Présentation du projet de systèmes d'exploitation

Ryan LAHFA, Théophane VALLAEYS, Julien MARQUET

Introduction au projet

Nature du système et fonctionnalités

Un système d'exploitation avec une approche micro-noyau, écrit dans Zig[1] qui cible les plateformes UEFI[2] pour x86_64.

Fonctionnalités :

- Appels systèmes rapides avec System Call Extensions ;

1. Un langage expérimental qui se veut être un remplacement du C pour le bas niveau.
2. Testé sur OVMF principalement.

Nature du système et fonctionnalités

Un système d'exploitation avec une approche micro-noyau, écrit dans Zig[1] qui cible les plateformes UEFI[2] pour x86_64.

Fonctionnalités :

- Appels systèmes rapides avec System Call Extensions ;
- Mémoire virtuelle et PML4 gérée ;

1. Un langage expérimental qui se veut être un remplacement du C pour le bas niveau.
2. Testé sur OVMF principalement.

Nature du système et fonctionnalités

Un système d'exploitation avec une approche micro-noyau, écrit dans Zig[1] qui cible les plateformes UEFI[2] pour x86_64.

Fonctionnalités :

- Appels systèmes rapides avec System Call Extensions ;
- Mémoire virtuelle et PML4 gérée ;
- Gestion vidéo avec le framebuffer linéaire UEFI, polices de caractères PSF2 intégrée, dessin de texte, « TTY » simple ;

1. Un langage expérimental qui se veut être un remplacement du C pour le bas niveau.
2. Testé sur OVMF principalement.

Nature du système et fonctionnalités

Un système d'exploitation avec une approche micro-noyau, écrit dans Zig[1] qui cible les plateformes UEFI[2] pour x86_64.

Fonctionnalités :

- Appels systèmes rapides avec System Call Extensions ;
- Mémoire virtuelle et PML4 gérée ;
- Gestion vidéo avec le framebuffer linéaire UEFI, polices de caractères PSF2 intégrée, dessin de texte, « TTY » simple ;
- Gestion des tâches utilisateur et noyau avec le scheduler de L4

1. Un langage expérimental qui se veut être un remplacement du C pour le bas niveau.
2. Testé sur OVMF principalement.

Nature du système et fonctionnalités

Un système d'exploitation avec une approche micro-noyau, écrit dans Zig[1] qui cible les plateformes UEFI[2] pour x86_64.

Fonctionnalités :

- Appels systèmes rapides avec System Call Extensions ;
- Mémoire virtuelle et PML4 gérée ;
- Gestion vidéo avec le framebuffer linéaire UEFI, polices de caractères PSF2 intégrée, dessin de texte, « TTY » simple ;
- Gestion des tâches utilisateur et noyau avec le scheduler de L4
- Debuggage sur console série, debuggage avec GDB et tous les symboles

1. Un langage expérimental qui se veut être un remplacement du C pour le bas niveau.
2. Testé sur OVMF principalement.

Plan de présentation du système

Plan de cette soutenance :

- x86_64, mémoire virtuelle, protection (Julien)

Plan de présentation du système

Plan de cette soutenance :

- x86_64, mémoire virtuelle, protection (Julien)
- Protocoles UEFI, framebuffer VGA, appels systèmes rapides (Ryan)

Plan de présentation du système

Plan de cette soutenance :

- x86_64, mémoire virtuelle, protection (Julien)
- Protocoles UEFI, framebuffer VGA, appels systèmes rapides (Ryan)
- Scheduler, tâches, préemptions par interruptions (Théophane)

Un mot sur Zig

Difficultés générales

Mémoire virtuelle en x86_64

Mémoire virtuelle en x86_64

- Deux espaces d'adressage : « physique » et « linéaire »[3]

3. En mode 64 bits

Mémoire virtuelle en x86_64

- Deux espaces d'adressage : « physique » et « linéaire »[3]
- Le système de mémoire virtuelle permet d'associer à chaque adresse *virtuelle* une adresse *physique*

3. En mode 64 bits

Mémoire virtuelle en x86_64

- Deux espaces d'adressage : « physique » et « linéaire »[3]
- Le système de mémoire virtuelle permet d'associer à chaque adresse *virtuelle* une adresse *physique*
- La granularité de ce système est la « page » : 4096 octets

3. En mode 64 bits

Mémoire virtuelle en x86_64

- Deux espaces d'adressage : « physique » et « linéaire »[3]
- Le système de mémoire virtuelle permet d'associer à chaque adresse *virtuelle* une adresse *physique*
- La granularité de ce système est la « page » : 4096 octets
- Traduction de linéaire vers physique en décomposant les adresses linéaires en morçeaux :

3. En mode 64 bits

Mémoire virtuelle en x86_64

- Deux espaces d'adressage : « physique » et « linéaire »[3]
- Le système de mémoire virtuelle permet d'associer à chaque adresse *virtuelle* une adresse *physique*
- La granularité de ce système est la « page » : 4096 octets
- Traduction de linéaire vers physique en décomposant les adresses linéaires en morçeaux :
- Les 12 derniers bits : adresse dans une page

3. En mode 64 bits

Mémoire virtuelle en x86_64

- Deux espaces d'adressage : « physique » et « linéaire »[3]
- Le système de mémoire virtuelle permet d'associer à chaque adresse *virtuelle* une adresse *physique*
- La granularité de ce système est la « page » : 4096 octets
- Traduction de linéaire vers physique en décomposant les adresses linéaires en morçeaux :
- Les 12 derniers bits : adresse dans une page
- 4*9 bits pour les 4 niveaux de « page tables »[4]

3. En mode 64 bits

4. Vocabulaire non officiel

Mémoire virtuelle en x86_64

- Deux espaces d'adressage : « physique » et « linéaire »[3]
- Le système de mémoire virtuelle permet d'associer à chaque adresse *virtuelle* une adresse *physique*
- La granularité de ce système est la « page » : 4096 octets
- Traduction de linéaire vers physique en décomposant les adresses linéaires en morçeaux :
- Les 12 derniers bits : adresse dans une page
- 4*9 bits pour les 4 niveaux de « page tables »[4]
- Les bits de poids forts sont remplis par extension de signe[5]

---

3. En mode 64 bits

4. Vocabulaire non officiel

5. Et une adresse non valide provoque une erreur, plusieurs heures de lecture de documentation ont été nécessaires pour obtenir cette information.

Organisation des structures de contrôle

- Les entrées des tables de pages sont chacune stockées sur une page

Organisation des structures de contrôle

- Les entrées des tables de pages sont chacune stockées sur une page
- Les tables les plus basses référencent les adresses des pages[6]

6. Adresses alignées sur une page, *i.e.* 4Ko

Organisation des structures de contrôle

- Les entrées des tables de pages sont chacune stockées sur une page
- Les tables les plus basses référencent les adresses des pages[6]
- Les tables de niveau supérieur référencent les pages inférieures[7]

6. Adresses alignées sur une page, *i.e.* 4Ko

7. Adresses aussi alignées sur une page

Organisation des structures de contrôle

- Les entrées des tables de pages sont chacune stockées sur une page
- Les tables les plus basses référencent les adresses des pages[6]
- Les tables de niveau supérieur référencent les pages inférieures[7]
- Chaque entrée comporte des fanions indiquant les droits sur la page, principalement :

6. Adresses alignées sur une page, *i.e.* 4Ko

7. Adresses aussi alignées sur une page

Organisation des structures de contrôle

- Les entrées des tables de pages sont chacune stockées sur une page
- Les tables les plus basses référencent les adresses des pages[6]
- Les tables de niveau supérieur référencent les pages inférieures[7]
- Chaque entrée comporte des fanions indiquant les droits sur la page, principalement :
- P : Indique que la page est bien présente

6. Adresses alignées sur une page, *i.e.* 4Ko

7. Adresses aussi alignées sur une page

Organisation des structures de contrôle

- Les entrées des tables de pages sont chacune stockées sur une page
- Les tables les plus basses référencent les adresses des pages[6]
- Les tables de niveau supérieur référencent les pages inférieures[7]
- Chaque entrée comporte des fanions indiquant les droits sur la page, principalement :
- P : Indique que la page est bien présente
- US : Indique que la page est accessible en mode utilisateur (_ring 3)

6. Adresses alignées sur une page, *i.e.* 4Ko

7. Adresses aussi alignées sur une page

Organisation des structures de contrôle

- Les entrées des tables de pages sont chacune stockées sur une page
- Les tables les plus basses référencent les adresses des pages[6]
- Les tables de niveau supérieur référencent les pages inférieures[7]
- Chaque entrée comporte des fanions indiquant les droits sur la page, principalement :
- P : Indique que la page est bien présente
- US : Indique que la page est accessible en mode utilisateur (_ring 3)
- RW : Indique qu'il est possible d'écrire sur la page[8]

6. Adresses alignées sur une page, *i.e.* 4Ko

7. Adresses aussi alignées sur une page

8. « If 0, writes may not be allowed » ; possiblement ambigu

Organisation des structures de contrôle

- Les entrées des tables de pages sont chacune stockées sur une page
- Les tables les plus basses référencent les adresses des pages[6]
- Les tables de niveau supérieur référencent les pages inférieures[7]
- Chaque entrée comporte des fanions indiquant les droits sur la page, principalement :
- P : Indique que la page est bien présente
- US : Indique que la page est accessible en mode utilisateur (_ring 3)
- RW : Indique qu'il est possible d'écrire sur la page[8]
- XD : Pour interdire l'exécution du contenu de la page

6. Adresses alignées sur une page, *i.e.* 4Ko

7. Adresses aussi alignées sur une page

8. « If 0, writes may not be allowed » ; possiblement ambigu

Organisation des structures de contrôle

- Les entrées des tables de pages sont chacune stockées sur une page
- Les tables les plus basses référencent les adresses des pages[6]
- Les tables de niveau supérieur référencent les pages inférieures[7]
- Chaque entrée comporte des fanions indiquant les droits sur la page, principalement :
- P : Indique que la page est bien présente
- US : Indique que la page est accessible en mode utilisateur (_ring 3)
- RW : Indique qu'il est possible d'écrire sur la page[8]
- XD : Pour interdire l'exécution du contenu de la page
- On peut aussi avoir des « hugepages »[9] de 2Mo ou 1Go

6. Adresses alignées sur une page, *i.e.* 4Ko
7. Adresses aussi alignées sur une page
8. « If 0, writes may not be allowed » ; possiblement ambigu
9. Vocabulaire non officiel

Organisation des structures de contrôle

- Les entrées des tables de pages sont chacune stockées sur une page
- Les tables les plus basses référencent les adresses des pages[6]
- Les tables de niveau supérieur référencent les pages inférieures[7]
- Chaque entrée comporte des fanions indiquant les droits sur la page, principalement :
- P : Indique que la page est bien présente
- US : Indique que la page est accessible en mode utilisateur (_ring 3)
- RW : Indique qu'il est possible d'écrire sur la page[8]
- XD : Pour interdire l'exécution du contenu de la page
- On peut aussi avoir des « hugepages »[9] de 2Mo ou 1Go
- La page racine[10] est référencée par le registre CR3.[11]

6. Adresses alignées sur une page, *i.e.* 4Ko
7. Adresses aussi alignées sur une page
8. « If 0, writes may not be allowed » ; possiblement ambigu
9. Vocabulaire non officiel
10. PML4 chez nous
11. À côté de quelques fanions supplémentaires que nous n'utilisons pas.

Application de la mémoire virtuelle

- On a une grande quantité de mémoire virtuelle disponible

Application de la mémoire virtuelle

- On a une grande quantité de mémoire virtuelle disponible
- Il est d'usage de séparer l'espace d'adressage en deux moitiés :

Application de la mémoire virtuelle

- On a une grande quantité de mémoire virtuelle disponible
- Il est d'usage de séparer l'espace d'adressage en deux moitiés :
- Moitié haute pour le noyau

Application de la mémoire virtuelle

- On a une grande quantité de mémoire virtuelle disponible
- Il est d'usage de séparer l'espace d'adressage en deux moitiés :
- Moitié haute pour le noyau
- Moitié basse pour l'utilisateur

Application de la mémoire virtuelle

- On a une grande quantité de mémoire virtuelle disponible
- Il est d'usage de séparer l'espace d'adressage en deux moitiés :
- Moitié haute pour le noyau
- Moitié basse pour l'utilisateur
- Grâce à la restructuration de la mémoire, on peut faire croire à chaque tâche qu'elle vit seule sur la machine

Application de la mémoire virtuelle

- On a une grande quantité de mémoire virtuelle disponible
- Il est d'usage de séparer l'espace d'adressage en deux moitiés :
- Moitié haute pour le noyau
- Moitié basse pour l'utilisateur
- Grâce à la restructuration de la mémoire, on peut faire croire à chaque tâche qu'elle vit seule sur la machine
- Ceci permet d'avoir des exécutables dont le code *dépend* de la position en mémoire[12]

12. Position des données, des instructions, ...

Application de la mémoire virtuelle

- On a une grande quantité de mémoire virtuelle disponible
- Il est d'usage de séparer l'espace d'adressage en deux moitiés :
- Moitié haute pour le noyau
- Moitié basse pour l'utilisateur
- Grâce à la restructuration de la mémoire, on peut faire croire à chaque tâche qu'elle vit seule sur la machine
- Ceci permet d'avoir des exécutables dont le code *dépend* de la position en mémoire[12]
- Permet aussi de simplifier l'agencement de la mémoire pour l'utilisateur : il est beaucoup plus simple de faire grandir la pile par exemple

12. Position des données, des instructions, ...

Application de la mémoire virtuelle

- On a une grande quantité de mémoire virtuelle disponible
- Il est d'usage de séparer l'espace d'adressage en deux moitiés :
- Moitié haute pour le noyau
- Moitié basse pour l'utilisateur
- Grâce à la restructuration de la mémoire, on peut faire croire à chaque tâche qu'elle vit seule sur la machine
- Ceci permet d'avoir des exécutables dont le code *dépend* de la position en mémoire[12]
- Permet aussi de simplifier l'agencement de la mémoire pour l'utilisateur : il est beaucoup plus simple de faire grandir la pile par exemple
- La protection de la mémoire permet d'emêcher les utilisateurs de casser le noyau

12. Position des données, des instructions, ...

Application de la mémoire virtuelle

- On a une grande quantité de mémoire virtuelle disponible
- Il est d'usage de séparer l'espace d'adressage en deux moitiés :
- Moitié haute pour le noyau
- Moitié basse pour l'utilisateur
- Grâce à la restructuration de la mémoire, on peut faire croire à chaque tâche qu'elle vit seule sur la machine
- Ceci permet d'avoir des exécutables dont le code *dépend* de la position en mémoire[12]
- Permet aussi de simplifier l'agencement de la mémoire pour l'utilisateur : il est beaucoup plus simple de faire grandir la pile par exemple
- La protection de la mémoire permet d'emêcher les utilisateurs de casser le noyau
- Elle permet aussi de protéger les utilisateurs les uns des autres

12. Position des données, des instructions, ...

Note sur les TLBs

- TLBs : « Translation Lookaside Buffers »

Note sur les TLBs

- TLBs : « Translation Lookaside Buffers »
- Stockent des traduction $\text{linéaire} \rightarrow \text{physique}$

Note sur les TLBs

- TLBs : « Translation Lookaside Buffers »
- Stockent des traduction $\text{linéaire} \rightarrow \text{physique}$
- Permettent d'éviter de parcourir la hiérarchie de la mémoire virtuelle à chaque accès mémoire

Note sur les TLBs

- TLBs : « Translation Lookaside Buffers »
- Stockent des traduction $\text{linéaire} \rightarrow \text{physique}$
- Permettent d'éviter de parcourir la hiérarchie de la mémoire virtuelle à chaque accès mémoire
- Mais doivent êtres gérés à la main : il faut savoir quand les vider

Note sur les TLBs

- TLBs : « Translation Lookaside Buffers »
- Stockent des traduction $\text{linéaire} \rightarrow \text{physique}$
- Permettent d'éviter de parcourir la hiérarchie de la mémoire virtuelle à chaque accès mémoire
- Mais doivent êtres gérés à la main : il faut savoir quand les vider
- `invlpg` permet de vider les traductions associées à une page en particulier

Note sur les TLBs

- TLBs : « Translation Lookaside Buffers »
- Stockent des traduction $\text{linéaire} \rightarrow \text{physique}$
- Permettent d'éviter de parcourir la hiérarchie de la mémoire virtuelle à chaque accès mémoire
- Mais doivent êtres gérés à la main : il faut savoir quand les vider
- `invlpg` permet de vider les traductions associées à une page en particulier
- On peut aussi vider tous les TLBs en rechargant CR3

Note sur les TLBs

- TLBs : « Translation Lookaside Buffers »
- Stockent des traduction $\text{linéaire} \rightarrow \text{physique}$
- Permettent d'éviter de parcourir la hiérarchie de la mémoire virtuelle à chaque accès mémoire
- Mais doivent êtres gérés à la main : il faut savoir quand les vider
- `invlpg` permet de vider les traductions associées à une page en particulier
- On peut aussi vider tous les TLBs en rechargant CR3
- Ceci doit être fait en paritculier lors de la modification de la structure de la mémoire et lors d'un changement de permissions

Spécificités UEFI

Protocoles UEFI

Obtenir un framebuffer VGA

Obtenir un accès au disque

Reconstruire l’IDT, la GDT

Appels systèmes

Interruptions

Gestion des interruptions

- toutes les variables d'état mises sur la pile, map sur un Context
- récupère la fonction dans `handlers` qui renvoie un pointeur sur le prochain contexte
- Fin d'interruption, reprisse du processus (voir passage en user mode)

PIT (Programmable Interval Timer)

- Oscillateur à une fréquence de ~ 1.193182 MHz
    - Calcul d'un interval de temps approximant la fréquence voulue
- IRQ régulier, vers le scheduler
- Permet de mesurer le temps en nanosecoudes
    - CMOS pour la seconde et au-delà

Scheduler

Tâches

- Une tâche associée à chaque processus kernel ou utilisateur
  - Y compris le boot dès l'initialisation de la plateforme
- Nombre fixe de PID, stockés dans une bitmap
- Mémoire allouée une fois seulement :
  - 2 pointeurs pour le scheduler
  - pointeurs vers les piles
  - variables d'état (`kernel` mode, `pid`, `timeout`, `scheduled`…)

Round-Robin scheduling

- Pointeur vers chaque tâche la première fois qu'elle est scheduled
- Chaque tic du PIT : changement de tâche (préemption)
- Priorités : similaire au microkernel L4 (0 à 255)
- Tache vide

Status d'un processus

- États similaire à linux : `runnable`, `stopped`, `sleep`, `sleepNoInterrupt`, `zombie`
- Changement d'état rapide : modifier un octet
- Le scheduler rejette les processus non exécutables ensuite
  - Permet de gagner du temps
  - Contrepartie : pas en $O(1)$

Timeouts

- 3 listes doublement chaînées
- `e_soon`, `e_late`
- timeouts proches : triés (insertion $O(n)$, accès $O(1)$)
- timeouts tardifs et lointaints : désordonné (accès $O(n)$, insertion $O(1)$)
- temps écoulé calculé avec le PIT
- Retire immédiatement du scheduling (besoin des pointeurs)

Affichage à l’écran

Côté kernel

- Au boot, utilisation de l'interface UEFI pour trouver le mode d'affichage correct
    - Récupération du pointeur vers le framebuffer et des informations
- Délégation au driverspace
    - Librairie partagée pour le kernel et les drivers
    - Juste besoin de map une zone de la mémoire vers le buffer

Librairie

- Graphics : primitives de base
  - Conversion code couleur / pixel (`packed struct` sur 32 bits)
  - Gestions de couleurs
  - Dessinner une zone
  - Copier un tableau de pixel vers une portion d'écran (en redimensionnant)
  - Positionner et dessiner du texte (police simple) ; permet de se passer de UEFI
- TTY : afficage textuel
  - Affiche le texte à la suite
  - Retours à la ligne, scroll automatique
  - Gestion des caractères spéciaux (retour arrière, retour chariot, nouvelle ligne, tabulation)
  - Alignement du texte
  - "Suivre" la position du texte pour les steps
- Démo : écran de boot