



MÉMOIRE DE STAGE MASTER 2

Stage du 1^{er} février 2023 au 31 juillet 2023

Intitulé : Interruptions en espace utilisateur pour le réseau
BXI

Charles GOEDEFROIT

Encadré par Alexandre DENIS (Inria), Grégoire PICHON (Atos) et
Mathieu BARBE (Atos)

25 juillet 2023

Table des matières

1	Introduction	2
1.1	Présentation Inria	2
1.2	Présentation Atos (Eviden)	2
1.3	Environnement de travail	2
1.4	Le cadre	2
1.5	Présentation du plan	3
2	Context	3
2.1	HPC	3
2.2	OS bypass	4
2.2.1	Les interruptions matérielle	5
2.2.2	TODO : trouvé un titre	5
2.3	Polling	5
2.3.1	Inconvénients du polling	6
2.4	BXI	6
2.5	MPI	6
2.5.1	Les communications point à point	7
2.5.2	Les communications collective	7
2.5.3	La progression en tâche de fond	7
2.6	NewMadeleine	8
2.7	Travaux antérieur	8
3	Problématiques / Objectifs	9
3.1	Sujet	9
3.1.1	Nouveau mécanisme d'interruption en espace utilisateur	9
3.1.2	contentions...	9
3.2	Projet global	9
3.2.1	Les objectifs	9
3.3	Objectifs du stage	10
3.4	La suite	10
4	Exploration de uintr	10
4.1	Prérequis est accès	11
4.2	Fonctionnement des uintr	11
4.2.1	Les interruptions	11
4.2.2	Détail	13
4.2.3	Capacité présente et futur	14
4.2.4	Exemple	14
4.2.5	Partage du FD	15
4.2.6	+	15
4.3	Tests du mécanisme	16
4.4	Correction du patch pour l'appel système uintr_alt_stack	16

4.5	Mesure de la latence (Feedback Mathieu)	16
4.6	Performances	16
5	Intégration dans NewMadeleine	16
5.1	Présenté NewMadeleine details	16
5.2	Modification	17
5.3	Suite de tests	18
5.4	Performances	18
5.4.1	Résultats avec attente active	18
5.4.2	Résultats recouvrement communication par du calcule	18
6	Bilan	18
7	Remerciements	18
8	Annexes	19

1 Introduction

J'ai effectué mon stage de 6 mois dans l'équipe-projet TADaaM du Centre Inria de l'université de Bordeaux. Le sujet du stage à été proposé par Alexandre Denis (Inria) et Grégoire Pichon (Atos). Ils ont aussi encadré le stage avec Mathieu BARBE (Atos).

1.1 Présentation Inria

Inria est l'institut national français de recherche en sciences et technologies du numérique. Il compte plus de 3 900 chercheurs et ingénieurs au sein de 215 équipes-projets. La plupart des centres sont communs avec une grande université.

1.2 Présentation Atos (Eviden)

Atos est un des leaders internationaux de la transformation numérique. Elle couvre un champ large d'activité : le cloud, la cybersécurité, les services transactionnels, le conseil, l'infogérance, le Big Data, les supercalculateurs... Atos compte 112 000 collaborateurs. Atos est en restructuration pour ce séparé en 2 entités. L'entité qui nous intéresse pour ce stage est Eviden qui englobe notamment les activités autour des supercalculateurs et du HPC. Cette restructuration est récente donc quand Atos est évoqué dans ce document nous parlons de la partie Eviden.

1.3 Environnement de travail

Mon environnement de travail c'est les locaux du Centre Inria de l'université de Bordeaux. On a mis à disposition un bureau dans l'open space de l'équipe-projet TADaaM. Je peux participer et assister à des activités scientifiques intéressantes (séminaire, soutenance de thèse, soutenance HDR, activités diverses...). Nous avons accès aux salles de visio conférence, notamment pour les réunions de suivi de stage hebdomadaire. Il y a aussi un Baby-foot, une cafeteria, une petite Médiathèque...

1.4 Le cadre

Du côté de l'équipe TADaaM de Inria l'objectif de l'équipe est de faire de la recherche sur les sujets suivants :

- Gestion des I/O (ordonnancement, bande passante...)
- Placement de processus
- Partitionnement de maillage (i.e. SCOTCH)
- Localité matérielle (i.e. hwloc)

- Optimisation des communications pour les réseaux haute performance, MPI (i.e. NewMadeleine)

Elle est composée de chercheurs, ingénieurs de recherche, post-doc, doctorant et stagiaire. La culture informatique de l'équipe est l'utilisation des environnements linux, des logiciels open source, des cluster de calcul HPC, le traitement des données et le système. L'équipe a mis à ma disposition un ordinateur portable avec une station de travail brancher à un écran, à internet, à un clavier et une souris. Inria donne aussi accès à un ensemble de services, boîte email, service de communication Mattermost, service de visio Webex, intranet avec le menu de la cafeteria... Nous sommes libre d'installer le système d'exploitation et d'utiliser les outils informatiques que l'on veut.

Du côté d'Atos l'équipe BXI-LL (BXI Low Level) est une équipe dont l'objectif est de fournir le support logiciels bas niveau pour les cartes réseaux BXI. Cela consiste à développer les drivers de la carte, le support Lustre – un système de fichier parallèle et distribué qui est utilisé pour l'I/O – ne fait pas partie de ce stage – ainsi que l'interface logiciels Portal4. Cette interface permet aux implémentations MPI d'utiliser les réseaux BXI. Le personnel est composé d'ingénieurs, de doctorant et de stagiaire. /* TODO : vérifier */ La culture informatique de l'équipe est l'utilisation des clusters HPC, la programmation système et l'utilisation d'un système de sécurité PKI pour accéder aux ordinateurs et aux services internes. Atos nous a donné accès à une machine avec deux processeurs Intel® Sapphire Rapids. Pour accéder à cette machine ils nous ont fourni un accès ssh et un compte VPN. Nous avons eu tous les droits d'accès sur cette machine pour changer le noyau du système.

1.5 Présentation du plan

Dans ce mémoire nous allons commencer par une présentation du contexte dans lequel le stage se place. Puis nous continuerons avec les objectifs à long terme et les objectifs du stage. Nous verrons ensuite le nouveau mécanisme d'interruption et ce que nous avons fait avec. Nous continuerons par la présentation de l'intégration de ce mécanisme dans la bibliothèque de communication NewMadeleine et les tests que l'on a effectués. Pour finir un bilan du stage ainsi que les travaux qui vont suivre.

2 Contexte

2.1 HPC

Le stage s'est déroulé dans le contexte du Calcul Haute Performance (HPC, High-Performance Computing en anglais). Le HPC utilise des supercalculateurs pour la simulation numérique et le pré-apprentissage d'in-

telligences artificielles. Ces simulation simule la dynamique des fluides, la résistance structurelle, les interaction moléculaire, les flux d'aire...

Elles couvrent différent domaines :

- L'industrie : le médicales, l'automobiles, l'aviations, la constructions, l'aérospatiales...
- La défense : simulation atomique
- La recherche scientifiques : la création des galaxie, la fusion nucléaire, le climat...
- La météo

De nos jour les supercalculateurs sont composé de plusieurs ordinateur que l'on appelle noeud de calculs. Ceci sont regroupé en grappe (cluster), il sont tous vue est utiliser comme une seul grande machine. Ce fonctionnement pose des questions sur la répartition du calcul, la distribution de la mémoire, la communication entre les différent noeuds... Dans le contexte du stage, nous nous sommes concentrés sur les communications entre noeuds. Les noeuds sont connecté entre eux par un réseau haute performance. Ce réseau est dédiés au communications est n'est pas forcément de type Ethernet. La gestion des noeuds est généralement effectuée par un second réseau plus traditionnel (Ethernet ; TCP/IP). Les réseaux haute performance ont une latence autour de quelque microseconde. /* TODO : insisté sur la différence entre les réseaux pour le HPC et les réseaux classiques */ Chaque noeuds possède une ou plusieurs carte réseau et il faut les programmé.

2.2 OS bypass

Le stage ce passe donc aussi dans un context système. Chaque noeuds à sont propre système d'exploitation qui permet la gestion des ressources, des processus, des fichiers, des périphériques. Pour cela le système à 2 espaces :

- un espace noyau où seul le code du système peut s'exécuter. Le code du système peut donc modifié n'importe quelle endrois de la mémoire, exécuté n'importe quelle instructions...
- un espace utilisateur où le code de l'utilisateur est exécuter. Cette espace est limité par le noyau qui controle ce que fait l'utilisateur.

En temps normale les périphériques sont programmé directement depuis le noyau du système d'exploitation ceci pour des questions de sécurité, de standardisation des accès... Lors que l'on passe par le noyau (kernel en anglais) on à un surcoût qui n'est pas négligeable dans notre cas. Pour passer par le noyau on utilise généralement des appels système qui ce présente sous la forme d'une fonction. Un appel système vas effectué un changement de contexte (context switch) pour passé de l'espace utilisateur à l'espace noyau.

Ce changement de context est coûteux car il sauvegarde les états du code de l'utilisateur avant d'exécuté celui du noyau. Une fois le context switch effectué le code noyau de l'appel système s'exécute avant de refaire un context switch pour cette fois passé de l'espace noyau à l'espace utilisateur, et donc restoré l'états du code de l'utilisateur. Donc le fait de passé par un appel système coûte plusieurs microseconde. On vois donc qu'il n'est pas préférable d'utiliser les appels système car un appel système est déjà du même ordre de temps qu'une communication. En HPC on programme donc directement la carte réseau à partir de l'espace utilisateur (OS bypass en anglais). Pour cela on initialise toujours la carte à partir du noyau mais on fait une projection de la mémoire et des registre de la carte réseau dans l'espace d'adressage virtuelle du processus utilisateur.

2.2.1 Les interruptions matérielle

Les interruptions matérielle, aussi nommé IRQ pour Interrupt ReQuest, existe depuis long temps est serve notamment à remonté des exceptions du processeur au noyau. Par exemple elle sont utilisé quand il y a une eu une division par zero. Elle on ensuite été utilisé pour signaler des événements en provenance de périphériques et aussi pour de la communications entre processus. Dans ce document nous utiliseront le terme interruption ordinaire ou IRQ pour les désigner.

2.2.2 TODO : trouvé un titre

Pour transmettre des événement à l'utilisateur les périphériques utilise généralement les interruptions ordinaire, qui passe par le système donc on ne les utilise très peu en HPC. En HPC on fait donc du polling.

2.3 Polling

Le polling consiste à scruté (poll) régulièrement si on à reçus un événement. Concrètement cela consiste à lire une zone mémoire modifier par la carte réseau et voire si un bit est passé à 1. Pour cela il es possible de dédié un thread qui vas faire de l'attente active, scruté sans cesse si un événement à été reçus. Mais on perd une unité de calcule lors ce que le thread est ordonnancer donc de la puissance de calcul, donc cette technique est peu utilisé. Il est aussi possible d'entrelacés le code de l'utilisateur avec des scrutation, c'est fréquemment utiliser mais ça oblige à l'utilisateur de prendre en compte la progression. Une autre solution qui est utiliser par Pioman dans NewMadeleine (Nous en parlerons plus tard) consiste à faire ces scrutation de manière opportuniste dans les threads qui on fini leur calcul, mais pour cela il faut déjà utilisé plusieurs thread et avoir une application qui à des calculs d'une durée hétérogène.

Il faut donc, peut importe la technique, régulièrement scruté pour faire progressé les communications.

2.3.1 Inconvénients du polling

Dans le cas d'un thread dédié qui fait de l'attente active la réactivité est excellente hormis quand il y à plus de thread que d'unité de calcul est que le thread n'est pas ordonnancer. /* TODO : cité ce papier <https://inria.hal.science/hal-03695835v1> et modifier */ Quand on entrelaces le calcul est les scrutations la réactivité est moins bonne car il faut attendre que le calcul sois fini pour faire un poll. C'est ce qui est fait habituellement.

Quand on utilise les thread de façon opportunist pour faire des scrutations la réactivité est moins bonne car il faut qu'il y est un thread disponible.

Donc un choix doit être fait entre perdre de la capacité de calcul ou perdre en réactivité. En plus on perd un peu de temps de calcul à effectué du polling.

2.4 BXI

BXI pour Bull eXascale Interconnect est un type de réseau d'interconnection (réseau haute performance) développé par Atos. Historiquement développé par Bull qui à été racheté par Atos. Ce réseau est dédié aux communications entre noeuds. Il est composé de carte réseau BXI et de switch BXI. La carte BXI est capable de faire progressé le communication réseau sans aucune intervention du processeur (offload des communications réseau). Le processeur à juste à soumettre une commande dans une file sur la carte et elle s'occupe de tous. Le processeur peut ensuite récupéré une file d'événements pour savoir ce qu'il c'est passé, en somme faire un poll. La carte est également capable de déclenché des interruptions. L'utilisation de la carte passe donc par une implémentation du protocole Portal4. L'utilisateur utilise donc le protocole pour envoyer et recevoir des paquets réseau.

2.5 MPI

MPI pour Message Passing Interface est un standard pour fournir une interface pour effectué des communications entre plusieurs processus, qui sont souvent sur des noeuds différent et qu'on appel *processus MPI*. Cette interface fourni une abstraction pour transmettre des données entre plusieurs processus. L'abstraction masque la complexité des communications. L'interface permet d'envoyer et de recevoir des messages. Pour cela il y a deux mode de communications :

2.5.1 Les communications point à point

C'est à dire entre deux processus MPI, aussi appelé One-to-One. Pour ce faire le processus MPI récepteur vas appelé la fonction `MPI_Recv` qui est bloquante et vas attendre la reception d'un message. L'émetteur vas lui faire un appel à la fonction `MPI_Send` qui est aussi bloquante et vas envoyer un message est attendre que la communication sois fini. L'utilisation des fonctions `MPI_Send` / `MPI_Recv` bloque le code ce qui nous fait perdre du temps à attendre. La nome propose aussi une version non bloquante de ces fonctions qui sont `MPI_Isend` et `MPI_Irecv`. Cette version ce contente de posté la communication et rend immédiatement la main. Pour la progression et verifier la terminaison il faut donc d'autre fonctions `MPI_Test` qui vérifie la progression, et la fait si nécessaire, et la fonction `MPI_Wait` qui attend activement la terminaison et s'occupe de la progression si nécessaire. Il est important de noté que le standard ne précise pas si la progression ce fait en tâche de fond ou non, c'est aux implémentations de la nome MPI de choisir. C'est pour cela que la progression peut ce faire au niveau des fonctions `MPI_Wait` et `MPI_Test` ou être fait avant est donc l'appel aux fonctions s'occupe juste de la terminaison. L'envoi des messages peut donc être asynchrone. /* gardé cette phrase ? */

2.5.2 Les communications collective

Les communications collective ce fond entre plusieurs processus. Il en existe de différent type :

- un processus vers plusieurs (One-to-All) par exemple un broadcast d'un message
- de plusieurs processus vers un seul (All-to-One) par exemple une reduction (e.g. un processus reçoit la somme des valeurs des autre processus)
- de plusieurs processus vers plusieurs (All-to-All) par exemple quand tous les processus on un message pour les autre

Pour les collective il existe également deux versions, bloquante et non bloquante, qui fonctionne de la même façon que les communications point à point.

2.5.3 La progression en tâche de fond

Pour faire progressé les communications en tâche de fond il est possible de :

- faire des appels à `MPI_test` régulièrement et faire du calcule entre chaque appel. Cela nous permet de recouvrir la latence des communications par du calcule. La progression ce faire également dans d'autre

appel aux fonctions MPI. Quand il n'y a plus de calcul à faire on repasse à une progression bloquante par un appel à `MPI_Wait` sauf si la communication est déjà fini.

- utilisé un thread dédié aux progressions. Dans ce cas c'est la bibliothèque MPI qui s'occupe de la progression des communications en tâche de fond grâce à un thread dédié. Il faut donc faire attention au placement des threads et à prendre en compte qu'un thread est déjà utilisé par la bibliothèque de communication. Il faut aussi éviter d'appeler trop souvent `MPI_Test` car ça crée de la contention.
- utilisation des threads de façon opportuniste c'est à dire quand un des thread à fini son calcul il va faire progresser les communications. C'est ce qui est fait par Pioman dans NewMadeleine.

2.6 NewMadeleine

NewMadeleine est une bibliothèque de communications qui supporte le RPC (Remote procedure call) et implémente aussi une interface MPI. On peut donc la considérer comme une implémentation du standard MPI avec des fonctionnalités supplémentaires. Elle est développée au Centre Inria de l'université de Bordeaux. Elle est basée sur un système de progression événementielle qui lui permet d'être asynchrone. Elle est composée en modules ce qui lui permet de charger dynamiquement des stratégies d'optimisation sur les paquets. Les stratégies sont l'agrégation de paquets, l'utilisation de priorités et le multi-rail /* TODO : précision multi-rail/split_balance */. Elle possède également un système de drivers pour supporter différents types de communications comme des réseaux (e.g. portals4 pour BXI, ibverbs pour InfiniBand, psm2 pour OmniPath, ofi (Open Fabrics Interfaces), TCP) et localement (shm (mémoire partagée), socket, self).

2.7 Travaux antérieurs

TODO :

- système de progression dans NewMadeleine
- progression avec pioman
- overlap
- Travaux de Mathieu Barbe durant un stage en 2019 sur l'utilisation d'interruption pour transmettre des événements réseau.
- Ces travaux vis à diminuer la latence du fait de passer par un driver noyau.
- Il parle dans la perspective de traitement direct des interruptions depuis l'espace utilisateur ce qui mène à ce stage.

TODO : pu / core logique, core, CPU, processeur...

3 Problématiques / Objectifs

3.1 Sujet

Le sujet est *Interruptions en espace utilisateur pour le réseau BXI*.

3.1.1 Nouveau mécanisme d'interruption en espace utilisateur

Ce nouveau mécanisme qui permet de dérouler une interruption à partir de l'espace utilisateur et très récent. Il est seulement disponible sur les processeur Intel® Sapphire Rapids qui sont officiellement sortis le 10 janvier 2023. La plupart des processeur ont été disponibles à la vente le 14 mars. AMD n'a pas encore annoncé de support pour les interruption en espace utilisateur.

3.1.2 contentions...

Le fonctionnement actuelle de la progression des communications amène à des problèmes de contention mémoire car plusieurs threads peuvent chercher à lire / modifier la même zone mémoire. Utiliser les interruptions permettra de ne plus avoir ce problèmes car seul le thread concerné par une zone mémoire serai prévenu.

TODO :

3.2 Projet global

Dans la plupart des autre domain les périphériques envoi une interruption ordinaire à l'application, par le bais des signaux ou d'un appel système bloquante, pour avertir d'un changement. L'idée serai de faire la même chose en HPC grâce au interruption en espace utilisateur. Le projet globale vise donc à faire progresser les communications entre plusieurs noeuds du réseau BXI sans faire de polling et en utilisant les interruptions en espace utilisateur. Cela permettrai de réduire globalement le temps de calcul d'une application. Pour ce faire la carte réseau BXI devra être capable de levé des interruptions en espace utilisateur. Le fait de supprimer le polling et de réduire le temps de calcul permettra de diminué la consommation électrique.

3.2.1 Les objectifs

Il y a plusieurs objectifs, les principaux sont :

- La réduction du temps de calcul.
- utiliser des interruptions en espace utilisateur pour remplacé le polling.
- Simplifier pour l'utilisateur le recouvrement des communications par du calcul pour qu'il n'est plus à ajouter des `MPI_Test` en pleins milieu de ces calcul.

- Amélioré la réactivité des communications sans qu'il y est besoin d'une unité de calcul dédié à l'attente active.

Ce stage est donc une premier étape de ce projet global.

3.3 Objectifs du stage

Le premier objectif est de défricher le fonctionnement des interruption en espace utilisateur à partir des éléments suivant :

- Le manuel Intel[®] de l'architectures 64 et IA-32 pour les développeurs logiciels
- La présentation du mécanisme de Sohil Mehta, ingénieur Intel[®] qui à développé le patch noyau, qui est une diapositive associer à des discussion sur LWN.net /* TODO : cité */¹
- le patch du noyau linux avec ces manuels. /* TODO : cité */

Le second objectif est de connaître les propriétés du mécanisme et de mesuré ça performance. Le troisième objectif est de ne plus avoir à faire du polling que se sois dédié un tread ou utilisé les threads de façon opportunist, ne plus perdre du temps de calcule à poll et résoudre les problèmes de réactivité. Pour cela on envisage l'intégration de ces interruption dans le driver de mémoire partagé (shm) de NewMadeleine. Pour testé dans un premier temps le fonctionnement avec des communications entre processus (IPC²). Pour intégrer les interruption dans les drivers NewMadeleine il faut également permettre la progression des communications à partir d'un handler d'interruption. Le dernier objectif est de montré que l'utilisation d'interruption permet bien d'amélioré le recouvrement des communications par du calcul.

3.4 La suite

Les objects suivant du projet global seront traité dans une thèse qui fait suite au stage.

4 Exploration de uintr

Pour cette partir j'ai utiliser des connaissance personnel autour du système, du développement noyau, de Linux... et aussi ce que j'ai appris en cours de *programmation système*, de *système d'exploitation* et *architecture des ordinateurs*.

1. <https://lwn.net/Articles/869140/> /* TODO : déplacé */
 2. Inter Process Communication en anglais /* TODO : on garde pour certain acronyme ? */

4.1 Prérequis est accès

Pour utilisé le mécanisme d'interruption en espace utilisateur il faut donc avoir accès à une CPU Intel® Sapphire Rapids. Comme ils sont sortie très récemment ils sont assez difficile d'accès. Atos nous a, non sans difficulté donné accès, à une machine qui possède 2 processeur Intel® Sapphire Rapids qui sont des Intel® Xeon® Platinum 8470. Ils y a eu plusieurs difficulté pour avoir une machine, pour quelle soit installer et pour quelle soit dans un réseau au quelle on puisse accéder depuis l'Inria. On à donc eu un accès VPN qui utilisé l'ancien système de VPN car le nouveau ne fonctionné pas. Nous avons donc eu accès à la machine environ deux mois et demi après le début du stage. La machine est déjà configuré avec le système d'exploitation Red Hat Enterprise Linux (RHEL) 9.1. Elle possède deux carte BIX v2 que nous n'avons pas utilisé lors du stage.

Il faut aussi avoir une version patché du noyaux linux avec le support du nouveau mécanisme. Cette version patché n'est pas encore disponible dans la branche principal du noyau. Elle est disponible sur le GitHub d'Intel®. Nous avons donc télécharger cette version patché. Nous l'avons compiler et installer sur la machine. Lors de la compilation il faut activé le support des uintr (Voir figure en annexes5). Il est aussi possible d'activé le support qu'un thread bloqué, c'est à dire pas ordonnancer ou dans une appel système interruptible, puisse recevoir une uintr.

Le mécanisme utilise de nouvelle instructions, il faut donc une version récente du compilateur GCC pour compiler les programme utilisateur qui utiliseront les uintr. Il faut donc la version **11.3.0** ou plus récente de GCC, sur RHEL il faut la version **12.1.1** ou superieur. Le support n'est pas encore disponible dans d'autre compilateur comme LLVM-Clang. Pour compiler un programme utilisateur il faut précisé le flag de compilation `-muintr` pour les fichiers qui définisse un handler d'interruption ou qui utilise les nouvelles instructions.

4.2 Fonctionnement des uintr

4.2.1 Les interruptions

Pour commencer nous allons voir comment les interruption matérielle fonctionnent. Nous allons nous concentrer sur l'envoi d'interruption entre unité de calcul, donc entre processus. Pour fonctionné les CPU on a une unité dédié au traitement des interruption l'APIC pour Advanced Programmable Interrupt Controller. Cette APIC permet au système de enregistré un handler pour chaque interruptions. Pour ce faire le système a une table *IDT* (Interrupt Descriptor Table) qui contiens 256 entrées. Donc 256 interruptions possible, les valeurs entre 0 et 255 sont aussi appelé vecteur. Les vecteur compris entre 0 et 31 sont réservé pour les exceptions et interruption système, ce entre 32 et 127 sont réservé pour les interruptions pour les périphériques, 128 est réservé

pour les appel système et entre 129 et 255 pour des utilisation varier.

Il est important de savoir que chaque unité de calcul (coeur logique) possède un *APIC ID* physique. Petit fun-fact le *core ID* est un sous ensemble de l'*APIC ID*.

Pour déclencher une interruption il y a quatre possibilités :

1. Une exceptions déclencher par le CPU (e.g. division par zero, défaut de segmentation...).
2. Une instruction comme `INT80 numSysCall` pour déclencher un appel système ou bien `INT3` pour définir un point d'arrêt, ou encore `INT0`, `BOUND` et `INT n`.
3. Des pins du processeur dédié à la réception d'interruption lancer à partir d'un périphérique.
4. Demander à l'APIC en lui même grâce à un registre ICR pour Interrupt Command Register. Il existe un ICR par vecteur il faut donc écrire l'*APIC ID* du destinataire dans le ICR du vecteur que l'on veut déclencher. Seul le CPU et le noyau peuvent modifier les ICR.

On vois bien que les IRQ fonction au niveau du noyau et du CPU.

Nous allons voir un exemple d'envoi d'IRQ entre 2 processus en cours d'exécution. Tous d'abord l'initialisation ce fait au démarrage du système et consiste principalement à définir les handler noyau dans l'*IDT*. Au préalable il faut définir le vecteur utiliser, le handler noyau, la technique pour contacté le système et l'identification du récepteur (e.g. par un patch du noyau, par un module noyau...).

Nous allons maintenant voir les états de l'envois d'une IRQ montré sur la figure suivant 1.

- ① Le récepteur fait un appel système pour indique au noyau comment il veut être avertie d'une interruption (e.g. un handler utilisateur, un descripteur de fichiers qu'il vas lire, un appel système bloquante, une zone mémoire ou écrire...).
- ② l'émetteur peut donc avertir le noyau qu'il faut envoyer une interruption pour cela il peut utiliser un appel système ou écrire dans un descripteur de fichiers...
- ③ Le noyau détermine l'unité de calcul ou ce trouve le récepteur. Pour cela il peut utiliser par exemple un *PID* (Processus ID) donné par l'émetteur ou autre. Il vas donc pouvoir détermine le *APIC ID* de unité de calcul à interrompre.
- ④ Le noyau écrire donc l'*APIC ID* dans le *ICR* d'un vecteur détermine à l'avance. L'émetteur vas reprendre la main après un autre changement de contexte.
- ⑤ L'APIC vas donc interrompre le récepteur qui vas donc stoppé sont execution est passer dans le noyau. Une fois dans le noyau le handler

vas ce déclencher et exécuté le code prévue au préalable (déclenchement d'un handler utilisateur, écrire dans un descripteur de fichiers, écrire dans une zone mémoire...).

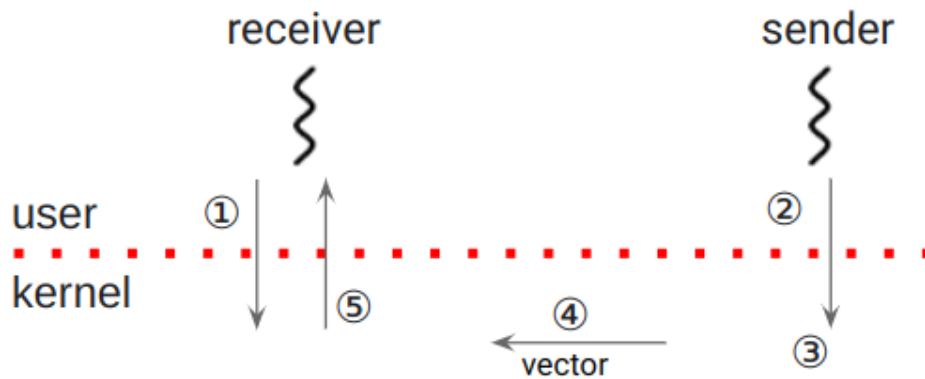


FIGURE 1 – L'envoi d'une interruption

Il est possible de masquer les interruptions grâce à deux instructions utilisable seulement par le noyau qui sont `clui` et `stui`. Ces interruption modifie un flag (*IF*) qui se trouve dans le registre d'états de l'unité de calcul `RFLAGS` (aussi nommé `EFLAGS` sur les architecture 32bits). La liste des interruptions pour les IRQ se trouve ici².

Comment on la vue ce mécanisme fonction totalement dans le noyau du système. Dans notre exemple il faut au minimum deux changement de contexte (context switch) pour le récepteur et l'émetteur et même peut être plus si le récepteur doit déclenché un handler coté utilisateur.

4.2.2 Détail

TODO : ...

Le mécanisme est composé de 5 instruction que l'on peut retrouvé dans le table TODO : ref.

- 5 instructions
- registre MSR pour l'init

Interruption	Interruption en espace utilisateur
cli (CLear Interrupt flag)	clui (CLear User Interrupt flag)
sti (SeT Interrupt flag)	stui (SeT User Interrupt flag)
	testui (Read User Interrupt flag)
iret (Interrupt RETurn)	uiret (User Interrupt RETurn)
APIC pin, APIC ICR <code>INT n</code> , <code>INT3</code> , <code>INT0</code> , <code>BOUND</code> , <code>INT80 n</code> et registre IRQ	sendipi <uipi_index>

FIGURE 2 – Instructions des interruption et interruption en espace utilisateur

4.2.3 Capacité présente et futur

- process => process
- kernel => process
- device => process
- au niveaux des thread...
- chaque thread peut avoir un seul thread de défini.
- Il existe 64 vecteur entre 0 et 63
- masquage
- le thread dois être en ring-3
- peut fonctionné si le thread est endormi mais j'ai pas testé car ...
- l'interruption est délivré si le processus est en mode utilisateur. Le comportement par défaut et que l'interruption est reçus quand le thread est à nouveau au espace utilisateur.
- Si on à configuré dans à la compilation du noyau il est possible de donné 3 flags au moment de lenregistrement du handler. Le premier flag `UINTR_HANDLER_FLAG_WAITING_ANY` pour activé le fait de pouvoir recesoire une interruption quand le processus n'est pas ordonacé ou est dans un appel système interruptible. Les 2 flague suivant `UINTR_HANDLER_FLAG_WAITING_RECEIVER`, `UINTR_HANDLER_FLAG_WAITING_TRANSMITTER` s'ajoute au flague précédent et précede si le cout du context switch est pour l'émetteur ou le recepteur de l'interruption.
- Le fonctionnement à partir du kernel est très similaire au fonctionnement des inrettrruption "ordinaire" plus l'appel du handler en espace utilisateur.

4.2.4 Example

- exemple avec schéma

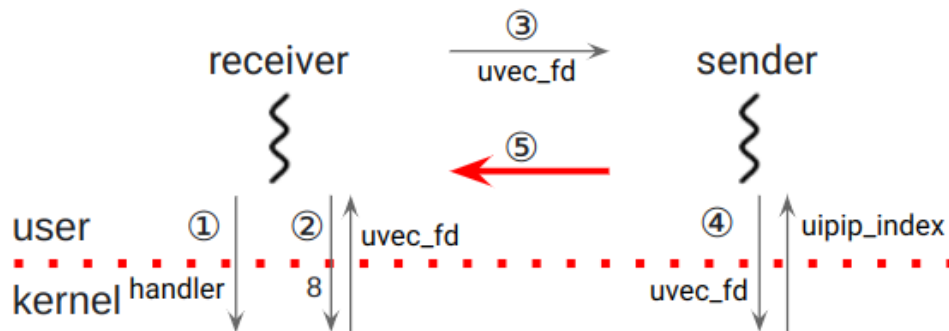


FIGURE 3 – Phase d’initialisation des uintr

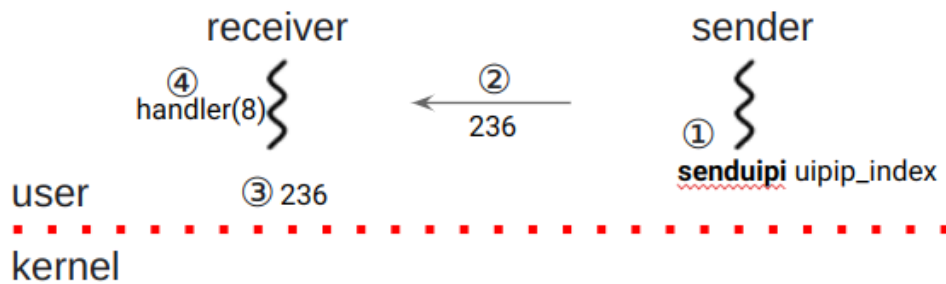


FIGURE 4 – L’envoi d’une uintr

4.2.5 Partage du FD

- Pour mes testes "jouer" j’ai utiliser `uintr_register_self()`
- pipe / socket / URL pour NewMadeleine (on en reparle en section XXX)
- inheritance / `pidfd_getfd` / sockets
- `pidfd_getfd`

4.2.6 +

- comparé au signaux ?
- TODO ...
- faire attetion tous les registre ne sont pas sauvegardé. La responsabilité de la sauvegarde est à l’utilisateur. Pour ça le compilateur permet de sauvegardé les registre généraux mais pas les vecteur SIMD. Donc il faut évité de les utilisé ou les sauvegardé avant.
- On ne peut pas attendre dans un handler donc on peut appelé seulement les focntion et appel système async safe, on en reparlera dans

le chapitre TODO : cite. En plus il faut faire attention aux opérations sur les chaînes de caractères de la libc memcpy, memmove, memset et memcmp qui utilisent des registres, il est possible de les inline sans utiliser de vecteur SIMD grâce au flag de compilation `-minline-all-stringops`.

4.3 Tests du mécanisme

- entre process
- entre threads
- avec alt stack
- test d'écrasement des interruptions
- avec binding (plus pourquoi avec binding que comparaison avec et sans)
- avec turbo boost

4.4 Correction du patch pour l'appel système uintr __alt __stack

TODO

4.5 Mesure de la latence (Feedback Mathieu)

TODO : expliquer que le binding est important

4.6 Performances

TODO

- Dans un premier temps j'ai pas bind les threads par conséquent j'avais de mauvais résultats.
- J'ai fait différents bind (on voit que c'est similaire sauf quand on passe entre 2 NUMA)
- si on monte la fréquence c'est mieux (turbo boost)
-

5 Intégration dans NewMadeleine

5.1 Présenté NewMadeleine details

- liste de progression recv
- liste de progression send
- p_pw
- post
- poll

- driver
- nm_schedule (appelé par nm_wait, ...)
- pioman
- existe : progression du nm_schedule ou de pioman vers le core_task

5.2 Modification

TODO : on désactive le poll avec les driver qui on des handler. Mais on fait toujours un poll qui fonctionne du premier coup.

- ajout d'un driver sig_shm
- ajout d'un driver uintr_shm
- progression à partir du handler du driver vers les core_task
- problématiques de gestion des interruptions
 - on ne peut pas faire d'attente dans les handler Les fonction doivent être async safe
 - on ne peut pas utiliser d'allocateur donc il faut utiliser des p_pw déjà alloué
 - pour avoir un p_pw disponible il faut faire progresser le communications
 - la progresser à une partie critique qui naissaisite un verrou
 - si on ne peut pas récupéré le core_lock (try_lock) il faut mettre à plus tard le trétement de l'interruption
 - on utilise donc une file lock-free
 - problématique des liste lock-free qui doivent être wait-free
 - le principe d'une lfqueue est d'attente si quelqu'un d'autre modifie la file
 - il faut donc une file wait-free
 - j'ai fait de la biblio
 - décrire les différante solutions
 - solution d'Alexandre
- gestion des multiple interruptions qui s'écrase, (prob_any / pour les large si la progression à traiter le pipeline courant)
- quand est ce qu'on envois des interruption ?
 - au moment ou on poste le premier paquet d'envois, pour indiquer au recepneur que des données sont disponible (le recepneur à toujours un paquet de reception posté)
 - au moment ou une progression est fait du coté du recepneur, pour indiquer une reception a l'émeteur
 - l'émeteur reçois une iterruption est détermine si l'émission est fini, petit paquet, ou si il faut envoyer la suite, paquet large.

5.3 Suite de tests

Tous les tests ne passe pas ?

5.4 Performances

5.4.1 Résultats avec attente active

TODO : voire le sur coup des interruption

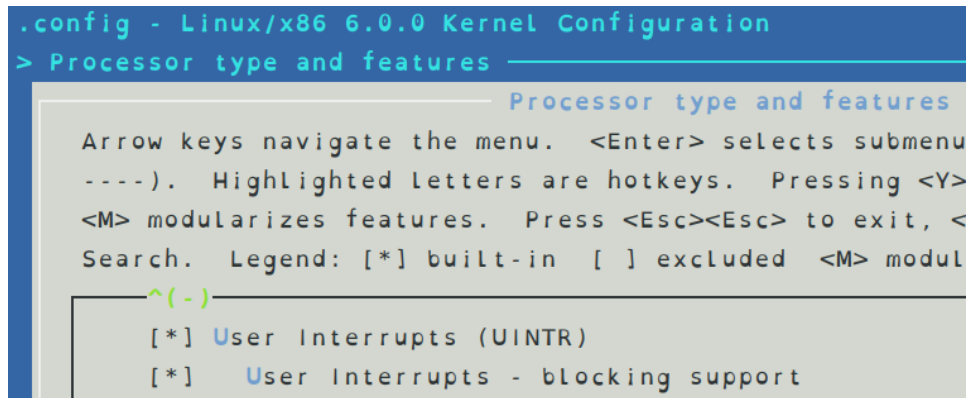
5.4.2 Résultats recouvrement communication par du calcule

TODO : courbe overlap reception

6 Bilan

7 Remerciements

8 Annexes



```
.config - Linux/x86 6.0.0 Kernel Configuration
> Processor type and features

Processor type and features
Arrow keys navigate the menu. <Enter> selects submenu
----). Highlighted letters are hotkeys. Pressing <Y>
<M> modularizes features. Press <Esc><Esc> to exit, <
Search. Legend: [*] built-in [ ] excluded <M> modul

^(-)
[*] User Interrupts (UINTR)
[*] User Interrupts - blocking support
```

FIGURE 5 – Activé le support des uintr à la compilation du noyau