

L2.1 - Document Cas d'Utilisations et Exigences Applicables à Linux Temps Reel



Document rédigé par :	Approuvé par :	Approuvé par :	Approuvé par :
Nicolas Raynaud (Sagem Communications) Olivier Gallot (Axupteam)			

Liste des évolutions

Edition	Date	Evolutions

Table des matières

1 - Introduction.....	6
2 - Organisation du document.....	6
3 - Quelques mots sur le temps réel.....	6
3.1 - Déterminisme fonctionnel.....	6
3.2 - Déterminisme temporel.....	6
3.2.1 - Notion d'échéance.....	7
3.2.1.1 - Échéance de début.....	7
3.2.1.2 - Échéance de fin.....	7
3.2.1.3 - Échéance dures, échéances molles.....	7
3.2.2 - Notion de latence.....	8
3.2.3 - Temps réel dur et temps réel mou.....	8
3.3 - Qualité de service.....	8
3.4 - Sureté de fonctionnement	8
3.4.1 - Disponibilité.....	8
3.4.2 - Fiabilité, Robustesse.....	8
3.4.3 - Sureté de fonctionnement.....	8
3.4.4 - Sécurité.....	9
4 - L'enquête	9
4.1 - Les systèmes temps réel chez Sagem Communications.....	9
4.2 - Problèmes temps réel rencontrés avec des linux standards.....	9
4.3 - Exigences générales exprimées par les équipes produits.....	10
4.4 - Potentiel du temps réel dans les produits	10
4.4.1 - Fonctions internes des produits embarqués.....	10
4.4.2 - Fonctions réseaux.....	11
5 - Énumération des exigences.....	12
5.1 - Exigences communes.....	12
5.1.1 - Documentation.....	12
5.1.2 - Audit.....	12
5.1.3 - Statistiques.....	12
5.1.4 - Chronologie.....	12
5.1.5 - Génération	13
5.2 - Xenomai.....	13
5.2.1 - Skins.....	13
5.2.2 - Couches logicielles temps réel.....	13
5.2.3 - Debug.....	13
5.3 - PREEMPT_RT.....	13
5.4 - POSIX.....	14
5.5 - Tests et analyse.....	14
6 - Notation des exigences.....	15
7 - Modélisation des cas d'utilisation.....	18
7.1 - Modèle de traitement de flux de données.....	18

7.1.1 - Détermination des dates de consommation.....	18
7.1.2 - limitation de la bufferisation	19
7.1.3 - Structure d'une tache intermédiaire.....	19
7.1.4 - Communication entre les taches.....	20
7.1.4.1 - Fifo d'entrée.....	20
7.1.4.2 - Fifo de sortie.....	20
7.1.4.3 - Implémentation des fifos	20
7.1.5 - Organisation du flux en paquets de données.....	21
7.1.5.1 - Implémentation des buffers.....	21
7.1.6 - Taches de production et consommation du traitement du flux.....	21
7.1.6.1 - Production.....	21
7.1.6.2 - Consommation.....	22
7.1.6.3 - Temps de traitement.....	22
7.1.6.4 - Gigue.....	22
7.1.6.5 - Histogramme.....	22
7.1.7 - Paramétrages du traitement de flux.....	22
7.1.7.1 - Multi instances.....	22
7.1.7.2 - Priorités.....	22
7.1.7.3 - Mode de réveil.....	22
7.1.7.4 - Tache de consommation du flux.....	23
7.1.7.5 - Tache de production du flux.....	23
7.1.7.6 - Profondeur des Fifos.....	23
7.1.7.7 - Histogramme.....	23
7.2 - Concurrence de flux programmés.....	23
7.2.1 - Méthode de programmation des paquets.....	23
7.2.2 - Concurrence de flux.....	25
7.3 - Stress du CPU et de la mémoire de masse.....	26
7.3.1 - Paramétrage de la tache de stress.....	26
7.3.1.1 - Instances d'occupation CPU.....	26
7.3.1.2 - Nombre d'instances d'écriture et taille des blocs.....	26
7.3.1.3 - Partitionnement de la mémoire de masse.....	27
7.4 - Estimation de charge CPU.....	27
7.4.1 - Charge globale.....	27
7.4.2 - Charge des taches.....	27
7.5 - Réception d'un signal sur un GPIO.....	27
7.6 - Production d'un signal sur un GPIO.....	28
7.7 - Le temps réel et le réseau.....	29
7.7.1 - Flux de VOIP.....	29
7.7.1.1 - Modélisation des flux RTP.....	29
7.7.1.2 - Mesures.....	30
7.7.1.3 - Perturbation de la VOIP.....	32
7.7.1.4 - Synchronisation des machines	33
7.7.2 - Accès au réseau via la stack réseau standard.....	34
7.7.3 - Accès au réseau via RTNet.....	35
7.8 - Driver dans l'espace utilisateur ULDD	36
8 - Description des uses cases.....	37
8.1 - Audit et statistiques.....	37

8.1.1 - Mesure de charge et temps CPU.....	37
8.1.2 - Mesures de retards.....	37
8.1.3 - Mesures de giges.....	37
8.1.4 - Production de l'histogramme des latences.....	38
8.1.5 - Chronogramme.....	38
8.2 - Traitement de flux.....	39
8.2.1 - Occupation complète du CPU (UCR 1).....	39
8.2.2 - Préservation du temps CPU attribué aux taches temps réel lors de stress non temps réel.....	40
8.2.3 - Stress temps réel et sauvegarde des caractéristiques des taches temps réel prioritaires.....	41
8.2.4 - Répartition équitable des ressources CPU entre les taches de même priorité.....	42
8.2.5 - Concurrence d'un traitement de flux avec une tache très prioritaire.....	42
8.2.6 - Priorités des taches et garantie des performances.....	43
8.2.7 - Taches temps réel et priorité d'accès à la mémoire de masse.....	43
8.2.8 - Ressources partagées entre taches temps réel.....	44
8.2.9 - Prémption d'une écriture sur un file system.....	44
8.2.10 - IPC.....	45
8.2.11 - Communication entre les taches de l'espace utilisateur et noyau.....	45
8.2.12 - Consommation à date précise.....	46
8.2.13 - Synchronisation de flux.....	46
8.2.14 - Traitement de flux et écriture sur flash.....	47
8.3 - Driver dans l'espace utilisateur (ULDD).....	47
8.4 - Acquisition usb.....	48
8.5 - Signal sur un GPIO.....	49
8.6 - Accès à un GPIO non interrompu.....	50
8.7 - Timer de haute résolution.....	50
8.8 - Asservissement.....	51
8.9 - Temps réel et réseau.....	53
8.9.1 - Utilisation de la stack Linux.....	53
8.9.1.1 - Concurrence de flux d'applications temps réel.....	53
8.9.1.2 - Concurrence de flux d'applications temps réel de même priorité.....	54
8.9.1.3 - Concurrence de flux d'applications temps réel avec ceux d'applications non temps réel.....	54
8.9.1.4 - Concurrence de flux routés ou bridgés au travers de la machine.....	55
8.9.2 - Utilisation de RTNet.....	56
8.9.2.1 - Concurrence de flux d'applications temps réel avec RTNet.....	56
8.9.2.2 - Concurrence de flux d'applications temps réel de même priorité avec RTNet.....	57
9 - Notation des cas d'utilisation.....	59

1 - Introduction

Ce document est le livrable « L2.1 : Document Cas d'Utilisations et Exigences applicables à Linux Temps Réel » associé au sous projet SP2 du projet RTEL4i.

Basé, entre autres sur des entretiens avec des responsables de projets de Sagem Communications qui apportent le point de vue utilisateur, il présente la formalisation des besoins des utilisateurs du temps réel sur un embarqué par la description de cas d'utilisations et la formulation des exigences.

2 - Organisation du document

Ce document présente dans un premier temps quelques notions de temps réel qui seront réutilisées par la suite.

Il présente ensuite une synthèse des résultats recueillis pendant l'enquête auprès de diverses équipes de Sagem Communications. Ces résultats sont ensuite présentés comme une liste d'exigences d'une part et une liste de cas d'utilisation d'autre part.

Les cas d'utilisation réels étant difficiles à présenter du fait de la grande variété des matériels embarqués, il a été décidé de les modéliser de façon à mettre en exergue chacune des problématiques temps réel. Les cas d'utilisation sont alors présentés comme une composition de divers modèles.

3 - Quelques mots sur le temps réel

Bien qu'inconnus du grand public, les systèmes temps réel sont de plus en plus présents dans notre environnement. On les trouve, embarqués dans des systèmes dédiés, dans des domaines comme le transport ou les télécoms mais aussi en production industrielle ou dans le monde médical.

Les industriels se fient aux systèmes temps réel pour procurer à des équipements confrontés à des mondes réels où les dates des événements et leur ordre ne peut être prévus, déterminisme fonctionnel, déterminisme temporel, qualité de service et sûreté de fonctionnement.

3.1 - Déterminisme fonctionnel

Le déterminisme fonctionnel garantit les mêmes sorties en réponse aux mêmes stimuli. Il est important de noter que l'exactitude du résultat dépend aussi bien du calcul que du moment où il a été produit.

3.2 - Déterminisme temporel

Le déterminisme temporel garantit que les réactions se font toujours en respectant les contraintes temporelles imposées au système. Il enlève toute incertitude sur le comportement individuel de chacune des tâches ou sur le comportement global de l'ensemble des tâches alors qu'elles fonctionnent ensemble.

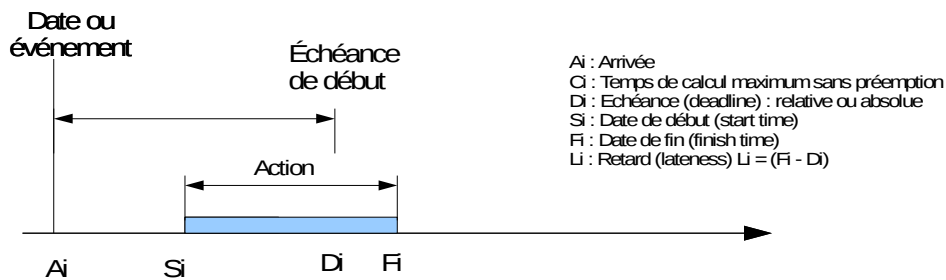
L'échéance, caractéristique du monde temps réel, est une des ces contraintes temporelles.

3.2.1 - Notion d'échéance

On appelle échéance une date à laquelle expire un délai.

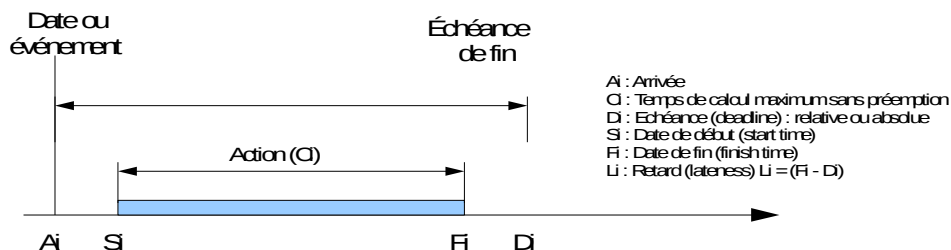
3.2.1.1 - Échéance de début

Une échéance peut définir la date à laquelle une action doit être commencée. Elle peut être la base d'un contrat entre une tâche et le système : La tâche demande au système d'être réveillée afin de pouvoir réagir, à un événement ou à une date donnée, dans le cadre d'une échéance.



3.2.1.2 - Échéance de fin

L'échéance peut aussi définir la date à laquelle une action doit être finie.



C'est plutôt à ce type d'échéances que les développeurs s'intéressent afin de garantir le déterminisme temporel d'un système. L'estimation de la distribution des retards L_i d'une tâche peut donner de bonnes indications sur les capacités du système en vue d'une évolution.

3.2.1.3 - Échéance dures, échéances molles

Dans le monde du temps réel, les échéances sont classées en fonction des conséquences qu'aurait le non-respect du contrat qu'elles représentent :

- Les échéances dures : Le non respect d'une échéance dure peut avoir des effets catastrophiques et engendre une exception pour un traitement d'erreur.
- Les échéances molles : Leur non-respect, dans certaines limites, de l'échéance peut être gênant mais pas catastrophique. il n'engendre pas d'exception.

3.2.2 - Notion de latence

La latence désigne le temps écoulé entre la fin d'un événement et le début de la réaction à celui-ci. Elle mesure les temps de traitement des données et des événements au sein d'un système tel que le temps de réponse à une interruption par exemple.

Un appareillage électronique (oscilloscope, analyseur logique) ou une instrumentation du code sont généralement utilisés pour mesurer la latence.

Enfin la gigue mesure la variation entre les valeurs minimale et maximale d'une latence.

3.2.3 - Temps réel dur et temps réel mou

La notion de temps réel est souvent confondue avec la capacité de répondre rapidement à des événements périodiques ou apériodiques. Dans le monde du temps réel, minimiser la moyenne des temps de réponse ne suffit pas. Il s'agit de satisfaire les échéances de toutes les tâches individuelles.

Les systèmes **temps réel dur** garantissent que, dans un environnement borné, ils satisferont toutes les échéances de toutes les tâches individuelles le nécessitant (les échéances dures). La criticité est évidente : « ca passe ou ca casse ».

Les systèmes **temps réel mou** ou souple ne donnent pas cette garantie. Ils ne gèrent pas d'échéances dures mais seulement des échéances molles. Ils seront dit d'autant meilleurs qu'ils seront capables de s'approcher des capacités d'un système temps réel dur. La criticité dépend du contexte et de ce que les utilisateurs sont prêts à accepter.

3.3 - Qualité de service

La qualité de service des systèmes temps réel permet d'assurer des caractéristiques de délai et de régularité aux services embarqués.

Elle peut être mise en œuvre par les systèmes temps réel grâce à leurs propriétés particulières.

Dans la mesure où les systèmes embarqués fournissent du multiservice, il doivent être capables d'assurer une bonne attribution des ressources afin de bien gérer la concurrence et les synchronisations de tâches. Ceci pour que tout ce qu'il y a à faire soit fait dans les délais impartis.

3.4 - Sureté de fonctionnement

3.4.1 - Disponibilité

Le système doit pouvoir être utilisé à tout moment. Ce sont souvent les procédures d'entretien qui rendent le système indisponible.

3.4.2 - Fiabilité, Robustesse

Le système doit pouvoir fonctionner dans des conditions dégradées. Il doit être capable de survivre aux divers aléas de l'environnement.

3.4.3 - Sureté de fonctionnement

Même quand le système fonctionne en mode dégradé, ses actions ne peuvent pas engendrer d'effets catastrophiques.

La sûreté de fonctionnement ne peut être fournie que par un travail d'ingénierie approfondi commençant par une spécification non ambiguë des hypothèses. Elle s'appuie également sur les propriétés d'un système temps réel dur.

3.4.4 - Sécurité

La sécurité consiste à assurer que les ressources mises en œuvre sont exploitées uniquement par des acteurs ayant les droits suffisants.

4 - L'enquête

L'enquête a été réalisée auprès de quelques équipes Sagem Communications afin de collecter leurs besoins applicables au temps réel.

Ces équipes font des produits variés, aussi bien des gateway que des set-up box ou des systèmes de gestion de feux rouges.

4.1 - Les systèmes temps réel chez Sagem Communications

Un rapide tour des produits nous montre que, même si Linux est de plus en plus présent dans les systèmes embarqués de la société, il n'y a pas d'utilisation de distributions Linux temps réel. Quelques systèmes temps réels non linux sont présents, tels que VxWorks ou OS21.

Les raisons de la non-utilisation de ces distributions Linux temps réel tiennent aussi bien de la non-connaissance de l'existence de ces technologies, que de la difficulté de mise en œuvre ou de l'absence de distribution pour l'architecture visée. Toutes ces raisons font que, du point de vue produit, l'investissement peut sembler trop élevé pour un résultat incertain.

Bien que n'utilisant pas de distribution linux temps réel, certaines équipes ont une vraie compétence acquise dans le passé sur des distributions non-Linux.

4.2 - Problèmes temps réel rencontrés avec des linux standards

Manque de déterminisme

De manière générale, les produits, utilisant Linux, souffrent du manque de déterminisme de ce système d'exploitation. Le retard, la latence et la gigue y sont des problèmes récurrents.

De plus, l'ajout de nouvelles fonctionnalités peut déstabiliser l'aspect multiservice d'un système. Il n'y a rien de connu, sinon l'expérience, pour estimer si une évolution logicielle est susceptible de déstabiliser les propriétés temporelles de l'existant.

Des aspects temps réel fournis par la puissance du CPU

Lorsque Linux est utilisé, les aspects « temps réel » de l'embarqué sont fournis le plus souvent par la puissance du CPU. Celui-ci est quelquefois assisté par des coprocesseurs qui fournissent une grande part de la capacité de réactivité du système. La résolution des divers problèmes de temps réel est faite par des tuning ou des développements demandant du temps et de l'énergie.

Certains services sont de gros consommateurs de CPU

Un produit requiert le fonctionnement simultané de plusieurs services sur une plateforme: c'est la notion de multi services. Des services sont quelquefois particulièrement gourmands en terme de consommation CPU et fonctionnent alors au détriment des autres services.

Pour être réactifs, certains services doivent être dans le monde kernel

Dans les cas extrêmes, afin d'obtenir de bonnes performances, le code devant fournir le temps réel doit être porté ou écrit dans le monde kernel risquant ainsi d'être exposé à la licence du noyau.

Pas de vue du déroulement des traitements

Lors des investigations pour identifier les causes d'un dysfonctionnement lors du multi services, il est courant que les développeurs se fassent une idée du déroulement réel des traitements dans l'embarqué par des expérimentations et des instrumentations du code. Ce sont des tâches délicates où il arrive que l'instrumentation cache les problèmes.

4.3 - Exigences générales exprimées par les équipes produits

Les deux premières exigences sont de pouvoir assurer l'ensemble des fonctionnalités et des performances que doivent faire le produit.

Un produit et donc le système d'exploitation sur lequel il s'appuie doivent être robustes. En effet la majeure partie des systèmes sont embarqués et doivent être autonomes de façon à réduire au maximum les interventions humaines d'administration.

Dans la plupart des cas, ce sont les événements du monde extérieur qui pilotent les systèmes. Ces événements peuvent être multiples et de natures variées pour un équipement. Il arrive également, mais plus rarement, que certaines tâches des systèmes soient pilotées par le temps. Le produit doit être capable de réagir dans les délais les meilleurs à ces événements matériels et temporels.

La majorité des produits ont obligation de répondre systématiquement et correctement à des événements en respectant des contraintes d'échéance.

Les produits souhaitent également assurer les performances maximum sans dégradation de la qualité de service.

Un produit est rarement fait en un seul jet. Pendant toute sa vie commerciale, il doit pouvoir s'enrichir. Il est donc important, afin de réduire les coûts et les délais de développement, de pouvoir ajouter des fonctionnalités facilement sans pour autant déséquilibrer ce qui fonctionne déjà.

4.4 - Potentiel du temps réel dans les produits

Les domaines où le temps réel peut être pertinents dans les produits Sagem Communications sont multiples. Ces produits sont de plus en plus complexes, ils intègrent des composants logiciels de multiples sources qui, typiquement, n'ont pas été écrits pour fonctionner ensemble. Or, la plupart de ces produits sont multi-services. En respectant ces contraintes, une intégration de ces composants dans des linux standards devient vite une tâche empirique s'appuyant sur l'expérience de l'intégrateur. Il semble évident que le temps réel peut avoir sa place dans les produits.

4.4.1 - Fonctions internes des produits embarqués

L'utilisation la plus évidente du temps réel dans les embarqués concerne bien sur le contrôle du temps : être capable d'exécuter des traitements à des instants datés très précis. Ainsi les setup box sont capables de reproduire les séquences d'images de films de façon continue et très régulière. Le moindre écart est visible par l'utilisateur.

Une autre utilisation basée sur le contrôle du temps concerne la gestion de composants spécifiques de feux rouges devant s'assurer de la bonne exécution d'un ordre sous 300 millisecondes.

Un autre type de traitement rencontrés notamment dans les setup box concerne les traitements de flux de données soutenus sous la forme de collaboration entre des tâches. Pour chaque paquet, une succession de traitements sont à appliquer par des composants spécifiques. Le flux de données permet à plusieurs paquets de subir simultanément différentes phases des traitements comme le fait le pipeline d'un processeur.

4.4.2 - Fonctions réseaux

Lors de l'enquête, nous n'avons pas vu de cas où des produits Sagem Communications mettaient en œuvre de réseau temps réel.

Les produits sont pour la plupart connectés au réseau local de l'utilisateur et quelquefois au réseau WAN d'un fournisseur d'accès. Ces réseaux ne fournissent que rarement de la qualité de service. La plupart de ces produits offrent des services nécessitant une certaine qualité de service ne pouvant par être fournie par IP. Parmi eux, il est possible de citer :

- la VOIP
- la TOIP
- le streaming
- les flux internet que l'on qualifiera de standard HTTP, FTP, SMTP, POP...

Cette liste de services n'est pas exhaustive. Certains produits utilisent des protocoles plus exotiques mais aussi sensibles aux aspects temporels.

Les difficultés de gestion de ces flux viennent surtout du fait que :

- les médias peuvent avoir des capacités asymétriques (par exemple ADSL environ 22Mbps en downstream et quelque Méga en upstream).
- La nature des flux peut également être asymétrique. La télévision sur IP voit la majorité de son flux venir directement du réseau opérateur. Suivant la qualité de la télévision, les débits peuvent aller de 4Mbps à 12Mbps par chaînes visionnées. Or certains cas d'utilisation de la télévision sur IP mettent en œuvre plusieurs setup-box et autorisent l'enregistrement d'un programme pendant qu'un autre est visionné.
- Des flux « streamés », comme ceux de la VOIP, abordant une interface réseau à la capacité contrainte subissent de plein fouet la concurrence de flux envoyés sous forme de salves. L'absence d'un accès privilégié à l'interface réseau impose la mise en œuvre de scheduler de qualité de service réseau.
- Tout ces flux, ayant de fortes contraintes, doivent pouvoir fonctionner de façon concurrentes. Il est demandé à chacun que les trames prioritaires atteignent leurs cibles dans les temps qui lui sont impartis.

5 - Énumération des exigences

5.1 - Exigences communes

- EX_ 1 : Pouvoir facilement mettre en œuvre le temps réel.
- EX_ 2 : Disposer d'un système robuste
- EX_ 3 : Ne pas perdre les performances en introduisant le temps réel.
- EX_ 4 : Conserver les propriétés temporelles d'un système lors de l'ajout d'une application temps réel si la capacité de la machine le permet.
- EX_ 5 : Pouvoir piloter une application par le temps
- EX_ 6 : Pouvoir piloter une application par les événements
- EX_ 7 : Disposer de timer avec une précision de l'ordre de la nano seconde
- EX_ 8 : Avoir des latences de l'ordre de la microseconde.

5.1.1 - Documentation

- EX_ 9 : Avoir une documentation détaillée des API de programmation
- EX_ 10 : Avoir une documentation détaillée des outils de configuration et d'administration

5.1.2 - Audit

- EX_ 11 : Pouvoir énumérer les tâches temps réel
- EX_ 12 : Pouvoir connaître la priorité d'une tâche
- EX_ 13 : Pouvoir connaître l'état d'une tâche
- EX_ 14 : Pouvoir savoir de quel processus Linux dépend une tâche

5.1.3 - Statistiques

- EX_ 15 : Pouvoir mesurer la charge de chacune des tâches temps réel
- EX_ 16 : Pouvoir mesurer la charge de l'ensemble des tâches temps réel
- EX_ 17 : Pouvoir mesurer la mémoire occupée par chacune des tâches temps réel
- EX_ 18 : Pouvoir compter les erreurs de gestion de queue de messages
- EX_ 19 : Pouvoir compter, pour chaque tâche, le nombre de fois ou une échéance a été manquée
- EX_ 20 : Pouvoir mesurer la gigue moyenne au réveil de chacune des tâches (variance/écart-type)
- EX_ 21 : Pouvoir mesurer le retard négatif moyen de chacune des tâches périodiques (variance/écart-type)

5.1.4 - Chronologie

- EX_ 22 : Pouvoir visualiser la chronologie des événements
- EX_ 23 : Pouvoir visualiser les suivis temporel de la charge de chaque tâche
- EX_ 24 : Pouvoir visualiser les suivis temporel de l'ensemble des charges

5.1.5 - Génération

- EX_ 25 : Pouvoir ajouter simplement le co noyau temps réel xenomai au noyau linux.
- EX_ 26 : Pouvoir choisir l'algorithme d'ordonnancement (EDF, RM (Rate Monothonic))
- EX_ 27 : Pouvoir utiliser un algorithme d'allocation mémoire de complexité O(1). (tlsf memory allocator).
- EX_ 28 : Pouvoir traiter une interruption dans une fonction de l'espace utilisateur.

5.2 - Xenomai

- EX_ 29 : Disposer du temps réel dur dans le monde user
- EX_ 30 : Pouvoir faire communiquer les taches xenomai avec des processus linux au travers de fifo.
- EX_ 31 : Pouvoir faire communiquer les taches xenomai avec des processus linux au travers de mémoires partagées.
- EX_ 32 : Pouvoir garder sa priorité face aux autres taches primaires lorsqu'une tache primaire Xenomai effectue un appel système linux
- EX_ 33 : Pouvoir utiliser l'API comedi (Control and Measurement Device Interface) au sein de taches primaires de xenomai

5.2.1 - Skins

- EX_ 34 : Pouvoir utiliser la skin POSIX de Xenomai sur les plateformes de références.
- EX_ 35 : Pouvoir utiliser la skin VxWorks de Xenomai sur les plateformes de références.
- EX_ 36 : Pouvoir utiliser la skin Psos de Xenomai sur les plateformes de références.
- EX_ 37 : Pouvoir utiliser la skin vrtx de Xenomai sur les plateformes de références.

5.2.2 - Couches logicielles temps réel

- EX_ 38 : Pouvoir utiliser un controleur gigabit ethernet en temps réel (rtnet,xenomai).
- EX_ 39 : Pouvoir utiliser le protocole spi en temps réel (spi,xenomai)
- EX_ 40 : Pouvoir utiliser un controleur usb (uhci et ehci) en temps réel (usb4rt,xenomai).

5.2.3 - Debug

- EX_ 41 : Pouvoir tracer l'exécution d'Adeos I pipe
- EX_ 42 : Pouvoir tracer le code de xenomai avec LTT.
- EX_ 43 : Pouvoir debugger les taches xenomai (espace utilisateur,espace noyau),(gdb,kgdb)

5.3 - PREEMPT_RT

- EX_ 44 : Pouvoir décider qu'une section critique du code noyau soit préemptible ou non (CONFIG_PREEMPT_RT spinlock_t,rwlock_t vs rawspinlock_t).
- EX_ 45 : Pouvoir hériter de la priorité de la tache qui attend la libération d'un verrou (mutex et sémaphore). (CONFIG_PREEMPT_RT,inversion de priorité)

EX_ 46 : Pouvoir préempter les handlers d'interruption (CONFIG_PREEMPT_RT, isr as preemptible kernel thread).

5.4 - POSIX

EX_ 47 : Disposer de timers POSIX pour les plateformes cibles de références avec une précision de l'ordre de quelques nano secondes.

EX_ 48 : Pouvoir hériter de la priorité de la tâche qui attend la libération d'un verrou (mutex et sémaphore) dans l'espace utilisateur pour la glibc ou la uclibc (PTHREAD_PRIO_INHERIT)

EX_ 49 : Pouvoir utiliser les fonctions i/o asynchrones avec la glibc ou la uclibc

EX_ 50 : Pouvoir gérer des horloges avec la glibc ou la uclibc

EX_ 51 : Pouvoir utiliser des mémoires partagées avec la glibc ou la uclibc

EX_ 52 : Pouvoir utiliser des bibliothèques partagées sur des systèmes sans mmu avec la glibc ou la uclibc

EX_ 53 : Pouvoir utiliser les timers POSIX en mode périodique et apériodique sans perte de résolution.

EX_ 54 : Pouvoir compiler du code « relogeable » (option pic) sur les plateformes de références et générer un binaire au format flat (nommu).

EX_ 55 : Pouvoir compiler les bibliothèques partagées pour XIP (Execute In Place) sur les plateformes de références avec la glibc ou la uclibc

5.5 - Tests et analyse

EX_ 56 : Pouvoir utiliser la détection de problème de synchronisation au sein du noyau et analyser la prise de verrou. (deadlock ,lockdep)

EX_ 57 : Pouvoir tester (benchmarking tools [interbench, netperf, hackbench, dbench, stress]) les latences dans le système, la performance de l'ordonnanceur, mesurer la performance du réseau.

EX_ 58 : Pouvoir mesurer le temps de changement de contexte. (lmbench)

EX_ 59 : Pouvoir mesurer la performance des IPC.(lmbench)

EX_ 60 : Pouvoir tracer le temps passé avec les interruptions désactivées, la préemption désactivée et lors des changements de contexte. (CONFIG_SCHED_TRACER)

6 - Notation des exigences

Ce paragraphe décrit l'importance du besoin associé à chacune des exigences

Nom de l'exigence	besoin
EX_ 1 : Pouvoir facilement mettre en œuvre le temps réel.	fort
EX_ 2 : Disposer d'un système robuste	fort
EX_ 3 : Ne pas perdre les performances en introduisant le temps réel.	fort
EX_ 4 : Conserver les propriétés temporelles d'un système lors de l'ajout d'une application temps réel si la capacité de la machine le permet.	fort
EX_ 5 : Pouvoir piloter une application par le temps	fort
EX_ 6 : Pouvoir piloter une application par les événements	fort
EX_ 7 : Disposer de timer avec une précision de l'ordre de la nano seconde	fort
EX_ 8 : Avoir des latences de l'ordre de la microseconde.	fort
EX_ 9 : Avoir une documentation détaillée des API de programmation	fort
EX_ 10 : Avoir une documentation détaillée des outils de configuration et d'administration	fort
EX_ 11 : Pouvoir énumérer les taches temps réel	fort
EX_ 12 : Pouvoir connaître la priorité d'une tache	fort
EX_ 13 : Pouvoir connaître l'état d'une tache	fort
EX_ 14 : Pouvoir savoir de quel processus Linux dépend une tache	fort
EX_ 15 : Pouvoir mesurer la charge de chacune des taches temps réel	fort
EX_ 16 : Pouvoir mesurer la charge de l'ensemble des taches temps réel	fort
EX_ 17 : Pouvoir mesurer la mémoire occupée par chacune des taches temps réel	fort
EX_ 18 : Pourvoir compter les erreurs de gestion de queue de messages	fort
EX_ 19 : Pouvoir compter, pour chaque tache, le nombre de fois ou une échéance a été manquée	moyen
EX_ 20 : Pouvoir mesurer la gigue moyenne au réveil de chacune des taches (variance/écart-type)	moyen
EX_ 21 : Pouvoir mesurer le retard négatif moyen de chacune des taches périodiques (variance/écart-type)	fort
EX_ 22 : Pouvoir visualiser la chronologie des événements	fort
EX_ 23 : Pouvoir visualiser les suivis temporel de la charge de chaque tache	fort
EX_ 24 : Pouvoir visualiser les suivis temporel de l'ensemble des charges	fort
EX_ 25 : Pouvoir ajouter simplement le co noyau temps réel xenomai au noyau linux.	fort
EX_ 26 : Pouvoir choisir l'algorithme d'ordonnancement (EDF, RM (Rate	moyen

EX_ 27 : Pouvoir utiliser un algorithme d'allocation mémoire de complexité O(1). (tlf memory allocator).	moyen
EX_ 28 : Pouvoir traiter une interruption dans une fonction de l'espace utilisateur.	fort
EX_ 29 : Disposer du temps réel dur dans le monde user	fort
EX_ 30 : Pouvoir faire communiquer les taches xenomai avec des processus linux au travers de fifo.	moyen
EX_ 31 : Pouvoir faire communiquer les taches xenomai avec des processus linux au travers de mémoires partagées.	faible
EX_ 32 : Pouvoir garder sa priorité face aux autres taches primaires lorsqu'une tache primaire Xenomai effectue un appel système linux	moyen
EX_ 33 : Pouvoir utiliser l'API comedi (Control and Measurement Device Interface) au sein de taches primaires de xenomai	faible
EX_ 34 : Pouvoir utiliser la skin POSIX de Xenomai sur les plateformes de références.	fort
EX_ 35 : Pouvoir utiliser la skin VxWorks de Xenomai sur les plateformes de références.	fort
EX_ 36 : Pouvoir utiliser la skin Psos de Xenomai sur les plateformes de références.	moyen
EX_ 37 : Pouvoir utiliser la skin vrtx de Xenomai sur les plateformes de références.	moyen
EX_ 38 : Pouvoir utiliser un contrôleur gigabit ethernet en temps réel (rtnet,xenomai).	moyen
EX_ 39 : Pouvoir utiliser le protocole spi en temps réel (spi,xenomai)	faible
EX_ 40 : Pouvoir utiliser un contrôleur usb (uhci et ehci) en temps réel (usb4rt,xenomai).	faible
EX_ 41 : Pouvoir tracer l'exécution d'Adeos I pipe	fort
EX_ 42 : Pouvoir tracer le code de xenomai avec LTT.	fort
EX_ 43 : Pouvoir debugger les taches xenomai (espace utilisateur,espace noyau), (gdb,kgdb)	fort
EX_ 44 : Pouvoir décider qu'une section critique du code noyau soit préemptible ou non (CONFIG_PREEMPT_RT spinlock_t,rwlock_t vs rawspinlock_t).	moyen
EX_ 45 : Pouvoir hériter de la priorité de la tache qui attend la libération d'un verrou (mutex et sémaphore). (CONFIG_PREEMPT_RT,inversion de priorité)	moyen
EX_ 46 : Pouvoir préempter les handlers d'interruption (CONFIG_PREEMPT_RT,isr as preemptible kernel thread).	moyen
EX_ 47 : Disposer de timers POSIX pour les plateformes cibles de références avec une précision de l'ordre de quelques nano secondes.	moyen

EX_ 48 : Pouvoir hériter de la priorité de la tâche qui attend la libération d'un verrou (mutex et sémaphore) dans l'espace utilisateur pour la glibc ou la uclibc (PTHREAD_PRIO_INHERIT)	faible
EX_ 49 : Pouvoir utiliser les fonctions i/o asynchrones avec la glibc ou la uclibc	fort
EX_ 50 : Pouvoir gérer des horloges avec la glibc ou la uclibc	fort
EX_ 51 : Pouvoir utiliser des mémoires partagées avec la glibc ou la uclibc	faible
EX_ 52 : Pouvoir utiliser des bibliothèques partagées sur des systèmes sans mmu avec la glibc ou la uclibc Pouvoir utiliser des bibliothèques partagées sur des systèmes sans mmu avec la glibc ou la uclibc	fort
EX_ 53 : Pouvoir utiliser les timers POSIX en mode périodique et aperiodique sans perte de résolution.	fort
EX_ 54 : Pouvoir compiler du code « relogeable » (option pic) sur les plateformes de références et générer un binaire au format flat (nommu).	moyen
EX_ 55 : Pouvoir compiler les bibliothèques partagées pour XIP (Execute In Place) sur les plateformes de références avec la glibc ou la uclibc	moyen
EX_ 56 : Pouvoir utiliser la détection de problème de synchronisation au sein du noyau et analyser la prise de verrou. (deadlock ,lockdep)	moyen
EX_ 57 : Pouvoir tester (benchmarking tools [interbench, netperf, hackbench, dbench, stress]) les latences dans le système, la performance de l'ordonnanceur, mesurer la performance du réseau.	moyen
EX_ 58 : Pouvoir mesurer le temps de changement de contexte. (lmbench)	moyen
EX_ 59 : Pouvoir mesurer la performance des IPC.(lmbench)	moyen
EX_ 60 : Pouvoir tracer le temps passé avec les interruptions désactivées, la préemption désactivée et lors des changements de contexte. (CONFIG_SCHED_TRACER)	moyen

7 - Modélisation des cas d'utilisation

Lors de l'enquête, nous nous sommes retrouvés devant un nombre relativement important de cas d'utilisation du temps réel tous différents mais avec beaucoup de problématiques communes.. Afin de pouvoir les représenter au mieux, il a été décidé de décrire chacun des cas d'utilisation comme une composition de briques de base, chacune de ces briques représentant alors un problème élémentaire du temps réel.

Parmi ces diverses briques de base, nous trouverons :

- le traitement de flux de données par plusieurs taches
- des gestions de GPIO
- le problème de l'accès à des média
- l'accès au réseau.

Nous aurons aussi à utiliser dans tous ces uses cases des fonctionnalités qu'une équipe produit est en droit d'attendre :

- L'estimation des ressources CPU utilisées par une tache, le système, éventuellement par le système non temps réel,
- Des indicateurs de charge et de stress permettant de voir si l'ajout, le retrait d'une tache à un impact significatif.

C'est seulement une fois les différentes briques de base présentées que seront abordés les cas d'utilisation.

7.1 - Modèle de traitement de flux de données

Ce modèle illustre ce que fait une application qui a besoin d'enchaîner des traitements sur un flux de données continu.

On suppose que des raisons légitimes font que les traitements ne peuvent alors pas être exécutés par une seule tache de manière séquentielle. Plusieurs taches sont alors contraintes de collaborer ensemble.

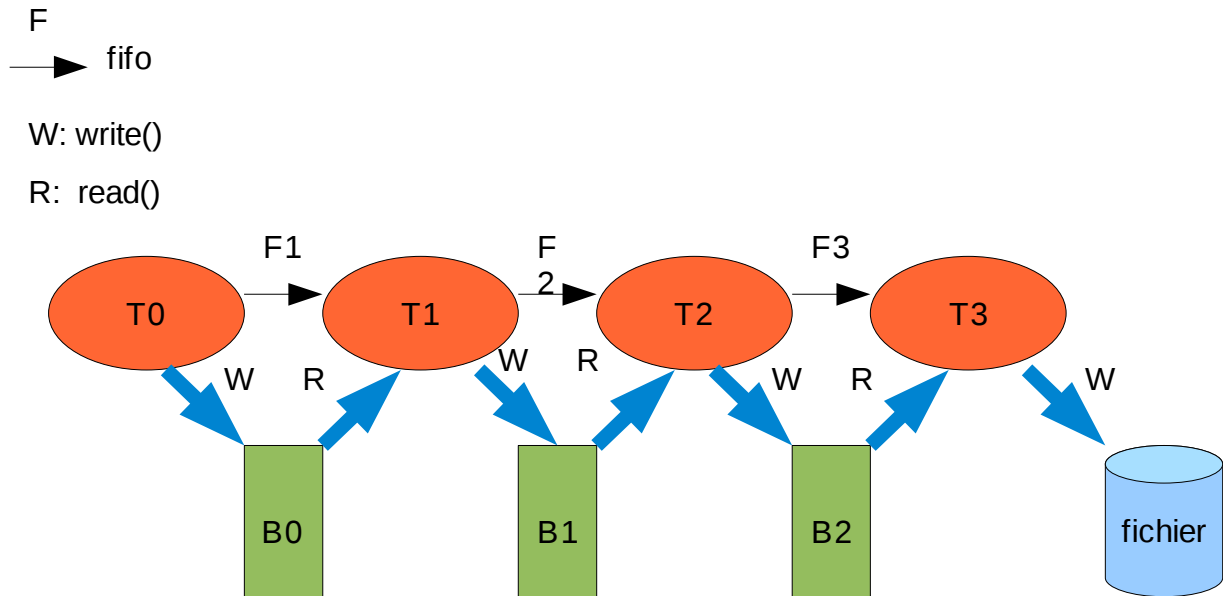
Dans notre exemple, les taches en bout de chaîne sont responsables de la production et de la consommation des données. Les taches intermédiaires reçoivent des données d'un coté, appliquent le traitement dont elles sont responsables et passent ces données à la tache suivante.

A cette occasion, la tache doit utiliser une certaine quantité de CPU, assurer que le flux d'un bout à l'autre se fasse de manière assurée et régulière.

7.1.1 - Détermination des dates de consommation

La programmation de la consommation consiste à programmer pour chaque paquet la date à laquelle il doit être consommé.

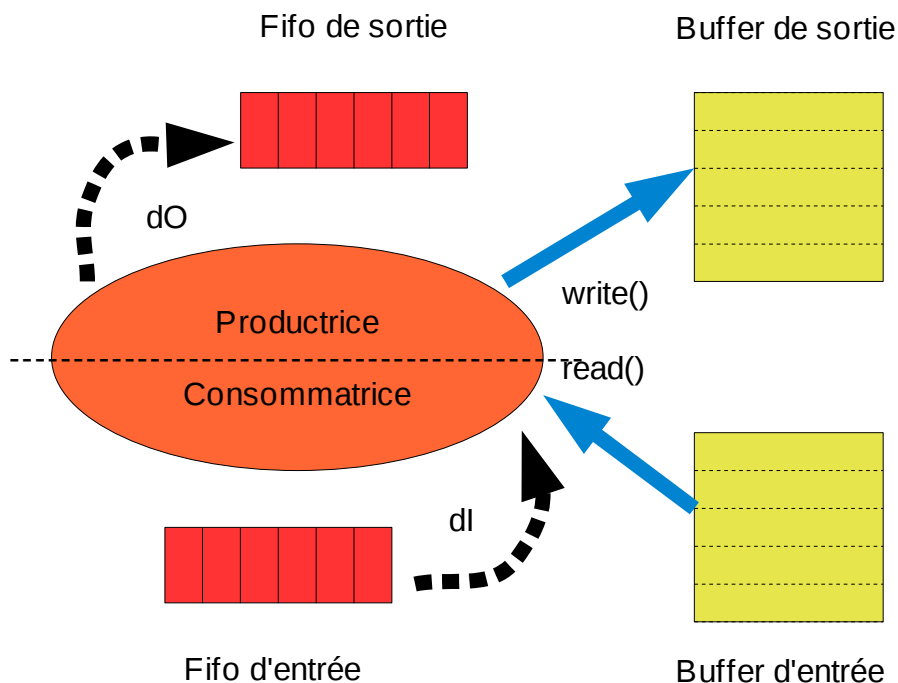
Dans ce modèle, les paquets sont datés lors de leur fabrication au sein de la première tache. La consommation des paquets par la dernière tache doit suivre strictement ces indications de dates



7.1.2 - limitation de la bufferisation

Les ressources mémoires d'un équipement embarqué sont généralement limitées. Il n'est pas rare de disposer seulement de 32 méga octets de mémoire vive. Les tailles des buffers et fifos du modèle seront donc adaptées aux contraintes mémoires des plateformes utilisées.

7.1.3 - Structure d'une tâche intermédiaire



Une tâche intermédiaire du traitement du flux est à la fois consommatrice et productrice de données.

Elle dispose d'une fifo à l'entrée, sur laquelle elle reçoit les descripteurs de paquet de la tâche précédente. Pour chaque descripteur de paquet reçu, elle alloue un buffer dans lequel elle fabrique un nouveau paquet et en poste la description dans une fifo à destination de la tâche suivante. Une fois consommé, le paquet reçu doit être libéré.

Lors de la fabrication du paquet, la tâche peut utiliser des données du paquet reçu.

Pour chaque paquet, la tâche doit consommer un certain quota de CPU pour illustrer un cas réel.

7.1.4 - Communication entre les tâches

Dans le modèle considéré chaque tâche productrice de données est reliée à une tâche consommatrice par une fifo. Cette fifo est vue comme la fifo de sortie de la tâche productrice et comme la fifo d'entrée de la tâche consommatrice.

7.1.4.1 - Fifo d'entrée

A l'initialisation du traitement de flux une tâche consommatrice de données est endormie dans l'attente de données à traiter.

Le réveil de la tâche peut se faire de diverses manières :

- Sur signal provoqué par l'arrivée d'un paquet sur la fifo d'entrée.
- Sur timer afin de faire une scrutation périodique des paquets en entrée.

Le mode de réveil de la tâche doit être paramétrable.

Si la fifo est vide au réveil de la tâche consommatrice alors elle produit une erreur underrun par incrément du compteur fifoEmpty propre à la fifo puis s'endort à nouveau.

7.1.4.2 - Fifo de sortie

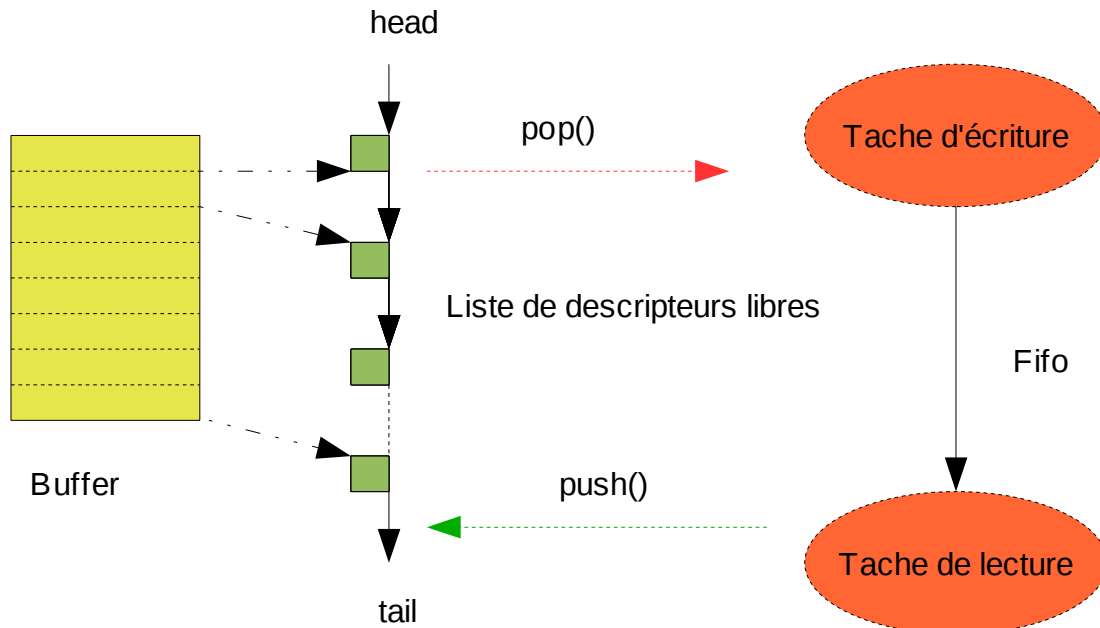
Lorsque la fifo de sortie est pleine le paquet courant est perdu et une erreur overrun est produite par incrément du compteur fifoFull propre à la fifo.

7.1.4.3 - Implémentation des fifos

Pour les tâches portées par des threads au sein d'un même processus linux, les fifos sont des tampons de mémoire circulaires avec des pointeurs de lecture et écriture. L'accès à la fifo est exclusif afin de garantir la synchronisation entre les tâches. La notification de dépôt de données dans la fifo peut être faite au moyen de variables conditions.

Dans le cas d'utilisation d'une skin xenomai vxworks ce sont des queues de messages.

Autrement pour les tâches portées par des processus distincts ce sont des tubes nommés (mkfifo).



Le flux est organisé en paquets de données rassemblés dans un buffer. Chaque paquet est référencé par un descripteur qui renseigne sa taille et son emplacement (offset) dans le buffer. Les taches écrivent et lisent les données dans les paquets à l'aide de ces descripteurs. Une liste chaînée rassemble les descripteurs des paquets libres d'utilisation. Initialement tous les paquets du buffer sont libres. Le cycle d'utilisation d'un paquet est le suivant: la tache d'écriture d'un paquet obtient le descripteur en tête de la liste des descripteurs libres, produit les données au sein du paquet et transmet le descripteur à la tache de lecture. Cette dernière lit toutes les données du paquet puis réinsère le descripteur en queue de la liste des descripteurs libres.

7.1.5.1 - Implémentation des buffers.

Les buffers devant contenir les paquets sont alloués et libérés par chacune des taches du traitement de flux dans des pools de buffers préalloués.

Ces pools de buffers sont des mémoires partagées entre les taches.

7.1.6 - Taches de production et consommation du traitement du flux

7.1.6.1 - Production

Cette tache initie le flux de données. Elle est périodiquement réveillée afin d'écrire N paquets de taille identique dans son buffer de sortie B0. Le contenu d'un paquet est aléatoire. La tache date les paquets générés lors de leur insertion dans sa fifo de sortie. Dans le mode de réveil par signal des taches consommatrices, le dépôt de données est notifié.

Si la fifo de sortie est pleine, le paquet à insérer est perdu et l'erreur overrun renseignée.

7.1.6.2 - Consommation

Cette tâche exploite le flux résultant du traitement. Elle retire les paquets de sa fifo d'entrée, calcule et enregistre leur latence de traitement afin qu'un histogramme puisse être produit par la suite.

Dans le cas de scrutation, l'absence de données dans la fifo d'entrée est une erreur de type underrun.

Un paramètre supplémentaire permet d'activer l'archive des paquets dans un fichier sur une mémoire de masse. L'archive est une simple copie des paquets les uns à la suite des autres dans le fichier.

7.1.6.3 - Temps de traitement

Le temps de traitement d'un paquet est calculé en faisant la différence entre la date de consommation du paquet et la date de génération.

7.1.6.4 - Gigue

La gigue pour le traitement d'un paquet du flux de données est la différence entre les valeurs maximum et minimum des latences de traitement d'un paquet du flux enregistrées sur la période de mesure. Cette période de mesure est paramétrable.

A l'issue d'une mesure, les gigue et latence moyennes sont présentées.

7.1.6.5 - Histogramme

La collecte des valeurs pour les histogrammes est faite pendant la mesure. Ces valeurs concernent les latences et giges de traitement d'un paquet. Elles permettent d'avoir une idée sur leur répartition pendant l'enregistrement.

Les outils permettant la collecte des données pour l'histogramme devraient être faits sous forme d'une librairie.

7.1.7 - Paramétrages du traitement de flux

Un ensemble de variables sont paramétrées par l'utilisateur afin d'offrir quelques libertés dans la configuration du traitement de flux.

7.1.7.1 - Multi instances

Le modèle de traitement de flux peut être instancié une ou plusieurs fois. Dans le cas de use case multi instances, l'ensemble des ressources mises en œuvre sont dupliquées pour chaque instance. Par défaut une seule instance est considérée.

7.1.7.2 - Priorités

Le paramétrage doit pouvoir permettre d'attribuer des priorités différentes à chacune des tâches du traitement du flux.

7.1.7.3 - Mode de réveil

Le réveil des tâches consommatrices de données peut se faire selon deux modes: par timer ou à l'arrivée de données. Une option est disponible pour sélectionner chacune des alternatives. Le réveil par timer est le mode par défaut. Afin d'ajuster au mieux le niveau de remplissage des fifos la période de scrutation sera paramétrée à l'exécution par l'utilisateur.

7.1.7.4 - Tache de consommation du flux

Par défaut la tache consommatrice du flux archive les paquets dans un fichier sur la mémoire de masse. Une option est disponible pour annuler cette action.

7.1.7.5 - Tache de production du flux

Des options sont disponibles pour paramétrer la fréquence de réveil de la tache de production, la taille et le nombre de paquets générés.

Suivant les protocoles, la taille des paquets varie généralement entre quelques dizaines et quelques centaines d'octets.

7.1.7.6 - Profondeur des Fifos

La profondeur de chacune des fifos est paramétrable.

7.1.7.7 - Histogramme

Une option est disponible pour paramétrer le nombre d'échantillons de latence recueillis pour produire l'histogramme.

Le nombre d'échantillons doit être suffisamment grand pour présenter une statistique significative.

7.2 - Concurrence de flux programmés

Ce modèle se base sur le modèle de traitement des flux.

Son originalité est de programmer pour chaque paquet la date à laquelle il doit être consommé. Ces dates sont calculées par la tache de production et associées au paquet. La tache de consommation stocke et consomme chaque paquet en fonction de la date qui lui a été affectée.

La tache de production peut, lorsqu'elle à la main, produire une salve de paquet à cette occasion.

Ce modèle est typiquement la représentation de la concurrence de traitement de flux vidéo avec des traitement de flux audio dans une setup box.

7.2.1 - Méthode de programmation des paquets

Dans notre modèle, sans précaution particulière, les paquets de plusieurs flux concurrents ont peu de chance d'être consommés à des dates communes. Afin de systématiser les coïncidences des dates de consommation du système de traitement de flux concurrents, ce qui constitue le cas le plus défavorable pour un système, il a été décidé de prendre pour référence le temps de la machine et d'aligner les dates de consommation de façon à pouvoir les écrire comme étant $t = n \times T$ ou T est une période de consommation et n un entier.

Avec cette règle, plusieurs traitements de flux concurrents A et B ont des chances maximum d'avoir des dates de consommations communes quelque soit leur périodes respectives.

Pour une tache de production, programmer une date de consommation consiste à estimer le temps de transit au travers de toutes les taches, prendre une marge de sécurité et aligner la date obtenue grâce à un modulo de façon à ce qu'elle soit de la forme $n \times T$

$$t_{\text{programmé}} = \text{PartieEntiere}((t_{\text{courant}} + t_{\text{transit}} + t_{\text{marge}} + T)/T) * T$$

où :

- T est la période de consommation.

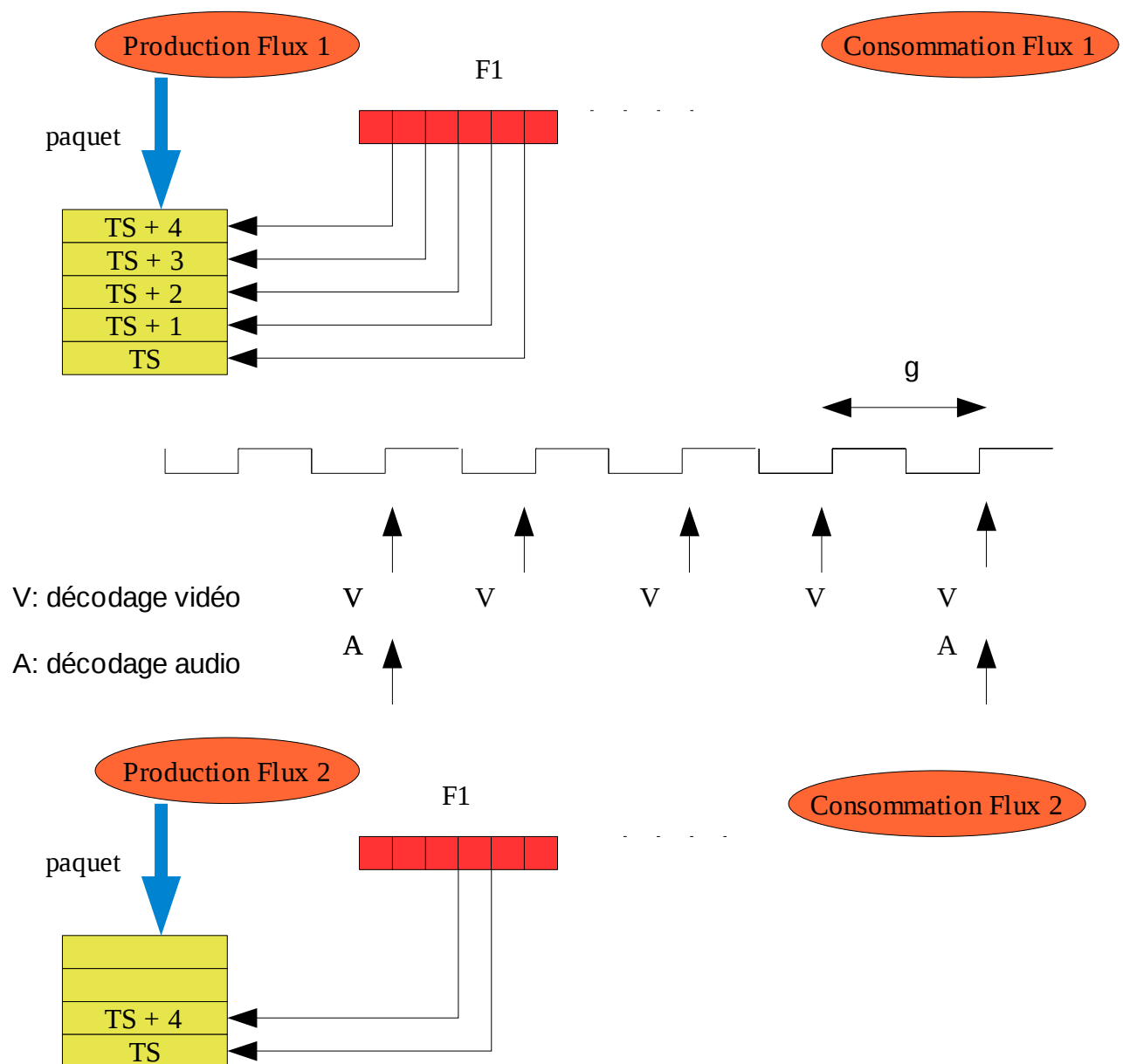
- $t_{\text{programmé}}$ est la date de consommation.
- t_{courant} est la date au moment du calcul par la tâche de production
- t_{transit} est le temps de transit estimé pour aller de la tâche de production à la tâche de consommation
- t_{marge} est un petit temps de sécurité de l'ordre de quelques périodes T pour prendre en compte les aléas du procédé.

7.2.2 - Concurrence de flux

Le modèle de concurrence des flux mets en œuvre au moins 2 flux concurrents utilisant chacun la méthode de programmation des paquets décrite ci dessus.

Les périodes de consommation de chacun de ces flux seront prises du même ordre de grandeur même si elles ne sont pas égales. L'idéal pour stresser le système étant de faire en sorte qu'elles soient égales ou multiples l'une de l'autre. Les tâches de production de chacun des traitements de flux seront configurées de façon à produire des salves de paquets à consommer au cours de la période par la tâche de consommation.

Le but de ce type de cas d'utilisation est bien de voir comment le système d'exploitation gère les échéances alors que les conditions initiales sont les plus défavorables.



On mesurera sur ce type de cas d'utilisation le comportement général en fonctionnement nominal et sous stress. Une attention particulière sera portée sur les giges observées à la consommation entre les dates programmées et les dates de consommation effectivement observées.

Sur les setupbox, la consommation peut se traduire par une simple consommation CPU qui traduit l'envoi du paquet pour produire l'image ou le son. Elle peut également se traduire par une écriture sur disque de l'information reçue pour pouvoir la visionner ultérieurement.

Les uses case doivent donc simuler cette consommation CPU ou écrire sur un file system pour s'approcher des conditions réelles.

7.3 - Stress du CPU et de la mémoire de masse

L'application de stress sert à occuper des ressources pour se placer explicitement en concurrence de taches temps réel.

Une des applications de stress envisagée est «Stress»

<http://weather.ou.edu/~apw/projects/stress/> un programme de référence pour la génération de charge sur un système POSIX. Il peut imposer de façon configurable une charge CPU, mémoire, d'entrée sortie et un stress de la mémoire de masse.

Cette tache est notamment utilisée pour occuper un temps CPU significatif et stresser la couche logicielle VFS du noyau linux. La consommation de temps CPU est réalisée par le calcul de racines carrées et le stress VFS par des écritures de taille configurable répétées sur la mémoire de masse. Le nombre d'instances (fork) pour chacun de ces éléments de charge est paramétrable. Il est ainsi possible d'accroître la charge appliquée au système de façon progressive.

Les mémoires de masse utilisées sont un disque dur et une mémoire flash.

La tache de stress doit pouvoir être exécutée :

- Dans l'espace non temps réel : Le premier cas valide le respect des contraintes des taches temps réel face à la tache de stress.
- Dans l'espace temps réel : Le second cas valide la gestion des priorités entre les taches temps réel concurrentes.

Dans les deux cas la tache de stress est interrompue pour servir les taches de plus forte priorité.

7.3.1 - Paramétrage de la tache de stress

7.3.1.1 - Instances d'occupation CPU

Par défaut une seule instance réalise des calcul de racines carrées afin de consommer du temps CPU. Une option est disponible pour augmenter le nombre d'instances.

7.3.1.2 - Nombre d'instances d'écriture et taille des blocs

Par défaut une seule instance réalise des écritures de blocs sur la mémoire de masse. Des options sont disponibles pour surcharger le nombre d'instances et la taille des écritures. La taille des écritures seront adaptées aux ressources de l'embarqué.

7.3.1.3 - Partitionnement de la mémoire de masse

Par défaut l'archive du flux traité et les blocs écrits par la tâche de stress sont produits sur une même partition de la mémoire de masse. Une option est disponible pour que la tâche de stress écrive sur une partition distincte.

7.4 - Estimation de charge CPU

Un outil logiciel estime la quantité de travail réalisé par le système sur la période considérée.

7.4.1 - Charge globale

Un indicateur de charge globale présente la proportion des ressources CPU disponibles sur le système qui sont utilisées au cours de la période de mesure. Ainsi un indicateur de charge globale de valeur x% signifie que sur les 100 échantillons relevés pendant la période considérée le processeur était occupé pour « x » d'entre eux.

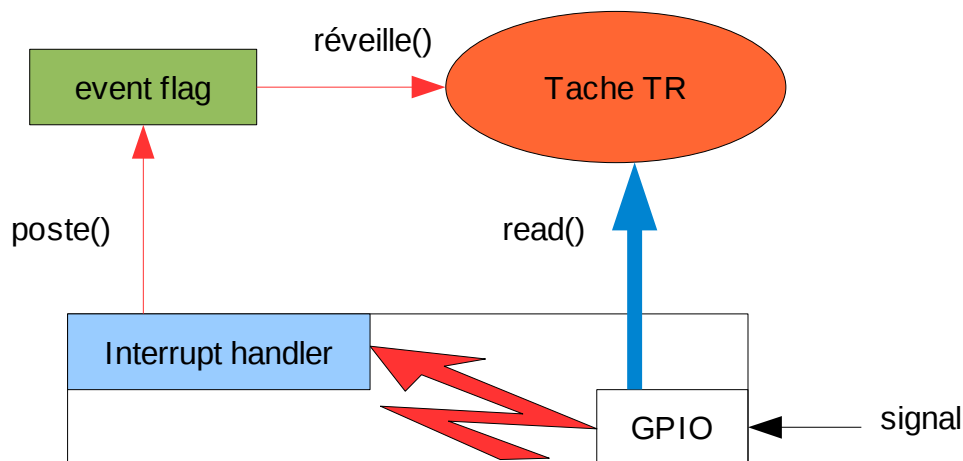
7.4.2 - Charge des tâches

L'outil présente sur la période de mesure pour chacune des tâches présentes dans le système

- son pourcentage d'utilisation du CPU
- la proportion des temps passés dans les états bloqué et en attente du CPU.

7.5 - Réception d'un signal sur un GPIO

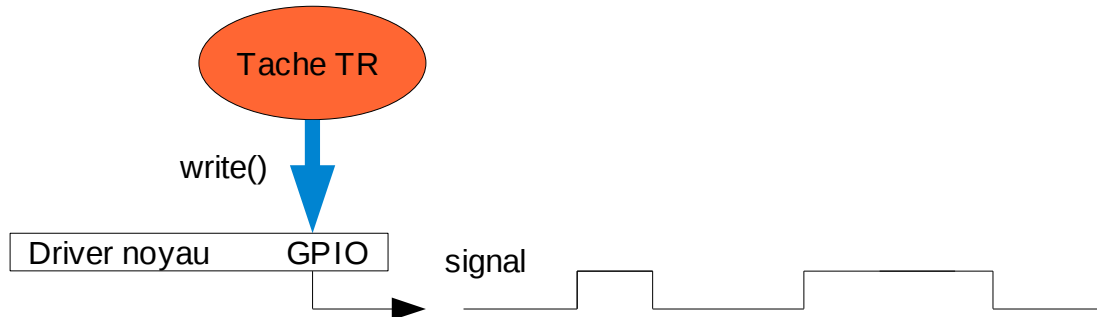
Ce modèle présente la réception d'un signal sur un GPIO afin de renseigner une position par exemple.



Un drapeau d'événements est utilisé pour notifier l'arrivée d'une nouvelle donnée sur le GPIO configuré en entrée. Le drapeau est écrit par le driver noyau afin de réveiller la tâche en attente de donnée à lire sur le GPIO. Le GPIO génère une interruption sur changement de niveau. Le handler du driver noyau acquitte l'interruption et poste l'événement dans le drapeau.

7.6 - Production d'un signal sur un GPIO

Ce modèle présente la production d'un signal sur un GPIO afin de piloter un équipement par exemple.



Une tache temps réel de l'espace utilisateur écrit sur le node d'un char device driver les valeurs 0 ou 1 en alternance et espacées dans le temps. Le driver produit deux valeurs distinctes sur un GPIO configuré en sortie pour former les niveaux 0 et 1 du signal mesurable avec un analyseur logique.

7.7 - Le temps réel et le réseau

7.7.1 - Flux de VOIP

Ce modèle illustre le cas d'utilisation de certains flux réseau de la VOIP soumis à de très fortes contraintes temporelles.

Typiquement, une stack de VOIP se décompose en 3 couches logicielles.

- Une couche permettant la déclaration de l'équipement au réseau
- Une couche permettant d'émettre ou de recevoir des appels vers ou depuis un distant
- Une couche supportant les canaux logiques : les voies par les quelles passent la voix.

Les deux premières couches n'ont pas besoin d'être extrêmement réactives, un retard de l'ordre de la seconde est tout à fait acceptable quand il s'agit de recevoir un appel téléphonique. La dernière couche par contre est chargée de faire transiter le signal d'un bout à l'autre de façon très régulière avec le moins de retard ou de perte possible, basée sur RTP/RTCP.

C'est donc à cette dernière couche que s'intéressent les uses cases de VOIP.

Pour des raisons de simplicité, notre modèle n'utilisera pas de vrais téléphones. Il se contentera de produire des flux fictifs ayant toutes les caractéristiques de flux réels, de les recevoir et de produire les statistiques.

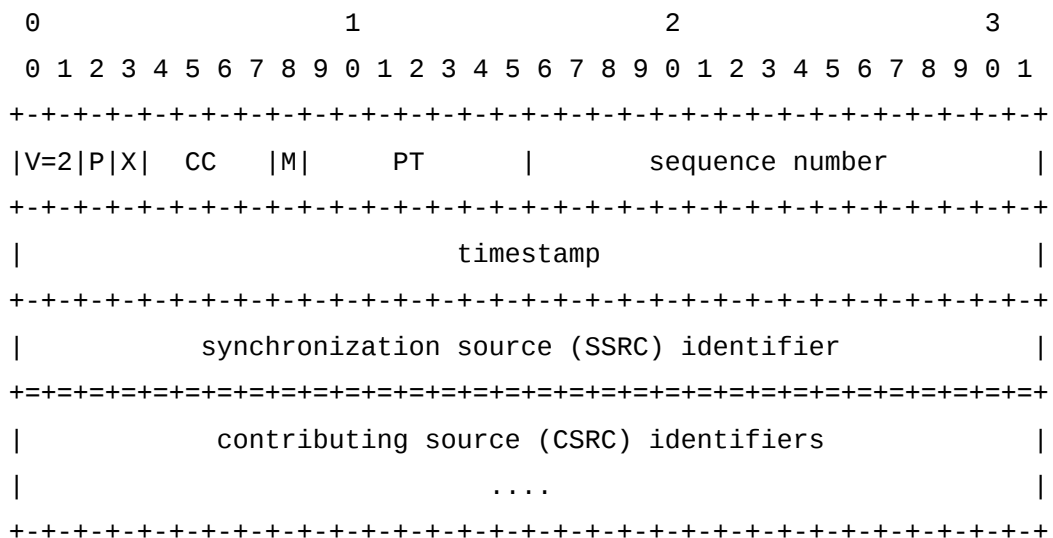
7.7.1.1 - Modélisation des flux RTP

Un simple appel de VOIP mets en œuvre 2 flux RTP unidirectionnel de sens opposés. Chacun de ces flux RTP est normalement accompagné de flux RTCP que nous ignorerons pour ce use case.

7.7.1.1.1 - Flux RTP-like

Les trames RTP sont des trames UDP/IP. Elles sont émises typiquement toutes les 10ms et ont une taille d'environ 64 octets.

L'entête RTP, issu de la RFC 3550, a la forme suivante:



La trame sera définie sans padding, extensions, CSRC, marker. La payloadtype n'a pas d'importance dans la mesure où nous ne faisons pas d'exercice d'interopérabilité.

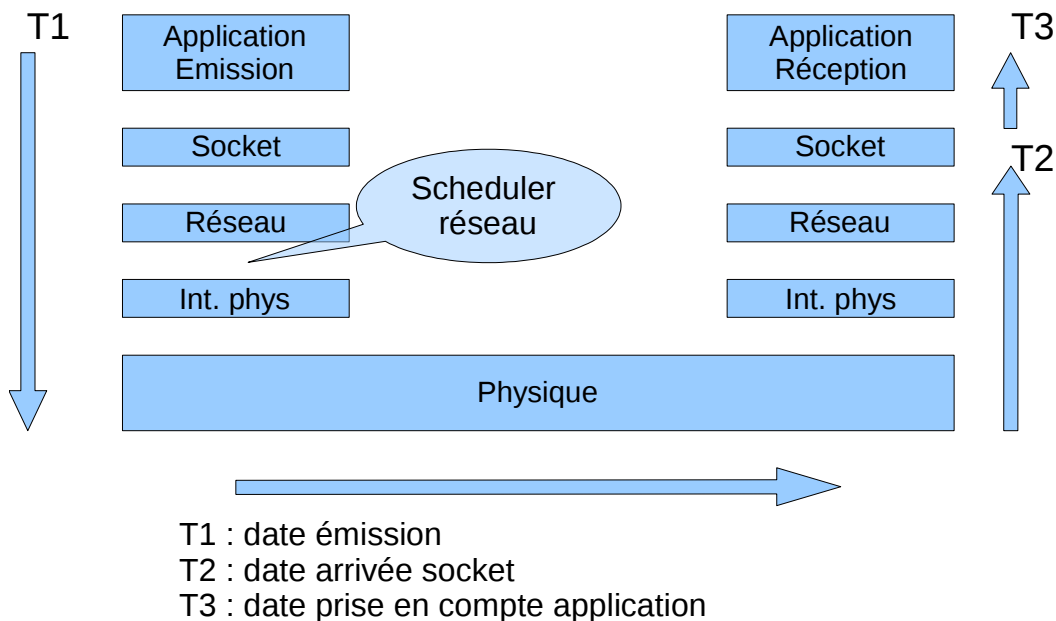
La valeur de SSRC sera également à 0.

Nous choisirons un timestamp en nanoseconde : il permettra de calculer les retards et la gigue au niveau des trames réseau.

7.7.1.2 - Mesures

Nativement, le RTP embarque des champs permettant de mesurer la perte et de dater la trame. Ceux ci sont mis à contribution dans nos cas d'utilisation pour mesurer la perte, les retards et la gigue. Les mesures seront faites de manière à identifier l'élément qui introduit du retard et de la gigue.

7.7.1.2.1 - Description du chemin d'une trame



Les Flux RTP sont basés sur UDP, un protocole non connecté, lui même basé sur IP.

Sur la machine émettrice, les paquets sont posés par l'application au niveau d'une socket qui les transmet à la pile UDP/IP où ils reçoivent les encapsulations adéquates avant d'être donnés au driver de l'interface physique pour être envoyés sur le réseau.

La machine réceptrice reçoit sur son interface la trame, qui est passée à la pile réseau qui la pose sur la socket. Sur notification de l'événement ou en allant scruter la socket l'application peut alors prendre en compte le paquet.

7.7.1.2.2 - Mesure de la perte

En l'absence de flux RTCP, la mesure de la perte est faite et stockée par le distant grâce au numéro de séquence. L'émetteur incrémente à l'émission ce numéro de séquence, le récepteur mesure les pertes de paquets et les comptabilise.

7.7.1.2.3 - Mesure des retards

La mesure des retards se fait au niveau de l'application de réception, en supposant que les deux distants sont synchronisés finement.

Plusieurs types de retards peuvent être mesurés :

- le retard global : mesuré entre l'application d'émission et l'application de réception. (T3-T1)
- Le retard dû au réseau : mesuré entre l'application d'émission et la socket de réception (date de réception sur la socket) (T2-T1)
- le retard dû au système d'exploitation: mesuré entre la socket de réception et l'application de réception.(T3-T2)

Les trois retards peuvent être mesurés à la réception de la trame en exploitant les timestamps du paquet et la date d'arrivée sur la socket de réception (SO_TIMESTAMPNS). Ils permettront de déterminer qui les crée.

Note : Dans le cas où la stack réseau utilisée est RTNet, il n'y a pas de possibilité de configurer SO_TIMESTAMPNS, on cherchera, si possible, une mesure équivalente.

Pour chacun des retards, il y a une mesure instantanée, une mesure moyenne, la variance et l'écart type.

7.7.1.2.4 - Mesure des giges

Nous considérerons dans notre cas d'utilisation que les flux de VOIP sont composés de trames de taille toujours égales pour un flux donné et émises de façon périodique.

Il est alors possible de mesurer les giges en se basant sur les mesures de retard instantanées. Comme pour les retards plusieurs types de giges pourront être mesurés.

- La gigue globale : mesurée à partir des informations de retards globaux.
- Le gigue dû au réseau
- la gigue dû au système d'exploitation.

Pour chacun des retards, une mesure moyenne, la variance et l'écart type. La mesure instantanée de la gigue n'a pas beaucoup de sens.

Seuls les retards instantanés de 2 trames consécutives pourront être utilisés pour faire des estimations de gigue afin de prendre en compte les effets d'une éventuelle perte de paquets.

7.7.1.3 - Perturbation de la VOIP

On ne s'intéressera pas du tout aux perturbations réseau introduites par une infrastructure réseau mais plutôt au stress imprimé par des flux concurrents au sein de la machine.

Les flux réseaux susceptibles d'entrer en concurrence avec les flux de VOIP peuvent être :

- des flux locaux associés à des applications temps réel.
- des flux locaux associés à des applications non temps réel.
- des flux routés ou bridgés au travers de la machine.

7.7.1.3.1 - Concurrence de flux d'applications temps réel

Ce sont des flux ayant typiquement des besoins de temps réel, devant partager la ressource commune que constitue une interface réseau. Les usages illustreront les cas d'applications aux priorités différentes. Les pertes, les retards et la gigue introduites par ces concurrences seront regardées de près.

7.7.1.3.2 - Concurrence de flux d'applications temps réel de même priorité

Ce cas illustre parfaitement le cas où plusieurs conversations téléphoniques fonctionnent en concurrence.

Si plusieurs flux de VOIP sont établis et qu'ils ont les mêmes caractéristiques de débit alors ils doivent être équilibrés. Les mesures de retard et de gigue dus au système doivent être identiques pour chacun d'eux.

7.7.1.3.3 - Concurrence de flux d'applications temps réel avec ceux d'applications non temps réel

Les applications temps réel exploitant les interfaces réseaux vont se retrouver en concurrence avec des applications non temps réel. Celles-ci sont susceptibles d'initier tout type de trafic occupant une part significative des ressources. Malgré cette concurrence, une tâche temps réel doit pouvoir émettre une trame sur l'interface quand elle est « *schédulee* ». De même, une application doit pouvoir être avertie et réveillée dans les meilleurs délais lorsqu'une trame est reçue par le driver de l'interface et déposée sur sa socket.

L'application Iperf <http://sourceforge.net/projects/iperf/> pourra être utilisée pour créer les diverses formes de stress réseau d'une application non temps réel.

7.7.1.3.4 - Concurrence de flux routés ou bridés au travers de la machine.

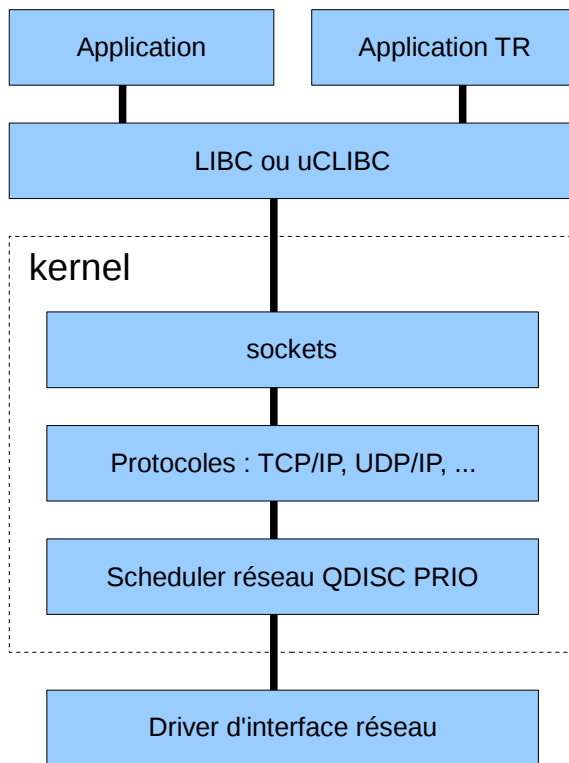
Ces cas d'utilisation visent à estimer l'influence des flux routés ou bridés traversant la machine sur les aspects temps réel de VOIP.

Une machine assurant un rôle de passerelle se voit traversée par des flux importants occupant ses ressources internes. Malgré cette concurrence, les applications temps réel doivent pouvoir émettre et recevoir leur trafic et ne doivent pas voir leurs caractéristiques temporelles se détériorer.

7.7.1.4 - Synchronisation des machines

La synchronisation des machines mises en œuvre se fait par le protocole Precision Time Protocol. Cela permet de synchroniser précisément les horloges des non temps réel afin d'affiner les mesures.

Une fois que les machines ont convergé suffisamment, on considère qu'elles partagent un temps commun suffisant pour être utilisé lors de l'estimation des retards. Le critère de convergence sera exprimé lors des divers cas d'utilisation.



7.7.2 - Accès au réseau via la stack réseau standard

Ce modèle vise à représenter une mise en œuvre du temps réel où les applications temps réel et les applications non temps réel utilisent une librairie de type LIBC ou uClibc pour accéder à la pile réseau standard de Linux. Il peut être mis en œuvre avec PreemptRT et Xenomai

Il n'est alors pas nécessaire de disposer d'un driver réseau spécialisé.

Trafic émis

Les divers flots réseaux utilisent les mêmes composants logiciels, il est nécessaire de distinguer ceux associés aux applications temps réel de ceux des autres applications afin de pouvoir appliquer un traitement différencié.

Ce traitement différencié permet de mettre en œuvre des chemins privilégiés pour permettre à des trames prioritaires de passer devant des trames non prioritaire.

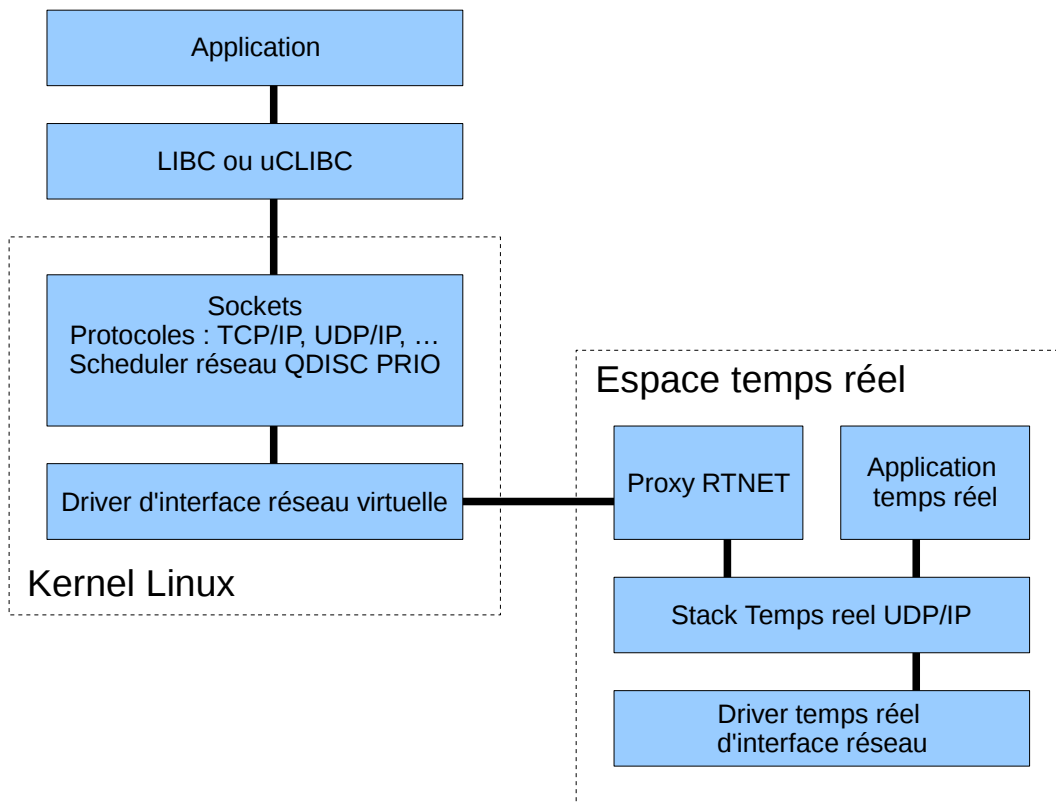
Les trames prioritaires sont marquées au niveau du champs priority de leur descripteur de trame skbuff. Ce marquage se fait par configuration de la socket (SO_PRIORITY). La priorité maximale sera donnée aux flux issus du temps réel.

Après avoir traversé la couche protocole, les trames sont posées dans l'une des files d'attente du scheduler QDISC en fonction de leur priorité. Le scheduler favorise les trames prioritaires lorsqu'il les envoie sur le driver d'interface.

Trafic reçu

Il n'y a pas de technique spécifique dans ce modèle pour privilégier les flux associés à des applications temps réel de celles qui ne le sont pas. Une fois arrivées au niveau driver d'interface, les trames traversent la stack réseau et sont déposées sur une socket. Le dépôt sur ces sockets peut provoquer une notification à l'application destinataire.

7.7.3 - Accès au réseau via RTNet



Ce modèle est représentatif pour Xenomai uniquement. Il vise à montrer une utilisation possible de RTNET.

Lors de l'enquête, aucune utilisation d'un réseau basée sur une mise en œuvre de réseaux synchrones RTMAC-TDMA n'a été vue.

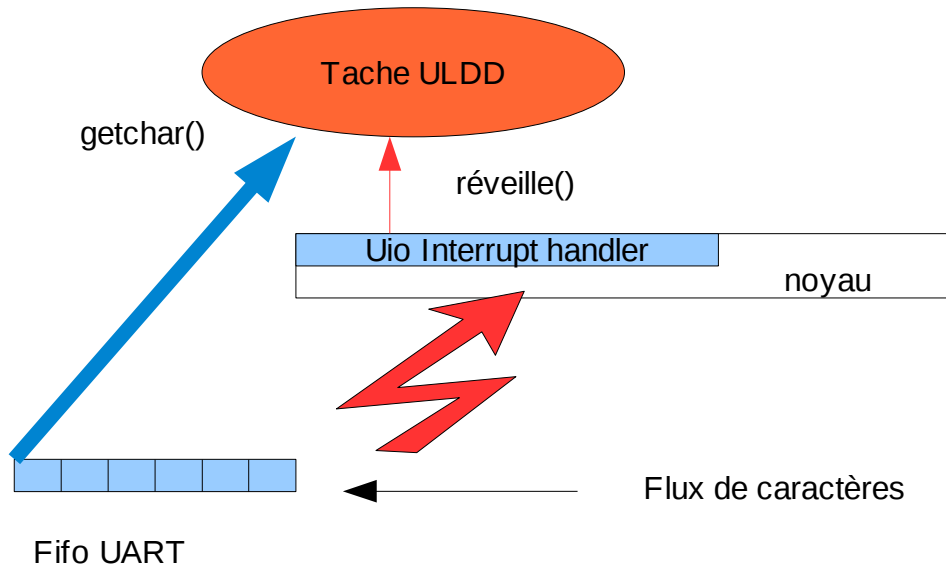
Il reste donc le partage d'un accès à une interface réseau. Dans ce cas d'utilisation, RTNET permet à une application temps réel d'accéder à l'interface réseau sans avoir à solliciter les services d'un scheduler de QOS. Il permet également à une stack linux d'accéder à cette interface réseau via le Proxy RTNet en transitant par une interface réseau virtuelle.

Ce modèle permet donc aux applications, temps réel ou non de partager un accès réseau. Il a quelques défauts connus :

- Les paquets des flux établis par la machine linux non temps réel sont systématiquement copiés ce qui a un impact sur les performances.
- La stack dans l'espace temps réel n'est pas protégée face à des agressions par un firewall
- La stack RTNET et la stack UDP/IP de la machine gère chacune de façon totalement indépendante l'attribution des ports UDP. L'intégrateur doit donc mettre en place une mécanique ou une politique pour ne jamais être en conflit vis avis de l'extérieur.

7.8 - Driver dans l'espace utilisateur ULDD

Ce modèle présente l'utilisation d'un driver dans l'espace utilisateur. Cette technique permet de s'affranchir des contraintes liées aux production et utilisation de code noyau mais dépend fortement des performances du système d'exploitation. Il est ainsi souhaitable depuis l'espace



utilisateur d'accéder dans les meilleures conditions aux registres et mémoires d'une carte d'entrée/sortie, de traiter les interruptions, d'avoir des garanties de latence, de pouvoir masquer les interruptions.

Le modèle considère le traitement de l'interruption d'un composant matériel d'entrée/sortie. La partie spécifique du driver est implémentée dans une tache de l'espace utilisateur. Un code noyau générique (uio driver) exporte les événements vers le driver de l'espace utilisateur.

Par souci de simplicité d'implémentation le matériel présenté est la fifo d'une UART.

La fifo génère une interruption à la réception de caractères. L'interruption est prise en compte par l'uio driver et relayée au handler de l'espace utilisateur de façon synchrone ou asynchrone selon le paramétrage. Le driver utilisateur extrait enfin les caractères de la fifo de réception de l'UART.

8 - Description des uses cases

8.1 - Audit et statistiques

Ces cas d'utilisation décrivent comment les équipes produit vont collecter les informations de configuration et de statistiques.

8.1.1 - Mesure de charge et temps CPU

but

Mesurer le temps d'utilisation CPU des taches au sein du système et produire un indicateur de charge.

description

Pour une période de mesure choisie, un outil logiciel présente les indicateurs de charge globale et le temps CPU utilisé par chacune des taches temps réel et non temps réel en exécution au sein du système.

scénario

Alors que plusieurs taches temps réel s'exécutent, l'utilisateur collecte les mesures de charge et de CPU.

résultats attendus

Obtention des données de charge et du temps CPU attribué aux taches temps réel et non temps réel.

8.1.2 - Mesures de retards

but

Mesurer le retard à l'exécution d'une action programmée pour se dérouler à une date précise.

description

Un ensemble de données sont datées pour être exploitées à un moment précis. Le retard se mesure par la différence entre la date effective de leur exploitation et leur date programmée.

scénario

L'utilisateur exécute une tache dans l'espace temps réel qui consomme des paquets de données pour lesquelles une date de consommation est prescrite. Pour chaque date de consommation la tache calcule et présente le retard à l'utilisateur.

résultats attendus

Les retards à la consommation des données sont présentées à l'utilisateur.

8.1.3 - Mesures de giges

but

Calculer et présenter la gigue obtenue sur un ensemble de mesures de latence.

description

Une application dans l'espace temps réel transmet régulièrement des paquets d'une tâche émettrice vers une tâche réceptrice. Pour chaque paquet la latence de transmission est enregistrée. Pour l'ensemble des latences obtenues sur la période de mesure choisie, la gigue mesure la différence entre les valeurs minimale et maximale.

scénario

L'utilisateur instrumente le code de l'application temps réel pour marquer les temps d'émission et de réception des paquets. L'utilisateur exécute l'application qui présente la gigue à chaque issue de la période de mesure choisie.

résultats attendus

Pour la période considérée, la gigue des latences mesurées est présentée à l'utilisateur.

8.1.4 - Production de l'histogramme des latences**but**

Produire l'histogramme d'un ensemble de mesures de latences.

description

Une application dans l'espace temps réel transmet régulièrement des paquets d'une tâche émettrice vers une tâche réceptrice. Pour chaque paquet la latence de transmission est enregistrée. Pour l'ensemble des latences obtenues sur la période de mesure choisie, l'histogramme présente une répartition des occurrences des latences mesurées.

scénario

L'utilisateur instrumente le code de l'application temps réel pour marquer les temps d'émission et de réception des paquets. L'utilisateur exécute l'application qui présente l'histogramme à chaque issue de la période de mesure choisie

résultats attendus

Pour la période de mesure considérée, l'histogramme est présenté à l'utilisateur. Les données sont organisées de façon à être affichées, avec un minimum de traitements, dans un outil graphique comme gnuplot.

8.1.5 - Chronogramme**but**

Tracer l'évolution temporelle de l'état d'une tâche: en exécution, en attente du CPU, bloquée.

description

Les dates de changements d'états des tâches temps réel sont enregistrés.

scénario

L'utilisateur active l'enregistrement des changements d'états à la compilation du système. Il exécute une application avec plusieurs tâches dans l'espace temps réel.

résultats attendus

Les dates de commutation entre les états sont disponibles pour chaque tache temps réel du système. Un outil externe au système peut recueillir ces données et les présenter graphiquement.

8.2 - Traitement de flux

Dans l'objectif d'observer le respect des contraintes des taches temps réel prioritaires au sein du système, les use cases suivants mettent en œuvre les modèles de traitement de flux et de stress CPU.

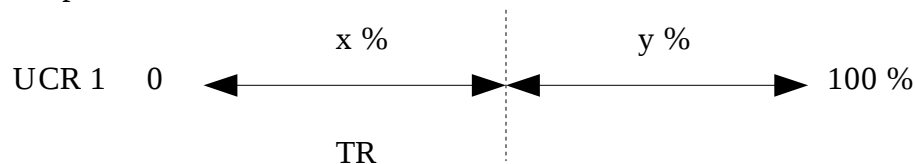
D'une façon générale il est attendu que le système temps réel préempte les taches de moindre priorité pour servir les taches prioritaires du traitement de flux afin de contenir la dérive des latence et gigue de traitement d'un paquet.

8.2.1 - Occupation complète du CPU (UCR 1)

x, y: attribution de temps CPU

TR: espace temps réel

NTR: espace non temps réel



but

Déterminer les paramètres du traitement de flux et de la tache de stress afin que l'indicateur de charge globale soit égal à 100% et que les paquets soient traités sans erreur.

description

Le traitement de flux est exécuté dans l'espace temps réel en concurrence de la tache de stress dans l'espace non temps réel. La tache consommatrice du flux est réduite à sa forme la plus simple: elle est réalisée sans archive sur la mémoire de masse. Les éléments de communication (fifo, buffer) sont ajustés afin d'avoir une exécution du traitement de flux sans erreur (aucun paquet perdu).

Ensuite les paramètres de la tache de stress sont ajustés afin de converger vers une mesure de la charge globale constante égale à 100%.

scénario

Dans un premier temps l'utilisateur exécute le traitement de flux sans la tache de stress CPU. Il procède à plusieurs essais en ajustant les paramètres de taille de fifo, de buffer et de nombre de paquets générés jusqu'à obtenir un traitement de flux sans perte de paquet.

Dans un second temps, l'utilisateur mesure à plusieurs reprises la charge produite par le traitement de flux et la tache de stress exécutés en concurrence. D'une exécution à l'autre

l'utilisateur ajuste les paramètres de la tâche de stress CPU pour converger vers un indicateur de charge constant égale à 100%.

résultats attendus

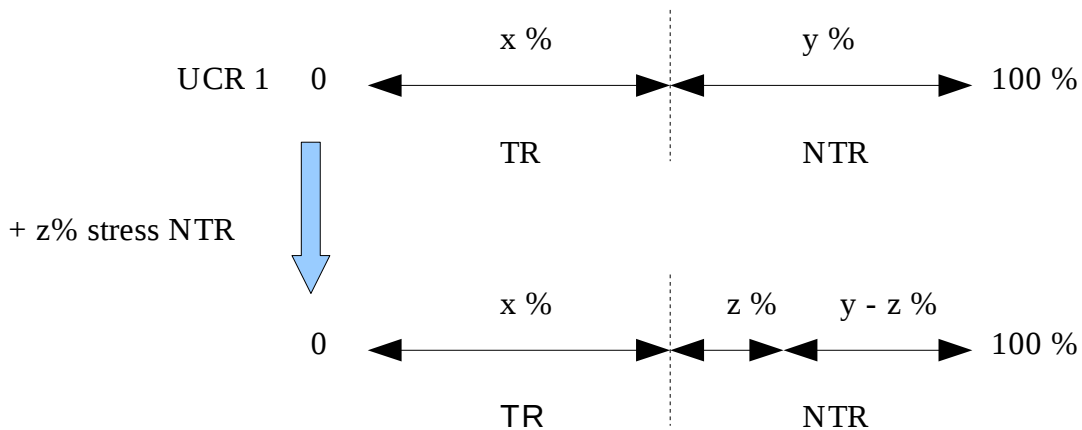
Le flux de données est traité sans erreur et la mesure de charge est constante égale à 100%. La totalité des ressources CPU sont utilisées et réparties entre x% pour les tâches de l'espace temps réel et y% pour les tâches non temps réel. Le paramétrage du traitement de flux et de la tâche de stress obtenu est référencé UCR1 (Use Case Référence 1) par la suite.

8.2.2 - Préservation du temps CPU attribué aux tâches temps réel lors de stress non temps réel

x, y: attribution de temps CPU

TR: espace temps réel

NTR: espace non temps réel



but

Montrer que quelque soit la charge dans l'espace non temps réel, les caractéristiques des tâches temps réel, le temps CPU qui leur est attribué sont conservés.

description

Un ensemble de tâches non temps réel sont ajoutées aux tâches instanciées par la configuration UCR1. Le nombre et l'activité de ces tâches supplémentaires sont augmentés à plusieurs reprises. Soit z% l'occupation CPU induite par ces tâches supplémentaires de l'espace non temps réel.

scénario

L'application de stress de la configuration UCR1 est paramétrée afin de générer la charge non temps réel supplémentaire. L'utilisateur augmente à plusieurs reprises le nombre d'instances de calcul des racines carrées et d'écriture sur la mémoire de masse jusqu'à paralyser les tâches de l'espace non temps réel.

scénario alternatif

Afin de stresser d'avantage le système, l'utilisateur ajoute la variation de la taille des écritures sur la mémoire de masse réalisées par l'application de stress.

résultats attendus

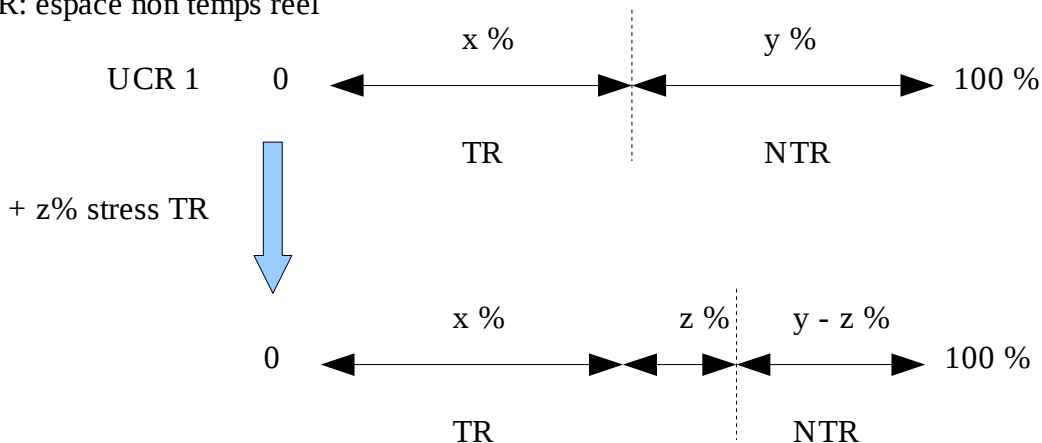
Quelque soit l'augmentation de la charge CPU des taches non temps réel, le temps CPU attribué aux taches temps réel est constant $x\%$. Le temps CPU attribué aux taches non temps réel en UCR1 est diminué du temps nécessaire pour servir les taches supplémentaires ($y-z\%$). Au plus fort de la charge CPU dans l'espace non temps réel, la gigue mesurée pour le traitement d'un paquet est identique à celle mesurée en UCR1.

8.2.3 - Stress temps réel et sauvegarde des caractéristiques des taches temps réel prioritaires

x, y : attribution de temps CPU

TR: espace temps réel

NTR: espace non temps réel



but

L'insertion de nouvelles taches temps réel au sein du système ne diminue pas le temps CPU attribué aux taches temps réel plus prioritaires.

description

Un ensemble de taches de stress dans l'espace temps réel sont ajoutées aux taches instanciées par la configuration UCR1. Les taches temps réel de UCR1 détiennent les plus fortes priorités au sein du système. Dans la limite de la puissance du processeur cible, le temps CPU $z\%$ attribué à ces taches supplémentaires est retiré aux taches de l'espace non temps réel.

scénario

L'utilisateur exécute la configuration UCR 1. En concurrence il exécute dans l'espace temps réel une nouvelle instance de l'application de stress dont les taches sont moins prioritaires que celle de UCR1. Il fait varier la charge du stress dans l'espace temps réel. Il mesure à chaque fois le temps d'occupation CPU des taches temps réel de UCR 1, des taches temps réel de l'application de stress et des taches de l'espace non temps réel.

résultats attendus

Quelque soit l'augmentation de la charge CPU des taches de stress temps réel, le temps CPU attribué aux taches temps réel de UCR1 est constant x %.

8.2.4 - Répartition équitable des ressources CPU entre les taches de même priorité**but**

Vérifier que le système ne défavorise pas une application.

description

Dans la limite de la puissance du processeur cible, toute nouvelle instance du traitement de flux ajoutée à la configuration UCR1 et de paramétrage identique occupe le même temps CPU.

scénario

L'utilisateur exécute la configuration UCR1 pour laquelle il mesure l'utilisation CPU x % des taches temps réel du traitement de flux. A plusieurs reprises, il ajoute l'exécution concurrente d'une nouvelle instance du traitement de flux dans l'espace temps réel et mesure le temps CPU occupé par les taches du système.

résultats attendus

Pour la mesure i , chacune des instances du traitement de flux occupe x % de temps CPU. A la mesure $i+1$, le temps CPU occupé par les i premières instances est conservé et celui de l'instance ajoutée en dernier est identique, égale à x %. Il est ainsi possible de déterminer le nombre maximum d'instances exécutables en concurrence dans le système sans dégrader leur performance.

8.2.5 - Concurrence d'un traitement de flux avec une tache très prioritaire**but**

Réaliser régulièrement une action prioritaire et courte en plus du décodage d'un flux.

description

Un flux est décodé dans l'espace temps réel suivant le modèle de consommation à dates précises. Toutes les 5 secondes une tache de plus haute priorité produit une séquence d'écritures sur un GPIO durant 5 microsecondes sans être interrompue. Une application de stress CPU est exécutés en concurrence. Pour chaque paquet le retard au décodage est calculé et présenté à l'utilisateur.

scénario

L'utilisateur exécute le traitement de flux et la tache d'écriture GPIO dans l'espace temps réel en concurrence de l'application de stress dans l'espace non temps réel. Il procède à plusieurs exécutions pour lesquelles il fait varier la charge produite par l'application de stress. Pour chaque variation il enregistre les mesures de retard au décodage.

scénario alternatif

L'utilisateur reprend le scénario précédent mais exécute l'application de stress dans l'espace temps réel avec une priorité plus faible que celle des traitement de flux.

résultats attendus

Quelque soit la charge de stress appliquée, la séquence d'écritures GPIO n'est pas interrompue et le flux est décodé sans retard.

8.2.6 - Priorités des tâches et garantie des performances**but**

Les contraintes temps réel des tâches prioritaires sont respectées malgré l'augmentation de charge produite par des tâches de moindre priorité.

description

Deux nouvelles instances de traitement flux moins prioritaires sont ajoutées à celle de la configuration UCR 1. Soient P1 la priorité du traitement de flux de UCR1, P2 et P3 les priorités des traitements de flux supplémentaires avec $P1 > P2 > P3$. Un stress avec la priorité P3 est ajouté dans l'espace temps réel.

scénario

L'utilisateur exécute la configuration UCR 1, la priorité du traitement de flux est P1. Il exécute en concurrence d'UCR1 deux nouvelles instances de traitement de flux de priorité P2 et P3 puis mesure les temps CPU occupés par les tâches du système. Dans un second temps il reprend l'expérience en ajoutant l'exécution concurrente de l'application de stress avec la priorité P3 dans l'espace temps réel.

résultats attendus

Avec ou sans stress temps réel ajouté, les temps CPU occupés par les traitements de flux de priorité P1 et P2 sont constants. Les contraintes temps réel des tâches de haute priorité sont préservées.

8.2.7 - Tâches temps réel et priorité d'accès à la mémoire de masse**but**

Les tâches temps réel accèdent prioritairement aux ressources partagées notamment à la mémoire de masse.

description

Les tâches temps réel et non temps réel accèdent concurremment à une ressource partagée: la mémoire de masse. Dans le modèle de traitement de flux présenté précédemment, la tâche consommatrice du flux, dans l'espace temps réel, archive les paquets de données dans un fichier sur la mémoire de masse (migration de domaine xenomai). En concurrence dans l'espace non temps réel, la tâche de stress écrit et efface régulièrement des fichiers sur cette mémoire de masse.

scénario

L'utilisateur exécute en concurrence une instance du traitement de flux avec archive et l'application de stress mémoire comme décrit ci dessus. L'utilisateur fait varier la charge d'écriture de la tâche de stress en augmentant le nombre d'instances (fork) et la taille des fichiers écrits. Les temps passés dans l'état bloqué et en attente du CPU sont enregistrés.

résultats attendus

Les accès à la mémoire de masse de la tâche temps réel sont prioritaires quelque soit la charge d'écriture produite par la tâche de stress non temps réel. Les temps passés dans l'état bloqué et en attente du CPU pour la tâche consommatrice du flux sont relativement constants à chaque exécution.

8.2.8 - Ressources partagées entre tâches temps réel**but**

Les tâches temps réel de plus forte priorité accèdent prioritairement aux ressources partagées.

description

Plusieurs instances du traitement de flux de priorités différentes archivent leur flux sur la mémoire de masse. Le flux prioritaire est archivé sans attendre.

scénario

L'utilisateur exécute 3 instances du traitement de flux de priorités différentes en concurrence de l'application de stress. Il mesure les temps passés dans l'état bloqué des tâches consommatrices de flux.

résultats attendus

La tâche consommatrice du flux prioritaire détient le temps passé dans l'état bloqué le plus court.

8.2.9 - Prémption d'une écriture sur un file system**but**

Prémption d'une écriture sur un file system pour servir une tâche prioritaire.

Description

La tâche de stress écrit (fonction write()) un gros bloc de données sur la mémoire de masse afin d'occuper un temps CPU significatif.

Dans un système linux non temps réel, ce temps serait suffisamment grand pour gêner les autres tâches en exécution au sein du système.

scénario

Le processus du traitement de flux est exécuté dans l'espace temps réel en concurrence de l'application de stress dans l'espace non temps réel.

scénario alternatif 1

Les processus du traitement de flux et de l'application de stress de moindre priorité sont exécutés en concurrence dans l'espace temps réel.

résultats attendus

Le système temps réel préempte la tâche de stress dans ses écriture de la mémoire de masse et sert les tâches prioritaires du traitement de flux bloquées ou en attente du CPU. Les écritures sont reprises ultérieurement afin que l'application de stress écrive néanmoins ses données sur la mémoire de masse.

8.2.10 - IPC**but**

Communication inter processus

description

Le traitement du flux est réparti sur plusieurs processus qui échangent les données avec les IPC.

Les tâches du traitement de flux sont portées par des processus distincts. Les buffers de paquets sont des mémoires partagées entre les processus. Les fifos entre les tâches sont des pipes nommés.

scénario

L'utilisateur crée les fifos (mkfifo) et les mémoires partagées puis exécute tous les processus du traitement de flux dans l'espace temps réel en concurrence de l'application de stress dans l'espace non temps réel. Il procède à plusieurs essais en faisant varier la charge de la tâche de stress.

résultats attendus

Quelque soit la charge de stress appliquée dans l'espace non temps réel, les contraintes des processus de l'espace temps réel sont préservées. Le flux est traité sans perte de paquet, la latence de traitement d'un paquet et le débit sont constants.

8.2.11 - Communication entre les tâches de l'espace utilisateur et noyau**but**

Les tâches temps réel de l'espace utilisateur et de l'espace noyau échangent des données et se synchronisent avec les services habituels: pipes, queues de messages, mutex, sémaphores, event flag, variables condition, etc ... Ces ressources sont partagées entre le noyau et l'espace utilisateur.

description

Le traitement du flux est distribué sur des tâches temps réel de l'espace utilisateur et de l'espace noyau. Les tâches intermédiaires du traitement de flux (T[1-2]) s'exécutent dans l'espace noyau au sein d'un pseudo device driver et les tâches en bout du traitement de flux (T0 et T3) s'exécutent dans l'espace utilisateur temps réel.

scénario

Dans l'espace temps réel, l'utilisateur charge le pseudo device driver au sein du noyau et exécute les processus des tâches de consommation et production de flux. En concurrence, dans l'espace non temps réel, il exécute l'application de stress.

scénario alternatif

L'application de stress et le traitement de flux sont exécutés dans l'espace temps réel.
L'application de stress détient la priorité la plus faible.

résultats attendus

Les services de communication et synchronisation inter tâches usuels en programmation temps réel sont disponibles pour relier les tâches de l'espace utilisateur à celle de l'espace noyau. La performance des services est suffisante pour respecter les contraintes temps réel. Le flux est traité sans perte de paquet, la latence de traitement d'un paquet est constante quelque soit la charge de stress appliquée.

8.2.12 - Consommation à date précise**but**

Décoder les paquets d'un flux de données aux dates prescrites.

description

Un traitement de flux programme la date de consommation des paquets selon le modèle présenté précédemment. Les paquets du flux sont donc décodés à date précises. Pour chaque paquet le retard au décodage est calculé et présenté à l'utilisateur. Le traitement de flux exécuté dans l'espace temps réel est perturbé par une application de stress CPU. Plusieurs exécutions sont produites afin de faire varier la période T pour le décodage des paquets entre 100 microsecondes et 10 ms.

scénario

L'utilisateur exécute le traitement de flux et l'application de stress dans l'espace non temps réel. Il procède à plusieurs exécutions pour lesquelles il fait varier la période pour le décodage et la charge produite par l'application de stress. Pour chaque variation il enregistre les mesures de retard au décodage.

scénario alternatif

L'application de stress est exécutée dans l'espace temps réel avec une priorité plus faible que le traitement de flux.

résultats attendus

Quelque soit la charge de stress, les paquets sont décodés sans retard.

8.2.13 - Synchronisation de flux**but**

Synchroniser le décodage des paquets de deux flux concurrents.

description

Deux flux sont décodés dans l'espace temps réel selon le modèle de flux synchronisés présentés précédemment. Pour les deux flux le retard au décodage est calculé et présenté à l'utilisateur. Une application de stress CPU est exécuté en concurrence des traitements de flux. Plusieurs exécutions sont produites afin de faire varier les périodes T_v et T_a pour le décodage des paquets entre 100 microsecondes et 10 ms.

scénario

L'utilisateur exécute les traitements de flux synchronisés en concurrence de l'application de stress dans l'espace non temps réel. L'utilisateur procède à plusieurs exécutions pour lesquelles il fait varier la période pour le décodage et la charge de l'application de stress. Pour chaque exécution il enregistre les retards au décodage, la désynchronisation.

scénario alternatif

L'utilisateur reprend le scénario précédent mais exécute l'application de stress dans l'espace temps réel avec une priorité plus faible que celle des traitements de flux.

résultats attendus

Quelque soit la charge de stress appliquée, les tâches consommatrices des flux décodent les paquets sans retard, la synchronisation des flux est conservée.

8.2.14 - Traitement de flux et écriture sur flash**but**

Vérifier que l'utilisation d'une flash comme mémoire de masse ne déforme pas les caractéristiques temps réel.

description

Les écritures concurrentes du traitement de flux et d'une application de stress CPU sur une flash n'altère pas les caractéristiques des tâches temps réel.

scénario

L'utilisateur exécute une instance du traitement de flux avec des consommations de données à dates précises dans l'espace temps réel en concurrence de l'application de stress dans l'espace non temps réel. L'utilisateur procède à plusieurs exécutions en faisant varier la charge produite par la charge de stress. Pour chaque exécution il mesure les retards à la consommation des paquets.

scénario alternatif

L'utilisateur exécute l'application de stress dans l'espace temps réel avec une priorité plus faible que le traitement de flux.

résultats attendus

Quelque soit la charge produite par l'application de stress, les retards à la consommation sont faibles et invariables.

8.3 - Driver dans l'espace utilisateur (ULDD)**but**

Exécuter un driver temps réel Xenomai dans l'espace utilisateur sans pénalité sur la réception des événements.

description

Une instance du modèle ULDD est mise en œuvre dans l'espace temps réel en concurrence de l'application de stress. La latence de réception de l'événement (arrivée de caractères) dans

l'espace utilisateur est mesurée par le temps écoulé entre l'entrée dans le handler d'interruption uio driver et la lecture d'un caractère dans la fifo.

scénario

Un équipement distant produit un flux de données sur l'UART. Le driver de l'espace utilisateur et temps réel est exécuté en concurrence de l'application de stress dans l'espace non temps réel. L'utilisateur exécute le scénario tour à tour pour une attente synchrone (read() bloquant) ou asynchrone (signal SIGIO) de l'interruption UART. Pour chacun des cas l'utilisateur fait varier la charge produite par l'application de stress. Pour chaque variation il mesure la latence de réception.

scénario alternatif 1

Le driver et l'application de stress sont exécutés en concurrence dans l'espace temps réel. Le driver ULDD détient la priorité la plus haute.

résultats attendus

Quelque soit la charge de l'application de stress, la latence de réception de l'événement est constante.

8.4 - Acquisition usb

but

Lire les paquets présentées sur un lien usb high speed avec une latence inférieur à 100 microsecondes.

description

Une tâche de l'espace temps réel utilise un driver comedi et la stack usb temps réel usb4rt pour extraire les paquets (urb) déposés sur le lien usb par un équipement distant. La latence entre le dépôt sur le lien usb et le retour de la fonction de lecture du urb est mesurée.

scénario

L'acquisition usb est exécutée dans l'espace temps réel en concurrence de l'application de stress dans l'espace non temps réel.

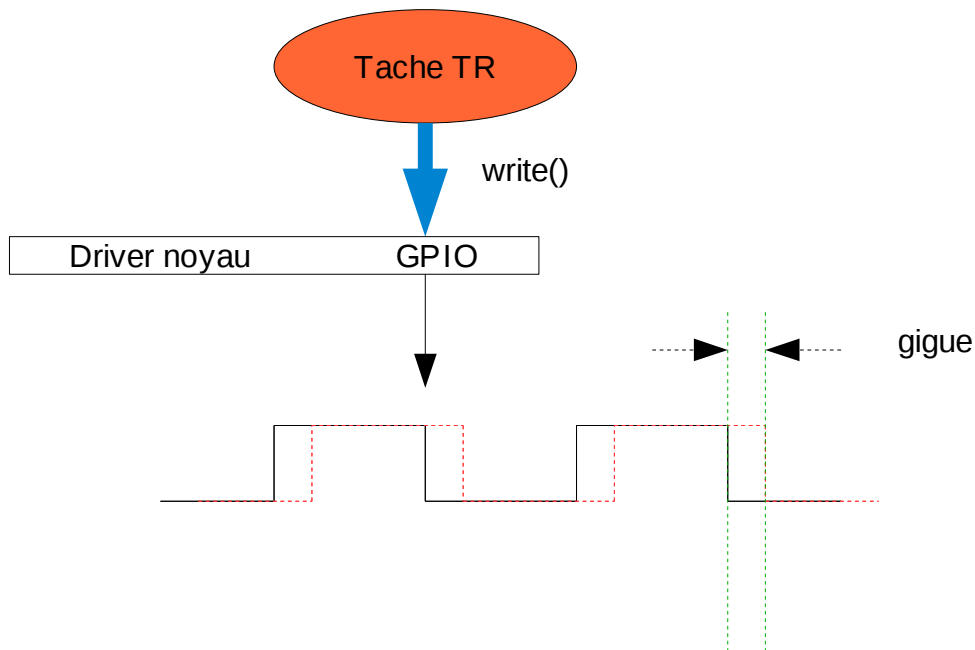
scénario alternatif

L'application de stress est exécuté dans l'espace temps réel avec une priorité plus faible que la tâche d'acquisition usb.

résultats attendus

Quelque soit la charge générée par l'application de stress, la latence mesurée est inférieure à 100 microsecondes.

8.5 - Signal sur un GPIO



but

Émettre un signal cadencé sur une fréquence précise.

description

Une tache temps réel produit un signal carré d'une demi période de 10 ms sur un GPIO selon le modèle présenté précédemment.

La tache réalise indéfiniment le cycle suivant: elle écrit la valeur 0x01, s'endort 10 ms, écrit la valeur 0 et s'endort à nouveau 10 ms.

Une tache de stress s'exécute en concurrence.

scénario

La tache temps réel génératrice du signal est instanciée dans l'espace utilisateur. Le signal est capturé sur un analyseur logique pendant l'exécution de l'application de stress dans l'espace non temps réel.

scénario alternatif 1

L'application de stress est exécutée dans l'espace temps réel avec une priorité inférieure à la tache productrice du signal.

scénario alternatif 2

La tache temps réel génératrice du signal est instanciée dans l'espace noyau au sein d'un driver.

résultats attendus

Quelque soit la charge appliquée par la tache de stress, l'analyseur logique présente un signal carré de gigue nulle.

8.6 - Accès à un GPIO non interrompu

but

Accéder à un GPIO pendant 5 microsecondes sans être interrompu.

description

Une tâche temps réel est réveillée par un timer toutes les 5 secondes et accède à un GPIO durant 5 us maximum. Cette tâche produit une séquence d'écriture sur le GPIO qui ne peut pas être retardée. Afin de garantir cette absence de retard, la tâche bloque les interruptions pendant son cycle d'écriture du GPIO puis les débloque.

scénario

La tâche temps réel d'écriture sur GPIO est affectée de la plus haute priorité au sein du système. Une application de stress dans l'espace temps réel programme un timer pour générer une interruption toutes les microsecondes. L'utilisateur produit le chronogramme de ces deux tâches exécutées en concurrence.

résultats attendus

Le chronogramme valide que la tâche temps réel n'est pas interrompue pendant sa séquence d'écriture sur le GPIO.

8.7 - Timer de haute résolution

but

Planifier des échéances avec une précision de l'ordre de la nano seconde.

description

Un timer périodique est programmé à la nano seconde près pour réveiller une tâche temps réel. Une application de stress système s'exécute en concurrence de cette tâche temps réel.

scénario

La tâche temps réel est programmée avec une période définie à la nano seconde près. L'utilisateur paramètre l'enregistrement de chronogramme puis exécute cette tâche temps réel en concurrence d'une instance de l'application de stress dans l'espace non temps réel.

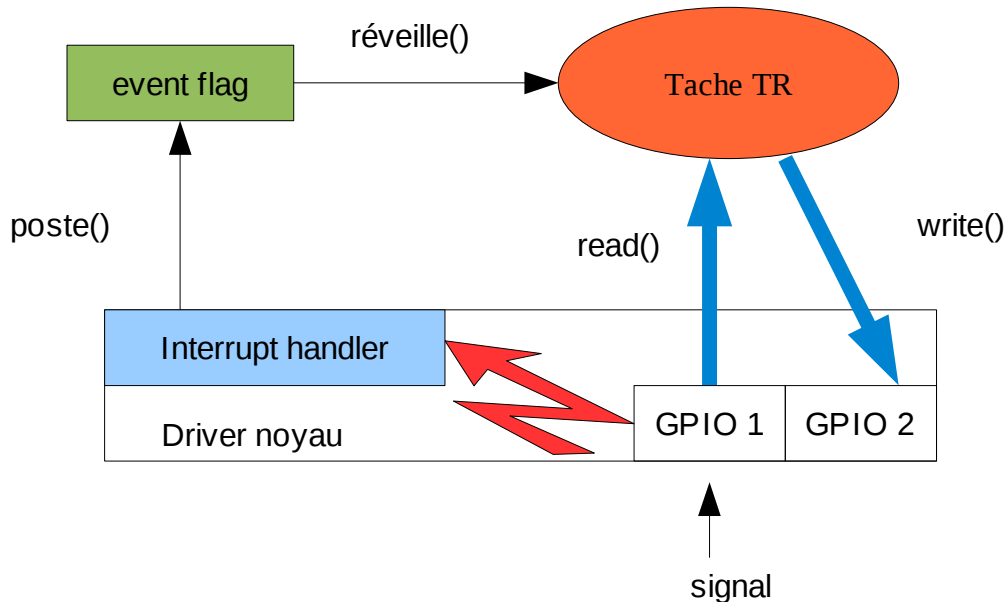
scénario alternatif

L'application de stress est exécutée dans l'espace temps réel avec une priorité plus faible que la tâche périodique.

résultats attendus

Le chronogramme valide le réveil de la tache temps réel avec une précision aussi proche que possible de celle programmée. La gigue au réveil est constante et faible.

8.8 - Asservissement



but

Enchaîner les lecture et écriture de GPIO, dans le cadre d'un asservissement, suivant une échéance.

description

Les modèles de réception et production de signal sur GPIO présentés précédemment sont mis en œuvre pour réaliser un asservissement.

La tache temps réel détient la priorité la plus forte au sein du système, elle lit une position sur le GPIO1 et produit une consigne sur le GPIO2 avant l'échéance de fin. En concurrence une application stress le système.

scénario

L'utilisateur charge le driver au sein du noyau puis exécute l'application portant la tache temps réel en concurrence de l'application de stress. Un équipement distant produit un signal sur le GPIO1. La tache temps réel écrit le GPIO2 dans une échéance de 1 microseconde après la levée de l'interruption de GPIO1. Le temps écoulé entre l'arrivée des données sur les GPIO1 et GPIO2 est mesuré avec un analyseur logique. L'application de stress est exécutée dans l'espace non temps réel puis dans l'espace temps réel. La mesure est faite dans les deux cas.

scénario alternatif

La tache temps réel écrit le GPIO2 dans une échéance de 100 nanosecondes après la levée de l'interruption de GPIO1.

scénario alternatif 2

La tâche temps réel écrit le GPIO2 dans une échéance de 10 nanosecondes après la levée de l'interruption de GPIO1.

résultats attendus

L'arrivée du signal sur le GPIO1 est notifié dans un temps constant quelque soit la charge du système.

8.9 - Temps réel et réseau

8.9.1 - Utilisation de la stack Linux

8.9.1.1 - Concurrence de flux d'applications temps réel

But

Vérifier que des applications temps réel réseau en concurrence arrivent à fonctionner en respectant les exigences de chacune.

description

Ce use case met en œuvre des applications temps réel dont les priorités d'exécution sont différentes. On pourra, pour ce genre de tests, utiliser des applications de VOIP avec des caractéristiques différentes en prenant garde de ne pas mettre en œuvre de QOS réseau particulière.

Au moins 3 machines sont mis en œuvre sur le même sous réseau, chacune connectée au même switch. On considère que la configuration réseau est faite, soit manuellement, soit par DHCP.

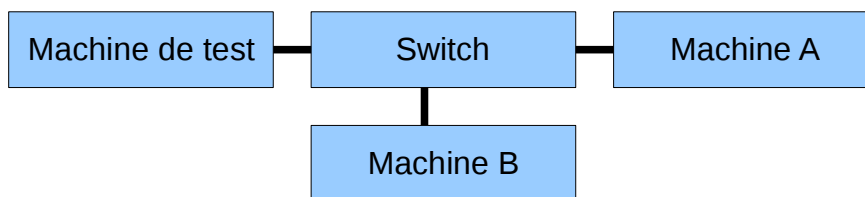
scénario

2 de ces machines (A et B) jouent le rôles des distants. Ce sont typiquement des PC puissants.

Une première application de VOIP est établie depuis la machine de teste vers la machine A.

Une deuxième application de priorité moindre établit un flux de VOIP vers la machine B.

Pour chacune de ces applications les flux générés doivent être calibrés de façon à être significatif en terme de nombre de paquet par seconde et en terme de débit.



scénario alternatif

En plus des flux établis précédemment, des applications de stress de type Iperf établissent des connexions supplémentaires entre la machine de test et les machines A et B.

résultats attendus

Si de la latence, de la perte de paquets et de la gigue sont observées elle doivent toucher d'abord les applications non temps réel (iperf), puis les applications temps réel en commençant par celles qui ont les priorités les plus faibles.

8.9.1.2 - Concurrence de flux d'applications temps réel de même priorité

but

Vérifier que plusieurs instances d'un même type de tâche réseau se voient attribuer des ressources temporelles équivalentes.

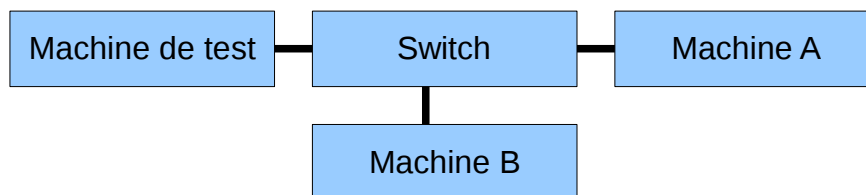
description

Ce use case mets en œuvre plusieurs applications de VOIP ayant exactement les mêmes caractéristiques réseau et ayant aussi les mêmes priorités. Malgré la présence d'éléments perturbateurs, les applications devraient conserver leurs caractéristiques de pertes, retards et giges. Au pire, les retards et les giges dûs au système d'exploitation devraient être insignifiants.

Au moins 3 machines sont mises en œuvre sur le même sous réseau, chacune connectée au même switch. On considère que la configuration réseau est faite, soit manuellement, soit par DHCP.

scénario

2 de ces machines (A et B) jouent le rôles des distants. Ce sont typiquement des PC puissants. Des flux de VOIP sont établis par des applications concurrentes sur la machine de test en direction de la machine A et de la machine B.



scénario alternatif

En plus des flux établis précédemment, des applications de stress de type Iperf établissent des connexions supplémentaires entre la machine de test et les machines A et B.

résultats attendus

Les applications temps réel de même priorité doivent être servies de manière régulière et équitable. Les caractéristiques de retard et de gigue dûs au système d'exploitation de la machine de test devraient rester stables quelque soient les conditions de test. De plus elles sont équivalentes pour chacune des connexions de VOIP.

Lors que des applications de stress sont mises en œuvre, les flux de VOIP doivent être protégés : il ne doit pas y avoir d'évolutions significative des pertes de paquets, des latences et des giges.

8.9.1.3 - Concurrence de flux d'applications temps réel avec ceux d'applications non temps réel

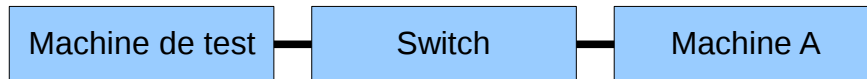
but

Vérifier que le temps réel et la QOS réseau protège efficacement les applications temps réel réseaux des flux initiés par des applications non temps réel agressives.

description

Ce test consiste à mettre en concurrence une ou plusieurs applications temps réel réseau avec des applications de stress réseau non temps réel.

Pour l'ensemble des applications, les flux sont établis depuis une machine de test vers une machine distante A.

**scénario**

Des flux de VOIP sont établis à partir d'applications temps réel entre la machine de test et un distant A. Ces flux de VOIP sont protégés par de la QOS simple comme décrit dans les modèle d'application de VOIP

Des flux de stress sont établis en concurrence de ces flux de VOIP. Ces flux sont configurés de façon à envoyer des burst importants.

scénario alternatif

Le même scénario que précédemment est mis en œuvre. La seule différence est que les flux de VOIP ne bénéficient plus de QOS

résultats attendus

Lors de ce test, il devrait être observé que les flux temps réel protégés par de la QOS ne voient pas de perte de paquets et ne voient pas de dérive de la latence et de la gigue.

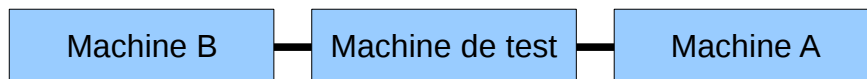
Si la QOS est désactivée, on devrait observer des pertes de paquets de la dérive et de la gigue.

8.9.1.4 - Concurrence de flux routés ou bridgés au travers de la machine.**but**

Vérifier que les flux routés traversant la machine ne perturbe pas les applications temps réel de la machine.

description

Plusieurs équipements Sagem Communications doivent pouvoir être traversés par des flux routés ou bridgés sans pour autant observer de baisse du déterminisme ou de performances. Ce cas d'utilisation traduit ce genre de configuration.



scénario

Une application temps réel à forte contrainte temporelle est lancée sur la machine de test. Elle doit mettre en œuvre des tâches très sensibles à la latence et à la gigue. Cette application n'est pas forcément orientée réseau mais doit occuper un taux significatif de CPU.

Des flux réseaux sont établis entre les machines A et B de façon à être routé ou bridgé au travers de la machine de test

résultats attendus

On ne devrait pas observer au niveau de la tâche temps réel de dérive de la latence et de la gigue.

8.9.2 - Utilisation de RTNet

Les mêmes uses cases que ceux utilisés pour la stack linux sont mis en œuvre. La seule grosse différence vient du fait que la mise en œuvre de la QOS pour protéger les flux temps réel s'avère inutile.

8.9.2.1 - Concurrence de flux d'applications temps réel avec RTNet**But**

Vérifier que des applications temps réel réseau en concurrence arrivent à fonctionner en respectant les exigences de chacune.

description

Ce use case met en œuvre des applications temps réel dont les priorités d'exécution sont différentes. On pourra, pour ce genre de tests, utiliser des applications de VOIP avec des caractéristiques différentes.

Au moins 3 machines sont mis en œuvre sur le même sous réseau, chacune connectée au même switch. On considère que la configuration réseau est faite, soit manuellement, soit par DHCP.

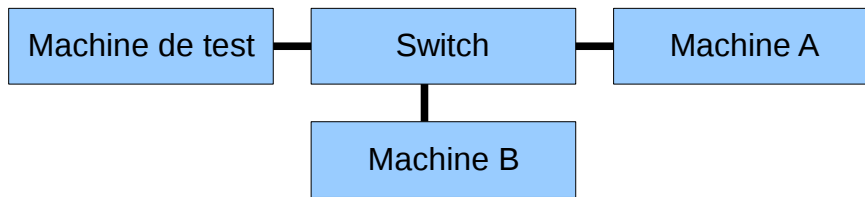
scénario

2 de ces machines (A et B) jouent le rôle des distants. Ce sont typiquement des PC puissants.

Une première application de VOIP est établie depuis la machine de teste vers la machine A.

Une deuxième application de priorité moindre établit un flux de VOIP vers la machine B.

Pour chacune de ces applications les flux générés doivent être calibrés de façon à être significatif en terme de nombre de paquet par seconde et en terme de débit.



scénario alternatif

En plus des flux établis précédemment, des applications de stress de type Iperf établissent des connexions supplémentaires entre la machine de test et les machines A et B.

résultats attendus

Si de la latence, de la perte de paquets et de la gigue sont observées elle doivent toucher d'abord les applications non temps réel (iperf), puis les applications temps réel en commençant par celles qui ont les priorités les plus faibles.

8.9.2.2 - Concurrence de flux d'applications temps réel de même priorité avec RTNet

but

Vérifier que plusieurs instances d'un même type de tâche réseau se voient attribuer des ressources temporelles équivalentes.

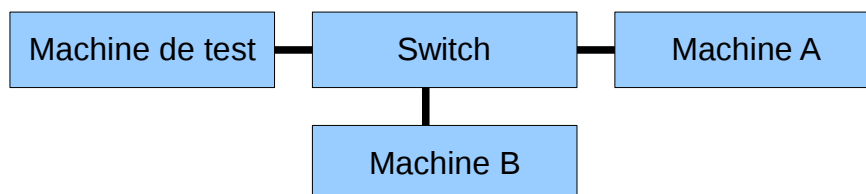
description

Ce use case mets en œuvre plusieurs applications de VOIP ayant exactement les mêmes caractéristiques réseau et ayant aussi les mêmes priorités. Malgré, la présence d'éléments perturbateurs, les applications devraient conserver leurs caractéristiques de pertes, retards et giges. Au pire, les retards et les giges dûs au système d'exploitation devraient être insignifiants.

Au moins 3 machines sont mises en œuvre sur le même sous réseau, chacune connectée au même switch. On considère que la configuration réseau est faite, soit manuellement, soit par DHCP.

scénario

2 de ces machines (A et B) jouent le rôles des distants. Ce sont typiquement des PC puissants. Des flux de VOIP sont établis par des applications concurrentes sur la machine de test en direction de la machine A et de la machine B.



scénario alternatif

En plus des flux établis précédemment, des applications de stress de type Iperf établissent des connexions supplémentaires entre la machine de test et les machines A et B.

résultats attendus

Les applications temps réel de même priorité doivent être servies de manière régulière et équitable. Les caractéristiques de retard et de gigue dûs au système d'exploitation de la machine de test devraient rester stables quelque soient les conditions de test. De plus elles sont équivalentes pour chacune des connexions de VOIP.

Lors que des applications de stress sont mises en œuvre, les flux de VOIP doivent être protégés : il ne doit pas y avoir d'évolutions significative des pertes de paquets, des latences et des giques.

9 - Notation des cas d'utilisation

Ce paragraphe décrit l'importance du besoin associé à chacun des cas d'utilisation

Nom du cas d'utilisation	notation
8.1.1 - Mesure de charge et temps CPU	fort
8.1.2 - Mesures de retards	fort
8.1.3 - Mesures de giges	fort
8.1.4 - Production de l'histogramme des latences	fort
8.1.5 - Chronogramme	fort
8.2.1 - Occupation complète du CPU (UCR 1)	fort
8.2.2 - Préservation du temps CPU attribué aux taches temps réel lors de stress non temps réel	moyen
8.2.3 - Stress temps réel et sauvegarde des caractéristiques des taches temps réel prioritaires	moyen
8.2.4 - Répartition équitable des ressources CPU entre les taches de même priorité	fort
8.2.5 - Concurrence d'un traitement de flux avec une tache très prioritaire	fort
8.2.6 - Priorités des taches et garantie des performances	moyen
8.2.7 - Taches temps réel et priorité d'accès à la mémoire de masse	fort
8.2.8 - Ressources partagées entre taches temps réel	moyen
8.2.9 - Prémption d'une écriture sur un file system	faible
8.2.10 - IPC	faible
8.2.11 - Communication entre les taches de l'espace utilisateur et noyau	moyen
8.2.12 - Consommation à date précise	fort
8.2.13 - Synchronisation de flux	fort
8.2.14 - Traitement de flux et écriture sur flash	fort
8.3 - Driver dans l'espace utilisateur (ULDD)	faible
8.4 - Acquisition usb	faible
8.5 - Signal sur un GPIO	faible
8.6 - Accès à un GPIO non interrompu	fort
8.7 - Timer de haute résolution	moyen
8.8 - Asservissement	moyen
8.9.1.1 - Concurrence de flux d'applications temps réel	fort
8.9.1.2 - Concurrence de flux d'applications temps réel de même priorité	fort
8.9.1.3 - Concurrence de flux d'applications temps réel avec ceux d'applications non temps réel	fort

8.9.1.4 - Concurrence de flux routés ou bridgés au travers de la machine.	fort
8.9.2.1 - Concurrence de flux d'applications temps réel avec RTNet	faible
8.9.2.2 - Concurrence de flux d'applications temps réel de même priorité avec RTNet	faible