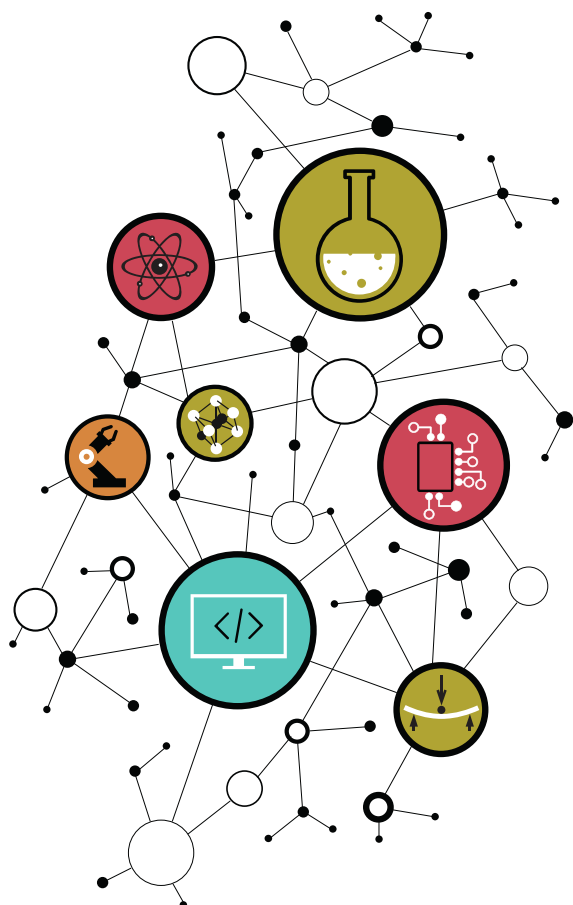


## PARTIE 8

# EXCEPTIONS MATÉRIELLES ET SIGNAUX UNIX



### I. Exception vs Signal

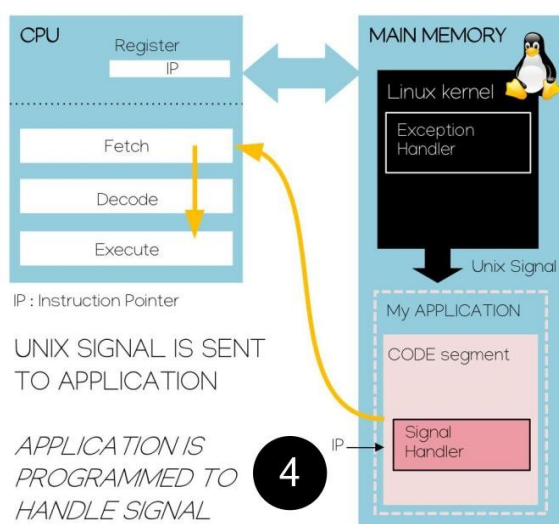
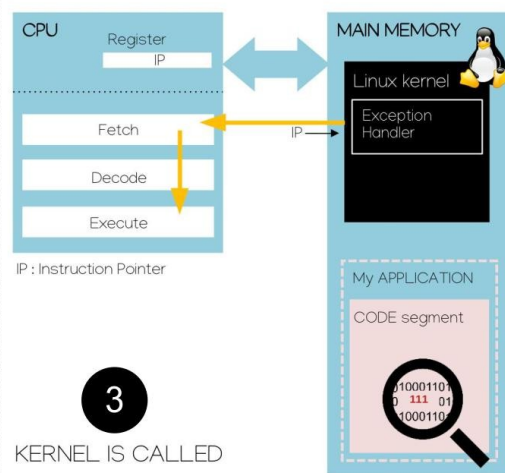
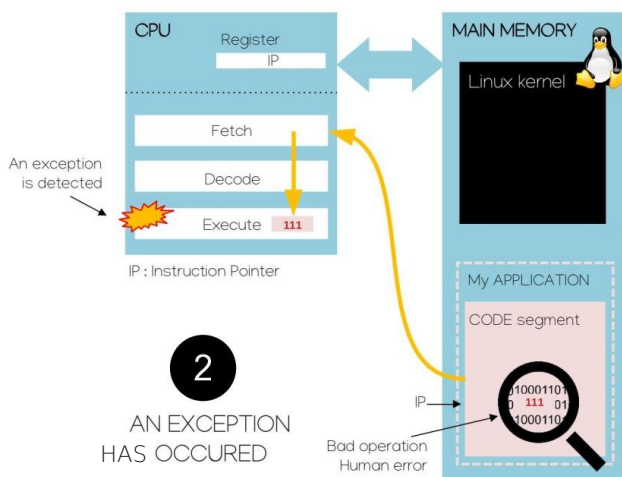
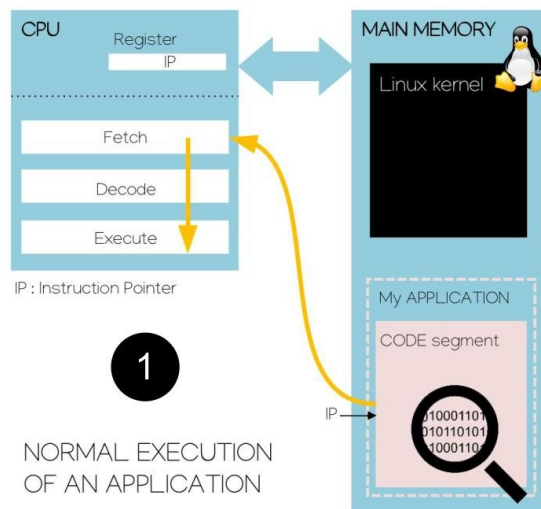
Une **exception matérielle** est un événement matériel synchrone généré par le CPU voire la MMU. Nous parlons d'événement synchrone au regard du fonctionnement d'un CPU dont les traitements restent synchronisés sur une référence d'horloge et non au regard de la probabilité d'occurrence. Une exception peut interrompre l'exécution d'une application à tout moment. Ces événements sont relevés par le CPU lorsque celui-ci détecte une condition prédéfinie et documentée non conventionnelle durant l'exécution d'une instruction (violation de privilège, division flottante par zéro, accès illégal en mémoire, etc). Contrairement aux interruptions (générées par les périphériques) provoquées par des causes externes au programme (asynchrone), les exceptions sont provoquées par des causes internes au programme (synchrone). Toute exception est une opération connue ne devant généralement en aucun cas arriver. Elles sont le plus souvent le fruit d'une erreur de programmation. Détaillons la séquence graphique présentée sur la page suivante :

1. L'application s'exécute normalement sur le CPU courant. Le noyau est au repos en attente d'un événement requérant son travail ;
2. Le programme en cours d'exécution implémente une opération binaire erronée. Durant son exécution par le CPU, l'instruction génère une exception. Le pointeur d'instruction IP est alors redirigé (par lecture et exécution d'un tableau de vecteurs) vers une fonction noyau dédiée au traitement des exceptions. Le noyau du système s'exécute alors sur le CPU courant ;
3. Le CPU vient donc de stopper l'exécution de l'application en cours et d'appeler une procédure enfouie du système. Une sauvegarde du contexte CPU (registres de travail internes) est également réalisée et pourra être accessible depuis l'application en espace utilisateur.

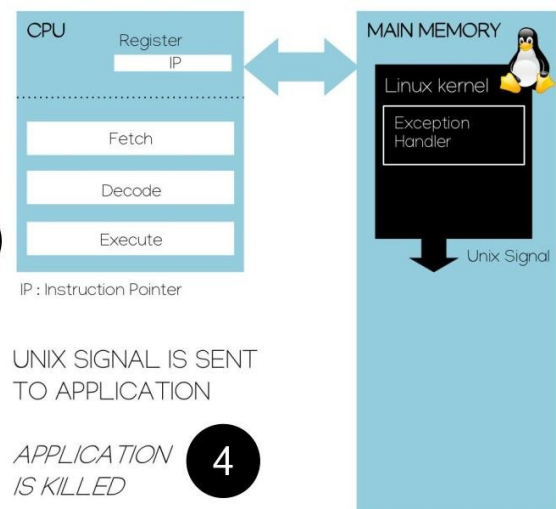
La procédure de gestion des exceptions est implémentée par la fonction `do_page_fault` dans le cas de Linux (présente dans le fichier `/arch/<cpu_architecture>/mm/fault.c` du système de fichier du *kernel* Linux<sup>15</sup>). La fonction de gestion des exceptions (ou *exception handler*) est chargée de relever le type de défaut et de générer, si l'exception le permet, un **signal logiciel** Unix à destination du processus ayant généré le défaut.

4. Le noyau du système vient d'envoyer un signal logiciel au processus. Deux cas de figure :
  - (schéma de droite) Si le processus ne traite pas le signal Unix, il est alors tué par le noyau du système qui assure la libération de toutes les ressources mémoire associées. Aucune autre application n'est impactée. Ceci assure un cloisonnement des défauts et la stabilité du système.
  - (schéma de gauche) Si le processus traite le signal Unix (le développeur a rédigé dans l'application un code spécifique au traitement de ces signaux Unix), l'application peut alors tenter d'acquiescer le défaut (restaurer un contexte CPU viable) ou tenter une solution de contournement (redirection vers un autre code de l'application viable, redémarrage ou mode dégradé de l'application, etc). Si cela est possible, un message d'erreur ou un fichier de log pourra être sauvé voire envoyé aux équipes de développement.

<sup>15</sup> <https://www.kernel.org/>



or



## II. Lecture seule

Placez-vous dans le répertoire `disco/except/`. Compiler le fichier `except_readonly.c` jusqu'à l'édition des liens incluse et exécutez le programme.

```
gcc except_readonly.c -o except_readonly
```

Interprétez et expliquez l'exception matérielle ainsi que le défaut logiciel à la racine de cette exception. Proposez un schéma commenté.

### III. Pointeur nul

Compilez le fichier `except_null_pointer.c` jusqu'à l'édition des liens incluse et exécutez le programme.

```
gcc except_null_pointer.c -o except_null_pointer
```

Interprétez et expliquez l'exception matérielle ainsi que le défaut logiciel à la racine de cette exception. Proposez un schéma commenté.

Le 19 juillet 2024 une panne informatique d'envergure mondiale se produit. CrowdStrike, une société spécialisée en cybersécurité, publie une mise à jour d'un de leurs fichiers (comme ils le font plusieurs fois par jour) sur les machines Windows équipées de leur anti-virus.

Cette fois-ci, le fichier contenait une erreur. Même si les sources divergent au sujet de l'origine réelle du bug, toutes tendent vers le même constat : dans un des fichiers de CrowdStrike rédigés en C++, un pointeur `null` a été déréférencé.

Cette exception matérielle non traitée provoquait systématiquement le *kill* du processus au démarrage des ordinateurs, ce qui menait au fameux **Blue Screen Of Death** (BSOD) puis forçait un redémarrage en boucle du système.

Au final, près de 8.5 millions de machines Windows ont été touchées à travers le monde, annulant notamment plusieurs milliers de vols commerciaux sur plusieurs jours. Le préjudice pourrait atteindre près de 10 milliards de \$<sup>16</sup>, les plus touchés étant dans l'ordre les industries de santé, le secteur bancaire et les compagnies aériennes<sup>17</sup>.

<sup>16</sup> <https://www.reuters.com/technology/insurers-face-business-interruption-claims-after-global-tech-outage-2024-07-19/>

<sup>17</sup> <https://edition.cnn.com/2024/07/24/tech/crowdstrike-outage-cost-cause/index.html>

### IV. Signal Unix

Une fois l'**exception matérielle** levée (*Null Pointer exception* dans l'exercice précédent), le noyau du système d'exploitation envoie un **Unix signal** (ici SIGSEGV, soit *Segmentation Violation*) au processus en cause. Le processus doit impérativement traiter ce signal, sans quoi il sera *kill*.

Compilez le fichier `signal_handler.c` jusqu'à l'édition des liens incluse et exécutez le programme.

```
gcc signal_handler.c -o signal_handler
```

Interpréter et expliquer l'exception matérielle ainsi que le défaut logiciel à la racine de cette exception.

Analysez le script assembleur du programme et précisez l'instruction ainsi que le registre à la source de l'exception.

```
gcc -S -fno-asynchronous-unwind-tables -fno-pie -fno-stack-protector -fcf-protection=none -Wall signal_handler.c
```

La séquence qui suit propose de remplacer le contenu du registre avec une valeur acceptable par le système pour la bonne exécution du processus.

Dé-commentez la section de code se trouvant dans la fonction `main()`. Ces instructions configurent la gestion des signaux système reçus par l'application. Compilez le fichier C jusqu'à l'édition des liens incluse, exécutez-le et analysez le fonctionnement du programme.

Dé-commentez la ligne de code dans la fonction `signal_handler()`. Celle ci propose une solution afin de corriger la source de l'exception (dans le `main`). Compilez, exécutez et analysez le résultat.

Voilà, vous venez de générer volontairement une exception (défaut de segment), de capter le signal Unix envoyé par le système, d'acquiescer l'erreur par remplacement du contenu d'un registre (RAX) et de redonner une chance à l'application de s'exécuter sans défaut.



