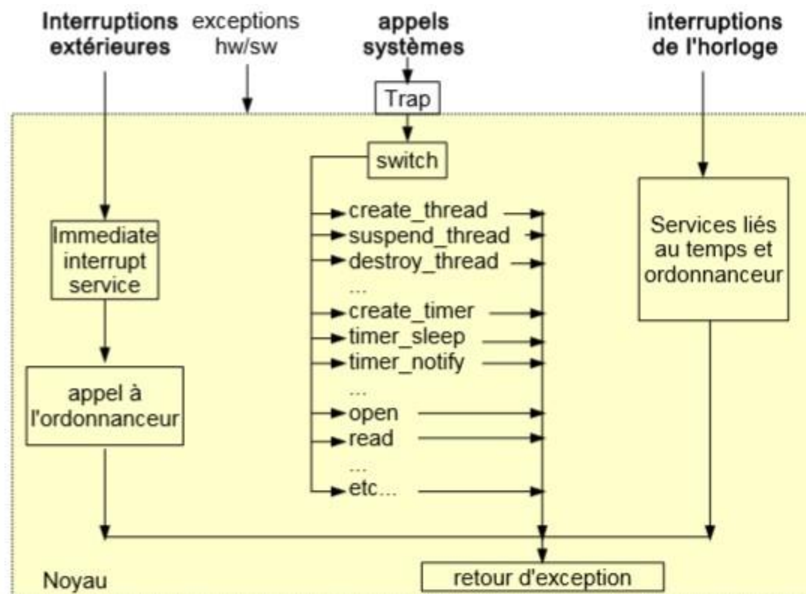


Figure 2.3 – Différents services fournis par le noyau

fonctionner sur les variantes du système d'exploitation UNIX.
Le terme POSIX a été suggéré par Richard Stallman, qui faisait partie du comité qui écrivit la première version de la norme. L'IEEE choisit de le retenir car il était facilement mémorisable. POSIX signifie interface portable de système d'exploitation, et le X exprime l'héritage UNIX. Les standards q'ont rapport avec les RTOS sont :

— 1003.1 pour le SE, processus, système de fichiers et l'interface de programmation du support.
— 1003.2 pour les services
— 1003.1b pour le temps réel
— 1003.1c pour les threads

2.4 Caractéristiques de l'offre des SE temps réel

Les exigences d'une application définissent les exigences de son RTOS sous-jacent. Cependant certains attributs sont impératifs :

—Fiabilité,
—Prédictibilité,
—Performance,
—Flexibilité.

2.4.1 Fiabilité

Les systèmes temps réels doivent être fiables. Le système devrait pouvoir fonctionner durant de longues périodes sans intervention humaine. Toutefois ce critère reste relatif au type de fonction exécutée. Par exemple, une calculatrice à énergie solaire numérique pourrait arrêter de fonctionner sans pour autant causer un problème. D'autre part, commutateur de télécommunication ne peut pas redémarrer pendant l'opération sans engendrer des dégâts.
En général, un système fiable est celui qui est disponible (continue à fournir le service) et n'échoue pas. Une méthode grâce à laquelle les développeurs catégorisent des systèmes t fiables est de compter le nombre d'échecs par année.

2.4.2 prédictibilité

Tout système informatique doit pouvoir assurer un résultat juste dans une tranche de temps assez courte c'est le contexte de la prédictibilité, de même pour les RTOS où le terme utilisé est « déterminisme ».
En effet un RTOS qui satisfait le déterminisme Logique(les mêmes entrées appliquées au système produisent les mêmes résultats) et le Déterminisme temporel : respect des contraintes temporelles (ex : échéance) est un système prédictible.

2.4.3 Performance

Contrairement à ce que l'on peut croire, la performance d'un RTOS n'est pas mesurée par rapport à sa rapidité , mais plutôt au fait qu'il soit « assez rapide » dans le contexte de temps de réponse permis.

2.4.4 Flexibilité

Un RTOS doit pouvoir être utilisé sur différentes plateforme, donc ils doivent être assez modulable afin que sa performance ne dépende pas du hardware sous-jacent .

2.5 Principaux SE temps réel

Le noyau uC/OS II : Ce noyau Temps Réel est digne d'intérêt et porté sur beaucoup de processeurs. Son utilisation est simple et donc conseillé aux débutants dans le Temps Réel.

ITRON : c'est une spécication RTOS ouverte, déstinée aux systemes enfouis decsendants du projet « TRON » (The Real-Time Operating System Nucleus). Les sociétés ayant participé à l'iplémentation de ce projet sont : Fujitsu, Hitachi, Mitsubhishi, Miyazaki, Morson, Erg Co., Firmware Systems, NEC, Sony Corp., Three Ace Computer Corp., et Toshiba [9].

OSE : un RTOS commercial d'Enea Data Systems qui se vante d'avoir rempli la faille entre les applications et le noyau en fournissant un panorama de nouvelles caractéristiques dans le noyau. son architecture basé sur les messages permet d'avoir une milleir synchronisation et communication entre processus [16].

OSEK-VDX : les « systèmes ouverts dans les réseaux automobile » est une spécification RTOS adopté par beaucoups d'organisations automobiles dans leur systèmes enfouis : Adam Opel AG, BMW AG, DaimlerChrysler AG, University of Karlsruhe - IIIT, PSA, Renault SA, Robert Bosch GmbH, Siemens AG, Volkswagen AG [15]

Le noyau RTLinux : est un noyau Temps Réel considérant Linux comme tâche de plus faible priorité. C'est une solution valable et digne d'intérêt. De plus, il est en téléchargement libre et possède un manuel en ligne

Le noyau RTAI : (Real Time Application Interface) il a évolué à du NMIT RTLinux (New Mexico Institute of Technology's Real-Time Linux), et prend une approche unique en exécutant Linux comme une tâche (d'une basse priorité) qui est en concurrence avec d'autres tâches temps réelles sur le CPU [19].

VRTX : un RTOS fiable de Mentor Graphics, qui est le premier système certifié de la rigoureuse standard avionique américaine « (FAA) RTCA/DO-178B »basé sur un Nanokernel exerçant au-dessus d'une couche d'abstraction matérielle afin de fournir une réponce rapide et prédictible [27].

VxWorks : de la société Wind River est un noyau Temps Réel moderne qui a été le premier à intégrer les couches réseau IP et l'API socket. Il intègre un shell et autorise l'édition de liens dynamique. Une plate-forme de cross développement est nécessaire. C'est un excellent RTOS. Son principal défaut est son prix et les royalties à verser pour chaque cible développée [28]

Windows CE : un système d'exploitation enfoui de Microsoft destiné aux petites machines avec de petits processeurs enfouis. Bien qu'avant la version 3.0, il n'était pas qualifié pas comme étant un vrai RTOS, Microsoft a apporté des améliorations considérables dans ce propos et sa dernière version 6.0 est qualifiée comme un noyau temps réel assez complet [22]

Le noyau QNX : QNX de la société QNX Software Systems est un système d'exploitation Temps Réel complet où la machine cible sert aussi de plateforme de développement (comme OS9). QNX peut être aussi utilisé avec une plate-forme de cross développement (Linux, Windows, Solaris).

La liste susmentionnée n'est en aucun cas exhaustive, mais c'est une concoction raisonnable des RTOS commerciaux et ouverts disponible actuellement.

2.6 Conclusion

Ce chapitre a rappellé, les concepts utilisés dans les systèmes temps réel. Pour cela, il introduit des définitions, et aborde les principes qui proviennent de la notion de multitâche et les défis liés au problème de synchronisation et d'ordonnancement temps réel achevé grâce à de nombreux algorithmes. Ce chapitre surligne, les principales différences entre les systèmes classiques et les systèmes temps réel puis fait une bréve présentation des plates-formes et des systèmes temps réel utilisés dans le milieu industriel, et ce, d'une façon générale. Il aborde les différents éléments les caractérisant, tout en rappelant brièvement leurs diverses évolutions. Les systèmes comme **RTAI**, **VxWorks** et **RTLinux**.

Chapitre 3

Linux et le temps réel

3.1 Introduction

Linux [1] est un membre de la grande famille des systèmes d'exploitations basés sur UNIX, un nouveau venu dont la popularité a explosé à la fin des années 90, linux a rejoint les SE populaires de l'époque, à l'instar de system V Release 4 (SVR4), AIX d' IBM, Solaris de Sun Microsystems et Mac OS X de Apple Computer, Inc.
Cependant ce qui le distingue des autres SEs est le fait qu'il soit ouvert (open source), son code est disponible sous la licence GNU General Public License (GPL)[5].
Linux était d'abord développé par Linus Torvalds en 1991 comme un système d'exploitation destiné aux PC IBM basés sur le microprocesseur INTEL 80386. Ensuite amélioré à l'aide des centaines de développeurs Linux autour du monde afin d'être à jour avec les changements matériels qu'a connu cette époque, l'union des développeurs a étendu sa compatibilité vers d'autres architectures [2] :

—Hewlett-Packard (HP) Alpha
—Intel Itanium, AMD64 d'AMD
—Intel Itanium, AMD64 d'AMD
—Motorola 68k et ses variantes
—Alpha
—Power PC
—ARM
—SPARC
—MIP

Dans ce chapitre nous verrons comment Linux a adopté le Temps réel.

3.2 Les qualités de linux

Ici, on discutera les caractéristiques importantes de linux (et des systèmes d'exploitation basés sur Unix en générale)[2] :

1. LINUX Est une marque déposée de Linus Torvalds.

- Multitâche : certes, l'ordonnanceur de linux implémente le multi-tâches préemptive dans le sens où un processus de haute priorité déclenché par un évènement non synchronisé va suspendre l'exécution du processus en cours d'exécution. Mais, les services du noyau linux ne sont guère préemptibles, c.-à-d. qu'un processus s'exécutant en mode noyau (faisant un appel système en l'occurrence) ne peut être interrompu, et compte tenu que certains services prennent un temps considérable, cela engendre une latence importante ce qui rend le noyau linux standard généralement non convenable pour des applications temps réel.
- Multi-utilisateur :linux a évolué comme étant un système à temps partagé : plusieurs personnes peuvent travailler sur le même ordinateur (coûteux) en même temps. Désormais, il existe des options qui supportent la discrétion et la protection de données, entretemps linux a préservé cet héritage et le met en profit dans les environnements serveurs.
- Multiprocesseurs : Linux offre un support complet pour un system multiprocesseurs symetrique (SMP) qui est une architecture parallèle qui consiste à multiplier les processeurs identiques au sein d'un ordinateur, de manière à augmenter la puissance de calcul, tout en conservant une unique mémoire.
- Mémoire protégée : chaque processus sur linux opère dans sa zone mémoire dédiée, ça permet de protéger les autres processus si un processus donné est affecté d'un bug , et c'est une manière de protéger le système d'exploitation aussi.
- Système de fichier hiérarchique : Un système de fichiers est un format de stockage des fichiers sur des supports tels que le disque dur, la disquette, le CDrom et autres mémoire de masse.Linux supporte beaucoup de systèmes de fichiers différents dont ceux compatibles avec DOS. Il possède également son propre système de fichiers appelé ext2 et ext3 qui est conçu pour une utilisation optimale du disque dur [25]
  - Ext2 : il s'agit du système de fichier le plus utilisé actuellement sous Linux. Son utilisation sur des stations de travail ou des serveurs ne pose aucun problème. En utilisant un système de redondance des structures vitales, il permet souvent de retrouver un système cohérant après un arrêt forcé, grâce à un utilitaire semi- automatique du nom de e2fsck. Cette procédure de restauration nécessite l'intervention d'un administrateur ce qui n'est pas concevable dans le cas d'un système embarqué.
  - Ext3 : est le nom d'un système de fichiers utilisé notamment par GNU/Linux et est une évolution de ext2, le précédent système de fichiers utilisé par défaut par de nombreuses distributions Linux.

3.3 Architecture du noyau Linux

Les noyaux Unix avaient commencé comme étant des noyaux **monolithiques** : chaque couche est intégrée dans le code du noyau et est exécutée en mode noyau au dépond du processus en cours d'exécution .en revanche l'architecture **micronoyau**,développée plus tard, ne requiert qu'une petite partie du noyau, y compris des primitives de synchronisation, un ordonnanceur simple et un mécanisme de communication entre processus.
Cette dernière architecture est favorisée en termes de flexibilité, bien qu'elle soit plus lente que la monolithique, à cause du coût d'échanges de messages explicites entre les couches. Or, les avantages de cette approche couvre sur ce défaut, car les micronoyaux oblige les programmeurs à adopter une approche modulaire, vu que chaque couche du SE est relativement indépendante qui doit interagir avec l'autre couche à l'intermédiaire d'une interface logicielle bien définie.
D'autant plus que cette architecture offre une flexibilité parce que les composants dépendants du matériel sont encapsulés dans le code du micronoyau. Finalement les systèmes conçus sur cette approche tendent à bénéficier d'un meilleur usage de la **RAM** comparés aux monolithiques.
Pour obtenir le meilleur des deux, linux propose un compromis entre les deux architectures grâce aux modules (figure 3.1). Un module est un fichier « kernel object » (*.ko) qui peut être lié (ou délié) au noyau de façon dynamique au cours de l'exécution, c'est une partie intégrante du noyau (ne tourne pas sous forme d'un processus supplémentaire au-dessus du noyau)[5].

3.4 Un Linux temps réel

les débuts de Linux comme un système monolithique comportant des sections non préemptives et très longues ont fait qu'il est un système généraliste et dont l'ordonnanceur est basé sur la « justice » plutôt que la priorité , donc pas conçu pour fournir un comportement temporel déterministe, mais comme cité auparavant en raison de son ouverture et de son applicabilité à travers une large gamme de plates-formes matérielles et de sa latence relativement basse, deux approches pour avoir un comportement temps réel avec linux sont possibles :

1. implémentation d'un noyau temps réel sous le noyau linux, appelée l'approche Co-noyau et qui est considérée comme l'embryon de beaucoup de SE temps réel existant aujourd'hui à l'instar de RTLinux de FMSLabs (ce point sera développé ultérieurement).
2. La seconde approche consiste à apporter des modifications au noyau linux traditionnel, cette approche a fait apparition en l'an 2000 à partir de la version 2.6 de linux a engendré un noyau complètement préemptible (grâce au **patch PREEMPT—RT**)

Figure 3.1 – insertion de module sur le noyau linux

Le premier système d'exploitation temps réel (RTOS) utilisant l'architecture micronoyau et un noyau Linux est **RTLinux** [26] dont la première version date de 1996. Basé sur l'architecture micronoyau, il mettait en avant le fait d'exécuter le noyau Linux en tant que tâche entièrement préemptible et l'exécution des tâches temps réel dans l'espace noyau sous forme de modules.
Ce RTOS est cependant devenu un produit commercial lorsque ses créateurs ont fondé FSMLabs. En 2007, Wind River a racheté la solution et l'a rendue accessible au public sous le nom de *Wind River Real-Time Core for Wind River Linux*, dont la ligne de produit a finalement été discontinuée en août 2011. Peu après la publication de RTLinux, un autre groupe de recherche a vu le fruit de son travail aboutir par la création de Real-Time Application Interface (**RTAI**) [19]. La différence principale entre **RTAI** et **RTLinux** se situe sur les changements effectués au noyau Linux. En effet, **RTLinux** modifie le comportement du noyau Linux pour accepter le micronoyau en insérant ses modifications directement dans le code source de Linux.

Ainsi, il devient Beaucoup plus difficile de maintenir l'état du noyau Linux au fur et à mesure de son évolution, de même que d'identifier, lors de l'apparition d'une erreur, si elle vient de la surcouche ajoutée par **RTLinux** ou de ses modifications du noyau Linux. **RTAI** de son côté a opté pour une couche d'abstraction matérielle composée d'une structure de pointeurs vers les adresses mémoires des gestionnaires d'interruptions

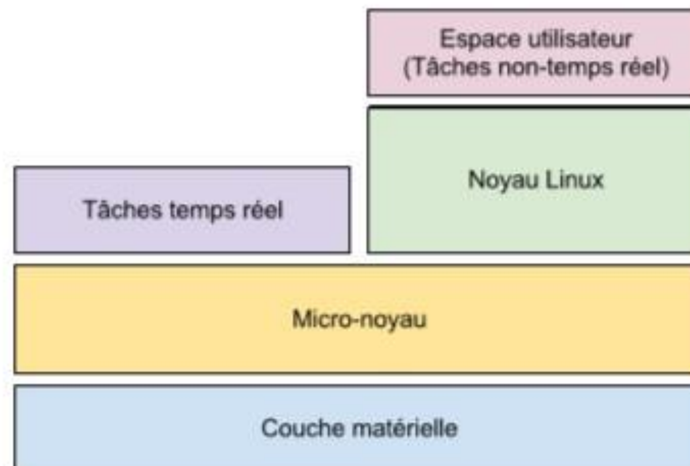
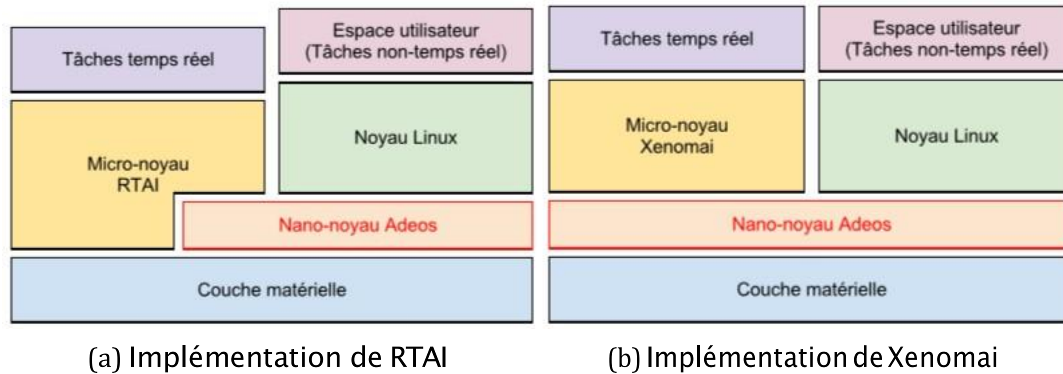


Figure 3.2 – Représentation de l'architecture de fonctionnement du temps réel sur un système d'exploitation basé sur l'architecture micro-noyau

ainsi que vers les fonctions permettant d'activer ou de désactiver ces interruptions.
La création de cette couche d'abstraction permet de minimiser les retouches du code source de Linux puisqu'elle est implémentée en ne modifiant et n'ajoutant que quelques dizaines de lignes de code.
De plus, l'utilisation d'une telle couche permet aisément de la désactiver pour isoler des bogues ou lorsque le temps réel n'est pas requis en changeant les valeurs des pointeurs pour les valeurs originales. Alors qu'en 2001 les créateurs de **RTLinux** déposent un brevet concernant le comportement du micronoyau, un nouveau projet voit le jour dans le but de le contourner, avec l'implication de **RTAI**. Il s'agit d'**Adeos** (Adaptive Domain Environment for Operating Systems)[17].Adeos vise à rendre le système temps réel modulable tout en permettant de séparer la couche d'abstraction matérielle du noyau. Il fournit un environnement exible permettant de partager les ressources matérielles entre plusieurs systèmes d'exploitation ou plusieurs instances d'un même système d'exploitation, en contactant en séquence les différents systèmes déclarés de façon à ce que le système le plus prioritaire décide si oui ou non une interruption, par exemple, devra être propagée. L'approche micronoyau adoptée par **RTAI** change alors quelque peu, le noyau temps réel interceptant toujours les interruptions mais reposant sur Adeos pour les propager au noyau Linux dans le cas où ces interruptions n'influent pas sur l'ordonnancement temps réel, comme le présente la Figure 3.3a [1]. Le développeur de ce «**nano-noyau**» est l'auteur et développeur prin- cipal de **Xenomai**, un autre RTOS à architecture **micro-noyau** [30]. Xenomai repose lui aussi sur Adeos mais laisse une plus grande place au nano-noyau dans l'architecture générale qu'il utilise, comme le montre la Figure 3.3b. En effet, dans le cas de Xenomai, Adeos fonctionnera



(a) Implémentation de RTAI (b) Implémentation de Xenomai

Figure 3.3 – Architectures basées sur Adeos utilisées par RTAI et Xenomai

simplement de la façon dont il a été conçu : comme un **hyperviseur** pouvant supporter plusieurs systèmes d'exploitation en leur donnant à chacun une priorité. Tandis que **RTAI** est un système qui se concentre principalement sur l'obtention des latences les plus faibles possibles (d'où l'utilisation d'une architecture où RTAI reçoit directement les requêtes d'interruption), **Xenomai** considère aussi l'extensibilité, la portabilité et la maintenabilité comme objectifs de développement. Sur ce type d'architecture, cependant, la séparation du noyau supportant le temps réel et du noyau standard Linux pose par contre problème lorsqu'il s'agit de déboguer l'exécution d'une application composée de tâches temps réel et d'autres non-temps réel. En effet, un système d'exploitation basé sur cette architecture rend ces tâches indépendantes. De même, la communication entre ces tâches nécessite l'utilisation de tubes nommés (**FIFO**) ou de mémoire partagée.

Un travail est fait sur le noyau Linux pour le rendre de plus en plus apte à fonctionner avec des applications temps réel. Si l'on utilise un noyau Linux standard sans configuration spécifique, lorsqu'un processus de l'espace utilisateur fait un appel système il ne peut pas être préempté. Cela signifie que si un processus de haute priorité souhaite faire un appel système, mais qu'un processus de basse priorité est déjà en train d'en exécuter un, le processus de haute priorité devra attendre que le processeur ait ni d'exécuter l'appel système du processus de plus basse priorité.

Depuis le noyau 2.6, l'option **CONFIG PREEMPT** a été ajoutée pour permettre de palier à ce comportement. En utilisant cette option, si un processus de haute priorité apparaît, un processus de plus faible priorité sera **préempté**, y compris s'il est au milieu d'un appel système, permettant d'obtenir les performances requises par les systèmes temps réel à contraintes souples.

Il existe un correctif spécifique au temps réel pour le noyau Linux, appelé **PREEMPT – RT** [13].

Ce correctif permet de corriger le comportement du noyau pour celui d'un système temps réel à contraintes strictes, tout en limitant le nombre de

modications apportées. La majeure partie du noyau devient préeemptible grâce à quelques changements ciblés, notamment la réimplémentation de certaines primitives de verrouillage du noyau, la conversion des gestionnaires d'interruptions en fils d'exécution ou encore la mise en œuvre de l'héritage de priorité pour les mutex internes au noyau an d'éviter les situations d'inversion de priorité [14].

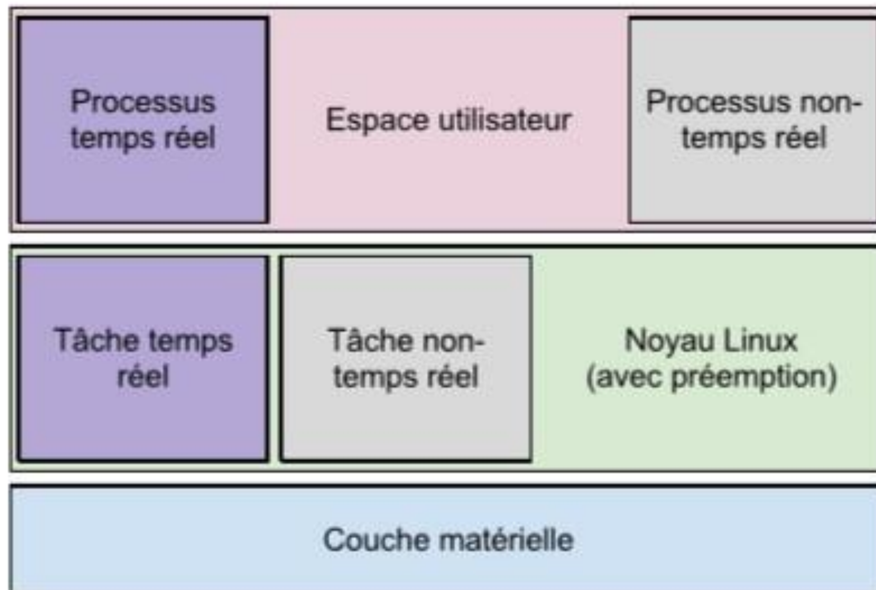


Figure 3.4 – Représentation de l'architecture de fonctionnement du temps réel sur un système d'exploitation GNU/Linux

enfin, le nombre des fonctionnalités développées pour ce correctif ont été intégrées dans le noyau standard. Lors de la création des premiers RTOS à architecture micronoyau, il était reproché à Linux de ne pas préempter l'exécution d'une tâche pendant un appel système. On lui reprochait aussi qu'une tâche de haute priorité doive attendre après une tâche de plus faible priorité pour accéder à une ressource, tant que cette dernière ne l'aura pas relâchée, ou encore que son ordonnanceur pouvait donner du temps d'exécution à une application alors qu'une application de plus haute priorité était en attente. La gestion du matériel faite par le système pouvait aussi poser problème, réordonnançant par moments les tâches pour utiliser le matériel de manière plus efficace. Nombre de ces reproches ont été corrigés, soit dans le noyau standard, soit dans le correctif PREEMPT—RT, et ne sont désormais plus d'actualité.
L'approche PREEMPT—RT s'agit de modifications apportées au noyau Linux pour lui donner un comportement temps réel, cette technique devenue populaire en l'an 2007 est présentée sous forme d'un patch et repose sur la philosophie suivante :

—l'essentiel du code du noyau est constitué par les drivers

- —seuls les drivers qui seront utilisés par l'application temps réel devront avoir un comportement temps réel
- —les parties du cœur du noyau qui peuvent affecter le comportement temps réel sont celles où la préemption est désactivée, et concernent le verrouillage et la gestion des interruptions

3.5 Traçage du noyau Linux

Le traçage est une technique d'analyse de performance des systèmes informatiques qui permet de récupérer des informations sur une application durant son exécution en limitant au maximum l'impact sur cette dernière. Le traçage se différencie du débogage par le fait que son but n'est pas d'arrêter le fonctionnement du programme analysé pour l'observer à un moment donné, mais au contraire de le laisser s'exécuter pour pouvoir faire une analyse à postériori du déroulement de son exécution. En outre, le traçage permet d'obtenir une vue d'ensemble des interactions entre les composantes d'une application et celles du système d'exploitation.

3.6 Conclusion

Les propriétés des systèmes temps réel sont de plus en plus recherchées sur des systèmes standards, et ainsi les évolutions faites dans le cadre de ces systèmes bien spécifiques sont petit à petit intégrées au noyau **Linux**. Aujourd'hui, le noyau standard de linux et les modifications/patchs téléchargeables apportent beaucoup de bénéfices et améliorent la perfor- mance temps réel du noyau. Dans ce chapitre nous avons discuté les techniques (anciennes et récentes) employées afin d'implémenter le temps réel.

Chapitre 4

Mise en oeuvre

4.1 Introduction

Le but du travail élaboré dans ce mémoire est de concrétiser l'approche temps réel avec linux Redhat. Avec l'atout majeur de Linux étant un système ouvert, nous voulions mettre en œuvre un Système d'exploitation apte à prendre en charge le temps réel aussi parfaitement qu'un système commercialisé, mais à moindre coût, nous allons concrétiser cela et expliquer la démarche entreprise dans ce chapitre

4.2 Traçage de RTLinux

dans cette section, nous allons comparer la performance (la **latence** 1.2.2) d'un Linux générique (Vanilla) et RTLinux effectué grâce à l'outil Cyclictest [29] . Le test se fait sur :

—vanilla 2.6.24.7 kernel

—vanilla 2.6.24.7 kernel avec RTLinux

—500,000 boucles

—hackbench en arrière-plan

Le résultat est le suivant (en μsec) :

Vanilla(500k échantillons)

—Min = 1

— **Max = 2857**

—Moyenne=11.47

—déviation standard = 54.94

Vanilla/RTLinux(500k échantillons)

—Min = 4

— **Max = 43**

—Moyenne=8.34

—déviation standard = 1.49

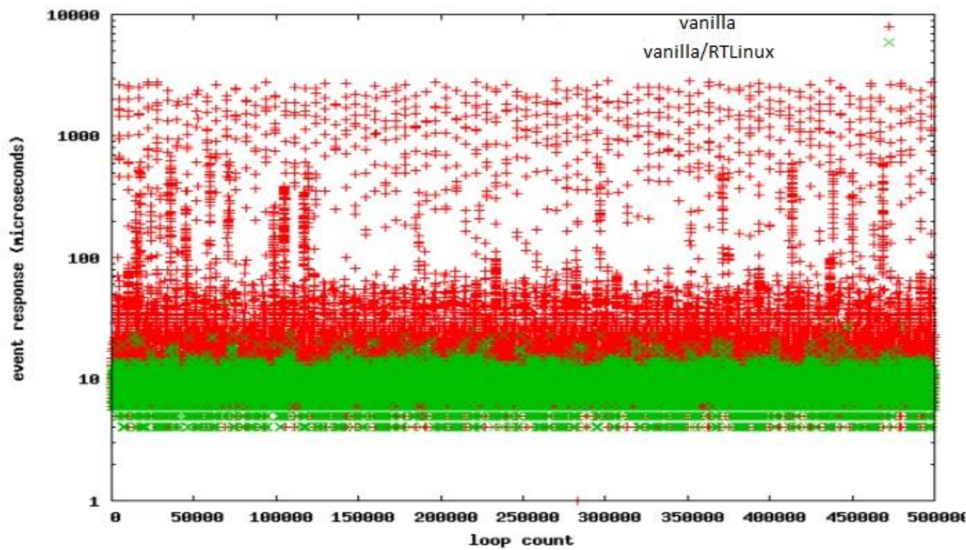


Figure 4.1 – Trace de RTLinux

4.2.1 L'outil de trace : Cyclictest

Cet outil permet de vérifier la performance temps réel du système d'ex- ploitation en exécutant de multiples processus sur différents processeurs, chacun exécutant une tâche périodique [4]. Chaque tâche peut avoir une période différente, et la priorité de chaque processeur peut être définie à n'importe quelle valeur jusqu'à une priorité temps réel.
La performance est alors évaluée en mesurant la différence entre la période désirée et la période réelle de la tâche exécutée. .Autrement dit, on analyse la différence entre l'heure à laquelle l'application aurait dû être réveillée et l'heure à laquelle elle a effectivement été réveillée par le système. Ce test nous donne une indication sur les latences dues à l'ordonnanceur temps réel du système.

4.3 L'installation de RTLinux

On a suivi les étapes suivantes pour achever l'installation de RTLinux sous Linux RedHat et les patchs adéquats :

1. L'installation de RedHat Linux
   On va procéder à la création d'une partition de disque et y installer la RedHat. On va installer les outils de développement du noyau, le compilateur gcc, et les services tels que :

```
1 # patch /*Pour crer un patch*/
2 # make
3 #bzip2 /*compression des fichiers*/
```

2. télécharger le noyau Linux
3. télécharger le noyau RTLinux et les patchs correspondants

4. mettre une copie du noyau **RTLinux** sur /usr/src/rtlinux-3.2-pre1 en utilisant les commandes suivantes :

```
1 #cd /usr/src
2 #tar xjf rtlinux−3.2−pre1.tar.bz2
```

Ça va créer le répertoire "rtlinux-3.2-pre1" sous /usr/src.

5. mettre une copie du noyau **Linux** sur /usr/src/rtlinux-3.2-pre1/linux en utilisant les commandes suivantes :

```
1 #cd /usr/src/rtlinux−3.2−pre1
2 #tar xjf linux−2.4.18.tar.bz2
3 #ln −fs linux−2.4.18/ linux
```

Cela va créer le répertoire symbolique "Linux" lié au "linux-2.4.18" sous /usr/src/rtlinux-3.2pre1.

6. **Patcher** le noyau Linux avec le patch RTLinux :

```
1 #cd /usr/src/rtlinux−3.2−pre1/patches
2 #bzip2 −d kernel patch−2.4.18−rtl3.2−pre1.bz2
3 #cd /usr/src/rtlinux−3.2−pre1/linux
4 #patch −p1 < /usr/src/rtlinux−3.2−pre1/patches/kernel_patch−2.4.18
```

7. Nettoyer tous les fichiers ".O"

```
1 #cd /usr/src/rtlinux−3.2−pre1/linu
2 #make mrproper
```

8. Configurer le noyau **Linux**

```
1 #cd /usr/src/rtlinux−3.2−pre1/linux
2 #make cong /* text mode */ ou bien
3 #make xcong /* X mode */
```

9. Construire un nouveau noyau **Linux** suivre ces étapes consécutive- ment :

```
1 #cd /usr/src/rtlinux−3.2−pre1/linux
2 #make dep /* dependency refresh */
3 #make bzImage /* compiler le noyau linux */
4 #make modules /* compiler les modules */
5 #su /* se mettre en superviseur (si pas fait) */
6 #make modules install /* Installer les modules linux compils */
7 #cp arch/i386/boot/bzImage /boot/rtzImage
```

10. Configurer le boot loader il faut d'abord s'assurer que le système du fichier racine "/" est redirigé vers /dev/hda3 et le boot vers /dev/hda2 en utilisant la commande

```
1 #df
```

11. Rebooter et selectionner RTLinux RTLinux doit démarrer

12. Configurer RTLinux

```
1 #cd /usr/src/rtlinux−3.2−pre1
2 #make xcong
```

13.Compiler RTLinux

```
1 #cd /usr/src/rtlinux−3.2−pre1
2 #make dep /* dependency refresh */
3 #make
4 #su
5 #make devices
6 #make install
```

14.éxecuter le test de régression

```
1 #./scripts/regression.sh
```

Au final nous aboutirons à un dual boot avec la spécification RT, désignant RTLinux (figure 4.2)

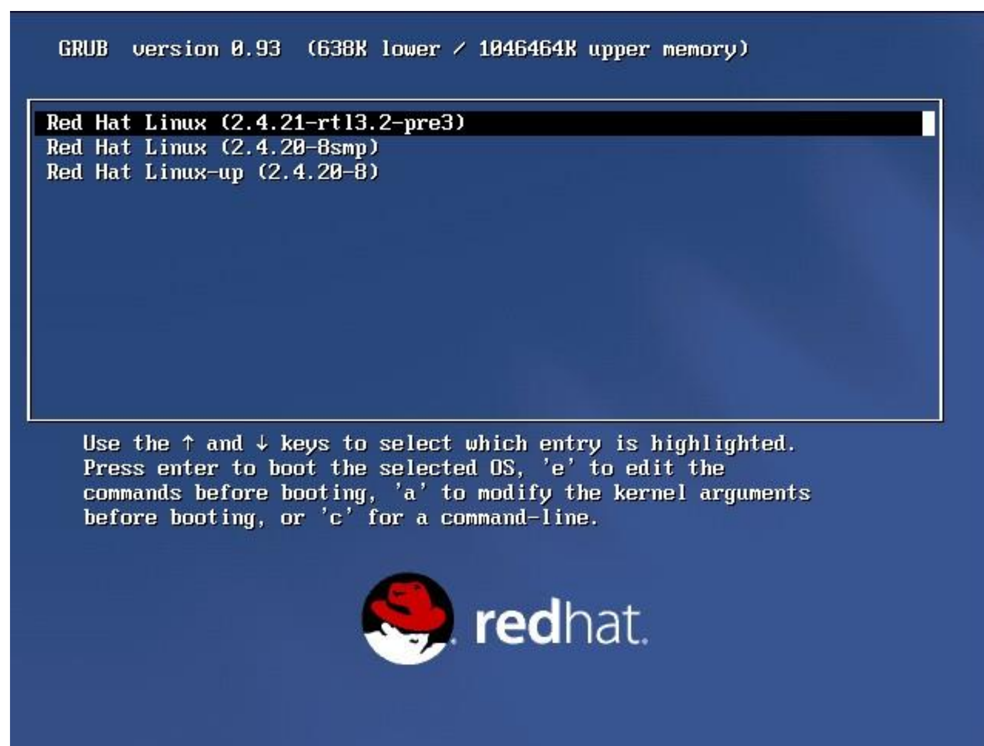


Figure 4.2 – Capture d'écran qui présente le Dual boot de Linux et RTLinux

4.4 Développement sous RTLinux

Le développement d'applications sous RTLinux suit les règles suivantes :

—Découpage en 2 parties : TR et non TR.
—La partie TR doit être la plus simple et la plus courte possible.
—Le reste est non TR et sera développé dans l'espace user sous forme de processus Linux.
—Les processus Linux et les tâches TR pourront communiquer par des FIFOS ou par mémoire partagée.

L'API POSIX thread existant déjà sous Linux a été portée sous RTLinux. Cela facilite la migration d'un développeur Linux vers RTLinux.

4.5 Structure d'un module

Un module est un morceau de code permettant d'ajouter des fonctionnalités au noyau : pilotes de périphériques matériels, protocoles réseaux, etc... sa structure se présente comme suit :

—Pas de main() ;
—Une fonction exécutée au chargement du module : init—module ;
—Une fonction exécutée au déchargement du module : cleanup—module ;
—Des fonctions appelées par les fonctions précédentes.

4.6 Test du port parallèle sous RTLinux

4.6.1 Utilisation du port parallèle

La meilleure façon de se familiariser avec un système temps réel est quand même de l'utiliser pour sa raison d'âtre : l'interaction avec le mode réel. Or, quelle interface avec l'extérieur est mieux adaptée que le port parallèle pour faire des entrées/sorties numériques ? Disponible sur la plupart des PC et simple à programmer ; on peut néanmoins s'initier à l'utilisation de deux modes de fonctionnement : *polling* et interruptions tout en mettant en pratique les fonctions primordiales de l'**API** RTLinux [8].

4.6.2 Le code du port parallèle

4.6.2.1 lpt—irq.C[12]

```c
#include <stdio.h>
#include <asm/io.h>
#include <unistd.h>
#include <sys/io.h>
#define LPT 0x378

int main(void){
        int in;
        if (ioperm(LPT, 3 , 1)< 0) {
                fprintf(stderr,
                "ioperm: erreur en accedant à IO_ports")
                exit(-1);
        }
        while(1) {
                in=inb(LPT+1);
                in = in >> 3;
                in = in & 0x0f;
                        if(in==0){
                            printf("une interruption sur LPT\n");
                            outb(0xff , LPT); /*le 0 logique sur LPT*/
                                usleeo(100);
                            outb(0x0 , LPT); /*le 1 logique sur LPT*/
                            }
            }
}
```

Figure 4.3 – Capture d'écran du code C du port lpt—irq

4.6.2.2 rt—irq—gen.C[12]

```c
#include <rtl.h>
#include <rtl_time.h>
#include <rtl_fifo.h>
#include <asm/rt_irq.h>
#include <asm/rt_time.h>
#include <rtl_sched.h>
#include <rtl/io.h>
#include <linux/cons.h>
#include <pthread.h>
#include <time.h>
#include <rtl_shed.h>
#include <rtl_sync.h>
#include <unistd.h>
#include <rtl_debug.h>
#include <errno.h>
#define LPT 0x378
#define LPT_IRQ 7
#define RTC_IRQ 8

struct sample {
            hrtime_t mmin;
            hrtime_t max;
};

/*le temps est mesure en nanoseconde*/
#define TIMEOUT 10000000
#define SAMPLES 10

pthred_t thread;
hrtime_t min_response;
hrtime_t max_response;
struct sample samp;
int samples;

void * irq_gen(void *arg) {

/*mettre ces variables dans des registre
 * va eviter des retards additionnels
 * causes du chargement et enregistrement
```

```c
void * irq_gen(void *arg) {

/*mettre ces variables dans des registre
 * va eviter des retards additionnels
 * causes du chargement et enregistrement
 * en memoire pendant la boucle du temps
*/
register int i;
register int orig;

int old_interrupts;
hrtime_t before, after, response;
struct sched_param p;
p.shed_priority = 1;
pthread_setschedparam (pthread_self(),SHED_FIFO, &p);

pthread_make_periodic_np (pthread_self(),gethrtime() , 100000000);

while (1) {
        min_response = 2000000000;
        max_response = 0;
        for (samples = 0;samples < SAMPLES; samples++)
                /* suspendre les interruptions pour avoir le temps */
                /* de round-trip avec la boucle de l'attente active */
                rtl_no_interrupts(old_interrupts);
                outb_p(0xf , LPT);
                orig = inb_p(LPT + 1);

                outb(0x0, LPT); /*declencher une interruption*/
                before = gethrtime();
                i = 0;

                while ((inb(LPT + 1) == orig) && i++<TIMEOUT) ;
                        after = gethrtime();
```

```c
                        /* si cette boucle n'etait pas presente ,le systeme*/
                        /* devra attendre longtemps ,jusqu'a ce qu'une
                        interruption*/
                        /*est mise en attente active (polling)*/
                        if (i>=TIMEOUT) {
                                rtl_print("timeout!!\n") ;
                                return 0;
                                }
                response = after - before;
                if (response < min_response) {
                    min_response = response;
                    }
                if (response > max_response) {
                    max_response = response;
                    }

                 /*restaurer les interruptions avant de mettre le thread
                 en veille */
                rtl_restore_interrupts (old_interrupts);
                pthread_wait_np();
                }
        samp.min = min_response;
        samp.max = max_response;
        rtf_put(0, &samp, sizeof(samp));
        }
        return 0 ;
    }

    int init_module(void) {
    rtf_destroy(0);
    rtf_create(0,4000);
    return pthread_create (&thread,NULL,irq_gen,0)
    }

    void cleanup_module(void) {
    rtl\_printf ("retrait de rt_gen_irq du CPU et le nettoyage de ACK");
    outb_p(0xf,LPT);
    pthread_delete_np (thread);
    rtf_destroy(0);
    }
```

Figure 4.4 – Captures d'écran du code C du port rt— irq— gen

4.7 Conclusion

Linux est non seulement une parfaite plateforme pour l'expérimentation et caractérisation des algorithmes temps réel, mais il a su aussi supporter le temps réel dans ses deux formes. Notre travail présente une mise en œuvre complète de ce pouvoir de linux, utilisant sa version Red-Hat et reposant sur l'environnement RTLinux. Compte tenu de ce qui précède, Linux peut se prévaloir d'être un concurrent pour de nombreux systèmes d'exploitation Temps Réel.

Conclusion générale

Au départ, le système d'exploitation Linux n'est pas un système temps réel. Avec **RTLinux**, un micronoyau est inséré au-dessous du noyau Linux qui constitue ainsi une tâche de priorité minimale. **RTLinux** intercepte les interruptions à la place du gestionnaire d'interruption de Linux rendant ainsi Linux préemptif. A ce stade, le service temps réel ainsi fournit est « Temps Réel Dur » tout en profitant des avantages de Linux pour développer des applications Real Time.

Nous avons pu voir au cours de ce mémoire que Linux adapté peut répondre véritablement aux impératifs de la mise en œuvre de solution de pointue dans l'univers industrielle.
Linux apparait donc comme une solution alternative aux systèmes et logiciels propriétaires, une solution économique qui permet, grâce à une communauté très active, de s'adapter aux évolutions technologiques très rapides et de développer ses propres applications.

Compte tenu de ce qui précède, linux peut se prévaloir d'être un concurrent pour de nombreux systèmes d'exploitation temps réel, cependant, il semble maintenant nécessaire d'aborder des questions qui relèvent de l'application des deux types de patch bien connus dans l'environnement Linux : le patch **Preempt Kernel** et le patch **low Latency**. Le principe du premier est d'ajouter systématiquement des occasions d'appel au **schedule()** (l'ordonnanceur Linux). Quant au second patch, son principe est un peu différent car au lieu d'opter pour une stratégie systématique, les développeurs du patch ont préféré effectuer une analyse afin d'ajouter des points de préemption subtilement placés dans les sources du noyau afin de casser des boucles d'attente trop longues. Pour commenter les résultats dans chaque cas, on compte utiliser de mesure soigneusement choisi.

Par ailleurs, nous entrevoyons pour nos travaux futurs de procéder à une étude comparative assez poussée entre les différentes extensions temps réel existantes sous Linux pour aboutir à la meilleure solution.

Enfin, on envisage de mener un développement d'une application temps réel complète afin de voir de près le comportement de l'extension **RTLinux**.

Bibliographie

[1]A. Luchetta A. Barbalace. *Performance Comparison of VxWorks, Linux, RTAI, and Xenomai in a Hard Real-Time Application*. T. 55. Nuclear Science, 2008, 435–439.

[2]Doug Abbott. *Linux for Embedded and Real-time Applications*. 2012. url : http://pan.baidu.com/share/link?uk=1040995049&shareid=2586234247&third=0&adapt=pc&fr=ftw.

[3]J. Layland C. Liu. « Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment ». In : *Journal of ACft* 29 (1973), p. 46–61. url : http://www.cis.upenn.edu/~lee/07cis505/Lec/liu73scheduling.pdf.

[4] *Cyclictest*. url : https://rt.wiki.kernel.org/index.php/Cyclictest.

[5]Marco Cesati Daniel P. Bovet. *Understanding the Linux Kernel*. O'Reilly, 2005. url : http://free-electrons.com/doc/books/lkn.pdf.

[6]Joan-Sébastien Morales Éric Gaul. url : http://profdinfo.com/web/420-KH2-LG/introduction_systemes_temps_reel.html.

[7]Don Gillies. url : http://www.ece.ubc.ca/~gillies/.

[8]« GNU/Linux : systèmes embarqués ». 2000.

[9]K. Tamaru H. Takada Y. Nakamoto. « The ITRON Project : Overview and Recent Results ». In : (1998).

[10]Y. Ding J. Lehoczky L. Sha. « The Rate Monotonic Scheduling Algorithm : Exact Characterization and Average Case Behaviour ». In : *Journal of ACft* (1989), p. 166–171. url : http://www-2.cs.cmu.edu/~ssaewong/research/exact-rm.pdf.

[11]Paritosh Pandya Mathai Joseph. « Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment ». In : *The Computer Journal* 29 (1986). Sous la dir. d'Oxford University Press, p. 390–395. url : http://comjnl.oxfordjournals.org/content/29/5/390.full.pdf.

[12]Nicholas Mc Guire.
« MiniRTL Hard Real Time Linux for Embedded Systems ».
In : *Proceedings of the 4th Annual Linux Showcase & Conference - Volume 4*. ALS'00.
Atlanta, Georgia : USENIX Association, 2000, p. 8–8.
url : http://dl.acm.org/citation.cfm?id=1268379.1268387.

[13]P. E. McKenney. « A realtime preemption overview ».
In : (2005). url : http://lwn.net/Articles/146861/.

[14]P. E. McKenney. « CONFIG PREEMPT RT patch ».
In : (2012). url : https:
//rt.wiki.kernel.org/index.php/CONFIG_PREEMPT_RT_Patch.

[15]« Open Systems and the Corresponding Interfaces for Automotive Electronics ». In : ().
url : http://www.osekvdx.org/os20r1a.pdf.

[16] *OSE Realtime Kernel*. url : http://www.ose.com/PDF/rtk.pdf.

[17]P. Mantegazza P. Gerum K. Yaghmour.
« A novel approach to real-time Free Software ». In : (2002).
url : https://lwn.net/Articles/1222/.

[19]S. Papacharalambous P. Mantegazza E. L. Dozio.
« RTAI : Real Time Application Interface ».
In : *Linux Journal* 2000 (2000).
url : http://dl.acm.org/citation.cfm?id=348554.348564.

[20]Prof. Ajit Pal.
Video lectures on " fticroprocessors and fticrocontrollers ".
url :https://www.youtube.com/watch?v=rpdygqOI9mM.

[21]Caroline Yao Qing Li.
Real-Time Concepts for Embedded Systems. CMP Books, 2003.
url : http://www.embeddedlinux.org.cn/rtconforembsys/
5107final/LiB0001.html.

[22] *Real-Time Systems with fticrosoft Windows CE*.
url : http://www.eu.microsoft.com/windows/embedded/ce/
resources/howitworks/realtime.asp.

[23]Cottet Francis Saad Bouzefrane Samia.
« Etude temporelle des applications temps réel distribuées à contraintes strictes basée sur une analyse d'ordonnançabilité = A temporal study of distributed hard real-time applications based on a schedulability analysis ». Thèse de doct. 1998.

[24]Jacques Mossière Sacha Krakowiak.
« la naissance des systèmes d'exploitation ». In : (2013).
url : https://interstices.info/jcms/nn_72288/la-naissance-des-systemes-d-exploitation.

[25]PH D. TIMOTHY P. *Linux secrets d'experts*. 1997.

[26]M. Barabanov V. Yodaiken. « A Real-Time Linux ». In : *Linux Journal* 34 (1997). url : http://users.soe.ucsc.edu/~sbrandt/courses/Winter00/290S/rtlinux.pdf.

[27] *VRTX Real-Time Operating System*. url : http://www.mentorgraphics.com/embedded/vrtxos/.

[28] *VxWorks Programmers Guide*. url : http://www.wrs.com/pdf/vxworksguide.pdf.

[29]Clark Williams. *An Overview of Realtime Linux*. url : http://people.redhat.com/bche/presentations/realtime-linux-summit08.pdf.

[30] *Xenomai : Real-Time Framework for Linux*. url : http://www.xenomai.org/.

Annexe A

Liste des Abréviations

RTOS	**R**eal Time **O**perating **S**ystem
SRT	**S**oft **R**eal Time
HRT	**H**ard **R**eal **T**ime
SE	**S**ystème d'**E**xploitation
CPU	**C**entral **P**rocessing **U**nit
POSIX	**P**ortable **O**perating **S**ystem **I**nterface on UNIX
FIFO	**F**irst **I**n **F**irst **O**ut
LILO	**LI**nux **LO**ader
RTM	**R**a**T**e **M**onotonic
TRON	**T**he **R**eal **T**ime **O**perating **S**ystem **N**ucleus
OSE	**O**perating **S**ystem **E**mbedded
OSEK	**O**ffene **S**ysteme und **S**chnittstellen für **E**lektronik im **K**raftfahrzeug
VDX	**V**ehicle **D**istributed e**X**ecutive
RTLinux	**R**eal **T**ime **Linux**
RTAI	**R**eal **T**ime **A**pplication **I**nterface
VRTX	**V**ersatile **R**eal **T**ime E**x**ecutive
FAA	**F**ederal **A**viation **A**dministration
GPL	**G**eneral **P**ublic **L**icense
API	**A**pplication **P**rogramming **I**nterface
TR	**T**emps **R**éel

Résumé

Ce travail rentre dans le cadre du développement des logiciels dits Open Source. Dans ce contexte, le système d'exploitation GNU/Linux séduit énormément d'entreprises indus-trielles notamment avec sa composante Temps Réel. Nous présenterons une mise en œuvre complète de ce pouvoir de linux, utilisant sa version RedHat et reposant sur l'environnement RTLinux. De ce fait Linux peut se prévaloir d'être un concurrent pour de nombreux systèmes d'exploitation Temps Réel commerciaux. L'objectif de ce projet est de mener un développement en interaction forte avec le maté-riel sous GNU/Linux en tant que système embarqué. En guise de mise en œuvre de l'environnement obtenu, nous allons construire un driver de pilote simple, montrer l'utilisation des ressources RTLinux et écrire les programmes de test pour ce pilote.

Mots clés : Système Temps Réel, RTOS, Linux, RTLinux, pilote de périphériques.

Abstract

This work falls within the scope of an Open source software development. In this context, the op-erating system GNU / LINUX is immensely appealing to many industrial companies especially with its Real-time component. We shall present a complete implementation of this feature using the RedHat Linux paired with the RTLinux environment. Thus Linux proved to be a great competitor to numerous commercial Real-time operating systems. The aim of this project is to develop a strongly interacting software with equipment under GNU / Linux as an embedded system. In order to concretize the implementation of the final environment, we built a simple driver ; demonstrate the use of RTLinux resources and to write the test program for this driver.

Key words : Real Time System, RTOS, Linux, RTLinux, peripheral driver.

ملخص

يدخل هذا العمل في إطار البرامج المفتوحة(Open source) ، وفي هذا السياق، يقوم نظام التشغيل GNU/Linux بإغراء الكثير من الشركات الصناعية، بالأخص قدرته على تنفيذ الوقت الحقيقي (Real Time). سنقوم بعرض كامل لتطبيق هذه الخاصية ل Linux وذلك تحت نظام التشغيل RedHat و المحيط RTLinux. وعليه، فان Linuxيعد منافسا لأنظمة التشغيل ال RTOS المسوقة.

ان الهدف من هذا العمل هو تحقيق برنامج شديد التفاعل مع الاجهزة تحت GNU/Linuxكنظام تشغيل مطمور. وعلى سبيل المتحصل عليه، سنبني سائق للأنظمة المحيطة، نبرز استخدام موارد RTLinux ونكتب برنامج بسيط لاختبار السائق (driver)،

الكلمات الدالة: نظام الوقت الحقيقي ،سائق للأنظمة المحيطة، RTOS، RTLinux، Linux