

Mémoire de stage Master 2

Stage du 1er 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)

24 juillet 2023

Table des matières

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 étais proposé par Alexandre Alexandre Denis (Inria) et Grégoire Pichon (Atos). Ils on aussi encadré le stage aussi que Mathieu BARBE (Atos).

1.1 Presentation Inria

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

1.2 Presentation Atos (Eviden)

Atos est un des leader international de la transformation numérique. Atos couvre un champ large d'activité : le cloud, la cybersécurité, les services, le conseil, les supercalculateurs... Atos compte 112 000 collaborateurs. Atos est en restructuration pour ce séparé en 2 entités. L'entités qui nous intéresse pour ce stage est Eviden qui englobe les activité autour des supercalculateurs et du HPC. Cette restructuration est récente donc quand je parle d'Atos dans ce document je parle 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. J'ai un bureau dans l'open space de l'équipe-projet TADaaM. Je peut participé et assisté à des activités scientifique intéressantes (séminaire, soutenance de thèse, soutenance HDR, activité diverse...). J'ai accès au salle de visio conférence, notamment pour les réunion de suivie de stage hebdomadaire. Il y a aussi un Baby-foot, une cafeteria, une petite Médiathèque...

1.4 Le cadre

Du coté de l'équipe TADaaM de Inria l'objectif de l'équipe est de faire de la recherche. 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 calcule HPC, le traitement des données et le système. L'équipe à 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, boite email, service de communication Mattermost, service de visio Webex, intranet avec le menu de la cafeteria... J'ai étais libre

d'installer mon système d'exploitation, d'utilisé les outils informatique que je voulais...

Du coté d'Atos l'équipe BXI-LL (BXI Low Level) est une équipe don l'objectif est de fournir le support logiciels bas niveau pour les carte réseaux BXI. Cela consiste à développé les drivers de la carte, le support Lustre [1] ainsi que l'interface logiciels Portal4. Cette interface permet aux implémentations MPI d'utiliser le réseaux BXI. Le personnel est composée d'ingénieurs, de doctorant et de stagiaire. /* TODO : vérifié */ La culture informatique de l'équipe est l'utilisation des cluster HPC, la programmation système et l'utilisation d'un système de sécurité PKI pour accédé aux ordinateurs et aux services interne. Atos nous a donné accès à une machine avec deux processeur Intel® Sapphire Rapids. Pour accédé à cette machine ils nous on fourni un accès ssh et un compte VPN. J'ai pu avoir tous les droits d'accès sur cette machine pour changé le noyaux du système.

1.5 Presentation du plan

Dans ce mémoire je vais commencé par vous présenté le context dans le quelle le stage ce place. Je continuerai par les objectifs à long term et les objectifs du stage. Je vous présenterai le nouveau mécanisme d'interruption et ce que j'ai fait avec. Je présenterai l'intégration de ce mécanisme dans la bibliothèque de communication NewMadeleine et les tests que l'on à effectué. Pour finir je ferai un bilan du stage et je parlerai de la suite.

2 Context

2.1 HPC

Le stage c'est déroulé dans le context du Calcule Haute Performance (HPC, High-Performance Computing en anglais). Le HPC utilise des supercalculateur pour la simulation numérique et le pré-apprentissage d'intelligences artificielles. Ces simulation simule la dynamique des fluides, la résistance structurelle, les interaction moléculaire, les flux d'aire...

Elles couvrent différant 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

1. Lustre est un système de fichier parallèle et distribué. Il est utiliser pour l'I/O est ne fait pas partie de ce stage.

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 calcule, la distribution de la mémoire, la communication entre les différant 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. 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 stendardisation des accés... 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'on ne peut pas utiliser d'appel 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.

Pour transmettre des événement à l'utilisateur les périphériques utilise

généralement les interruptions, qui passe par le système donc on ne les utilise pas 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 (j'en parlerai 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 dédier qui fait de l'attente active la réactivité est excellente hormis quand il y à plus de thread que d'unité de calcule est que le thread n'est pas ordonnancer.

Quand on entrelaces le calcule est les scrutations la réactivité est moins bonne car il faut attendre que le calcule 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 chois dois être fait entre perdre de la capacité de calcule ou perdre en réactivité. En plus on perd un peut 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 dédier 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 protocol Portal4. L'utilisateur utilise donc le protocol 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 différant 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` est donc totalement synchrone ce qui bloque le code. 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 de des fonctions `MPI_Wait` et `MPI_Test` ou être faire avant est donc l'appel aux fonctions s'occupe juste de la terminaison. L'envoi des messages peut donc être asynchrone.

2.5.2 Les communications collective

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

— un processus vers plusieurs (One-to-All) par example un broadcast d'un message

— de plusieurs processus vers un seul (All-to-One) par example une reduction (e.g. un processus reçois la somme des valeurs des autre processus)
— de plusieurs processus vers plusieurs (All-to-All) par example 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 asynchrone

Pour faire progressé les communications de façon asynchrone 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 à plus de calcul à faire on repasse à une progression synchrone par un appel à `MPI_Wait` sauf si la communication est déjà fini.
— utilisé un thread dédier 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 évité d'appelé trop souvent `MPI_Test` car ça crée de la contention
— utilisation des threads de façon opportunist c'est à dire quand un des thread à fini sont calcul il vas faire progressé les communications. C'est ce qui est faire par Pioman dans NewMadeleine.

2.6 NewMadeleine

NewMadeleine est une bibliothèque de communications qui historiquement à été développé pour le RPC (Remote procedure call). Elle a toujours cette capacité mais elle a évolué et implémente une interface MPI. On peut donc la considéré comme une implémentation du standard MPI avec des fonctionnalités supplémentaire. Elle est développé au Centre Inria de l'université de Bordeaux. Elle est basé sur un système de progression événementielle qui lui permet d'être asynchrone. Elle est composé en modules ce qui lui permet de chargé dynamiquement des stratégies d'optimisation sur le paquets. Les stratégies sont l'agrégation de paquets, l'utilisation de priorités, l'utilisation d'arbre de décision ?, split_balance ?. Elle possède également un système de drivers pour supporté différant type 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 partagé), socket, self).

2.7 Travaux antérieur

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 diminué la latence du au fait de passer par un driver noyaux.

— Il parle dans les perspective de directement traitement des interruptions depuis l'espace utilisateur ce qui mène à mon 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 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 permettrai de ne plus avoir ce problèmes car seul le thread concerné par une zone mémoire serai prévenu.

TODO :

3.2 Ne plus faire de Polling

Dans la plupart des autre domain les périphériques envoi une interruption à l'application pour avertir d'un changement. Souvent on passe par les signaux ou un appel système bloquante. L'idée du stage est de ne plus avoir à faire du polling et utiliser les interruptions comme dans les autre domains.

3.3 Objectifs

Il y a plusieurs objects. Les principaux sont :

— La réduction du temps de calcul.

— utiliser des interruptions en espace utilisateur pour replacé 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 dédier à l'attente active.

3.3.1 Objectifs global

Le stage s'inscrit dans un objectif globale qui est de faire progresser les communications entre plusieurs noeuds du réseau BXI sans faire de polling et en utilisant les interruptions en espace utilisateur. Cette objectif permet 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 consomption électrique.

3.3.2 Objectifs du stage

Le premier objectif est de défricher le fonctionnement des interruption en espace utilisateur à partir du manuel Intel® de l'architectures 64 et IA-32 pour les développeurs logiciels, de la présentation [2] [3] du mécanisme de Sohil Mehta [4] et du patch du noyau linux avec ces manuels. Le second objectif est de connaître les propriétés du mécanisme et de mesuré ça performance. Le troisième objectif est d'envisager l'intégration de ces interruption dans le driver de mémoire partagé (shm) de NewMadeleine. L'objectif est de testé dans un premier temps le fonctionnement avec des communications entre processus (IPC [5] en anglais). 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.

2. Qui est une diapositive associer à des discussion sur LWN.net
3. https://lwn.net/Articles/869140/
4. Ingénieur Intel® qui à développé le patch noyau
5. Inter Process Communication

4 Exploration de uintr

4.1 Prérequis est accès

- difficulté d'accès au processeur...
- J'ai eu accès au processeur 2 mois 1/2 après le début du stage
- J'ai donc eu accès à une machine avec 2 CPU SPR... VPN... détails Saphir Rapid et VPN
- il faut une version patché du noyaux linux pour utilisé c'est nouvelle interruption Ce patch n'est pas disponible en mainline linux. J'ai donc télécharger le patch fourni par Intel®, je l'ai compiler puis je l'ai installer sur la machine
- il faut compiler le noyau patché et activé les uintr dans le menu de configuration (il est aussi possible d'activé la posibilité que le kernel resoive l'interruption quand le processus est endormie en tache interruptible.)
- Pour utiliser les nouvelle instructions il faut une version récente de GCC... (version). Pas disponible avec LLVM-Clang
- pour compilé un programme il faut précié le flag de compilation `-muintr` pour les fichiers qui déclare un handler d'uintr.

4.2 Fonctionnement des uintr

4.2.1 Les interruptions

Pour commencer nous allons voire comment les interruption matérielle fonctionnent, elle sont nommé IRQ pour Interrupt ReQuest. Nous allons utiliseront le terme interruption ordinaire ou IRQ. Les interruption ordinaire existe depuis long temps est serve notamment à remonté des exceptions du processeur au noyau. Nous allons nous concentrer sur l'envois d'interruption entre unité de calcule, donc entre processus. Pour fonctionné les CPU on 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 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 `INTO`, `BOUND` et `INT n`.
3. Des pins du processeur dédier 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 example d'envoi d'IRQ entre 2 processus en cours d'execution. Tous d'abort 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 fau 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 calcule ou ce trouve le récepteur. Pour cela il peut utiliser par example 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...).

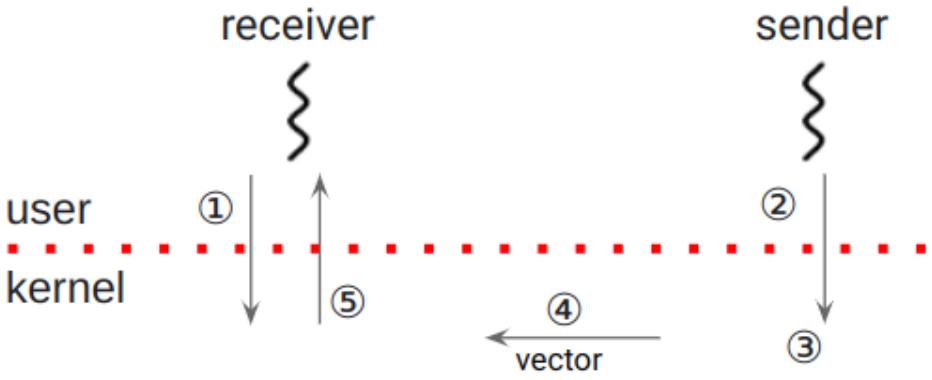

FIGURE 1 – L'envois d'une interruption

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 ce 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 ce trouve ici2.

Comment on la vue ce mécanisme fonction totalement dans le noyau du système. Dans notre example il faut au minimum deux changement de contexte (context switch) pour le récepteur et l'émetteur et même peut être plus si le récepteur dois 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 `INT n`, `INT3`, `INTO`, `BOUND`, `INT80 n` 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 defaut 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_WAITI s'ajoute au flague préciédent et précide 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 fonctionnemt des inrettruption "ordinaire" plus l'appel du handler en espace utilisateur.

4.2.4 Example

— example avec schéma

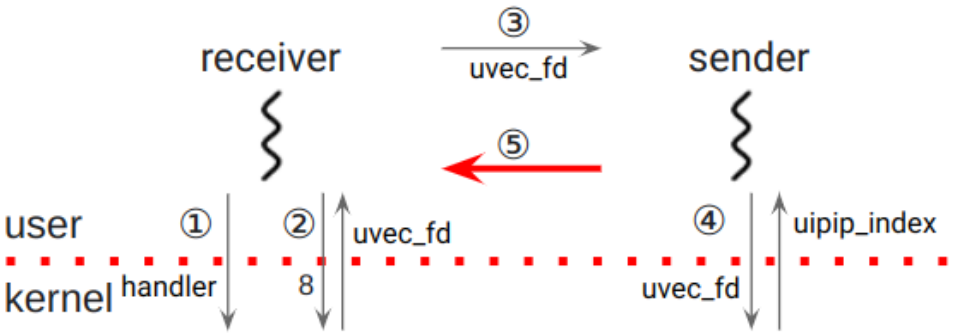

FIGURE 3 – Phase d'initialisation des uintr

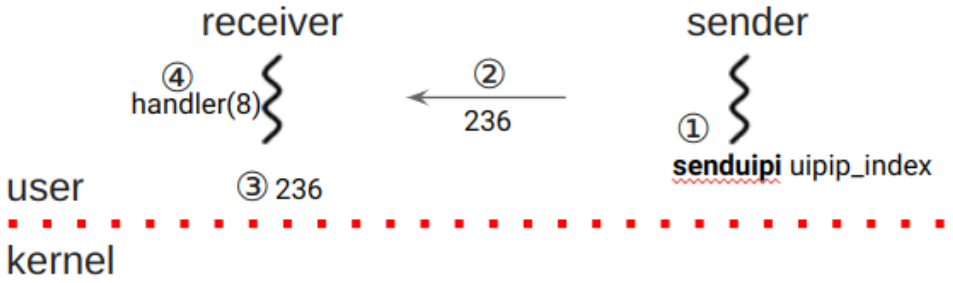

FIGURE 4 – L'envois d'une uintr

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 au operations sur les chaine de caractére de la libc memcpy, memmove, memset et memcmp qui utiliser des registre, 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 thread par pu donc j'avais de mauvais résultats.

— J'ai fait différents bind (on vois que c'est similaire sauf quand on passe entre 2 NUMA)

— si on monte la frequence 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 recepteur que des données sont disponible (le recepteur à toujours un paquet de reception posté)
    - au moment ou une progression est fait du coté du recepteur, pour indiquer une reception a l'éméteur
    - l'éméteur reçois une iterruption est détermine si l'émision 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