Correction partiel système:

Date: 2018-02-15, promo 2020.

Notes d'un élève, cours de Gabriel Laskar

Disclaimer

Ce sont des notes de cours imparfaites qui n'ont absolument rien d'officiel. Le professeur n'a eu strictement aucune implication dans la création de ce fichier. Il n'engage personne et en particulier ni le professeur ni les étudiants qui l'ont écrit.

C'est juste des élèves quelconques qui font de leur mieux pour suivre en cours et vous partager leurs notes pour vous aider dans vos révisions.

Sur ce, bon travail :).

Quand est-ce qu'un x86 cpu trigger un pagefault ?

- si déréférencement d'une zone non mappée
- si pas les bons droits (pas qu'est ce qu'il se passe en cas de pagefault)

Quels sont les différents états d'un process ?

- new
- ready : un proccess ne tourne pas mais est prêt à tourner à tout moment
- running : process actif sur le cpu (choisi par scheduler)
- dead : après exit()
- wait : quandd quelqu'un d'autre wait dessus pour récupérer la valuer de retour
- waitng/blocked : transition ac syscalls bloquants (read, write, wait, sleep, etc..) -> puis revient en ready

algo de scheduling : implémente de la préemption algo préemptif: on peut le mettre en pause (PAS algo de scheduling)

Round Robin

A file de process ready, et prend le premier quand le cpu est dispo puis qd on le remet dans ready, on le remet à la fin + préemption ac quaton de temps. => expliquer TOUTES les transistions : si un process qui fait un syscall bloquant, on a rien en running, et on doit en choisir un nouveau. -> time slice pas fixe au final, puisque l'un récupère le bout de quanton du précédent

Page table, dir, offset:

-> Intel man page 2064

A quoi sert la pagination pour un OS ?

Fournir de l'isolation au process, avoir pls taches qui peuvent partager un espace d'adressage (+ de pouvoir mapper à la demande, etc..)

Pagefault handler:

-> fait qqch d'intelligent en cas de pagefault. Agit par exemple quand il n'y a pas de correspondance entre mapping fait par utilisateur et mapping réel sur le mapping rééel. - Paging on demand : allocation de mémoire à la demande: je ne mappe pas tant que la zone allouée n'a pas été touchée (pour le faire au dernier moment et gagner de la place). - Copy On Write - Swapping : qd pression mémoire commence à être trop forte, certaines pages vont être virées (les moins récemment utilisées), puis placées sur un disk, marqué sur pt qu'elle n'est plus présente et qu'elle est swappable : enlever bit present et laisser l'adresse - Pagefault : pas une erreur Exception -> trap : ça arrive, ça passe, on a juste signaler l'erreur (passe à l'instruction d'après) -> fault : moment où on peut réparer (reste sur instruction qui a raté) Kernel renvoie l'erreur sous forme de signal - mapping de fichier : récupérer dans le fichier la zone qui nous intéresse.

Copy On Write : certaine zone marquées comme étant dupliquée (zone physique

pointées par pls espaces d'adressage). Par exemple utilisé quand on fork. -> Fork : dupliquer l'addres space -> cher -> on duplique les méta données, mais on duplique pas la ram on copie tout en RO, et on duplique quand il y a écriture

-> Mapping de fichier: zone de code de la libc par ex. -> écriture sur zone de mémoire que je n'ai jamais touchée : utilisateur = singe on garde une page dans un coin ac que des 0, et on lui file, puis on duplique. ex : je mappe une zone en anonyme, allouée de manière logique, et j'essaie d'y toucher

Quels syscalls pour lancer un nouveau process: fork et execve et *wait*

Correction Assembleur:

Que contient un fichier objet (elf) ?

Contient code, data, info link (pas d'information de où il est chargé en mémoire).

Quel est le but d'une relocation ?

Appel de fonction : ` extern int bar();

int foo() { return bar(); } ` -> je sais pas où c'est, ce sera réglé au link.

.type : type d'un symbole .global : binding de l'obj .hidden : pas global ? .text : raccourci .section .text .section ksyms

Comment on fait un function call quand on fait appel à une fonction d'une bibliothèque dynamique ?

call func@PLT : appel à un resolver, puis patcher la data au bon endroit -> PLT : jump à l'entrée de GOT préparation au resolver appel au resolver

Explain the role of ld

linker dans les toolchain : prend un paquet de .o pour linker en executable, càd regrouper les sections ensemble, les mapper, puis faire des relocs

ldso ?

interpréteur de programme qui a des bibli dynamiques : les charge, et permettre d'y accéder et de résoudre les noms

Prologue et epilogue ?

sauver et restaurer la stackframe de la fonction parente. -> stackframe : zone ac variable locales sur la pile.

Quelles opérations sont valides x86_64 ? lesquelles sont susceptibles de segfault ?

- 'cmpb %eax, %ebx' : pas valide : pas bonne taille de registre (byte pour registre de 32)
- 'cmpl %eax, (%rsi, %rcx, 4) : valide
- cmpl (%rsi, %rcx, 4), (%rdi, %rcx, 4) : invalide
- mov $0, %rax : valide
- mov $0, (%rax) : valide
- mov 0, %rax : valide, mais SEGV car déref de 0 dans rax
- mov 0, (%rax) : pas valide, deux dérefs mémoire -> fonctions peuvent être valides ET segv

Traduire en assembleur:

```
int blob(int *a, long b, short c)
{
```

```
  return a[b] + c;
}
```

Solution

```
   .global blob
  .type blob, @function

blob:
  push %rbp
  mv %rsp, %rbp

  /* a => %rdi,
     b => %rsi
     c => %dx
      */

  xor %rax, %rax // <=> mov $0, %rax
  add %dx, %ax
  add (%rdi, %rsi, 4), %rax

  leave
  ret
  .size blob, .-blob
```

```
int blob(int *a, long b, short c);

int array[] = {
    1, 2, 3, 4, 5
};
```

Solution:

```
.data
  .global array
  .type array, @object

array:
  .long 1, 2, 3, 4, 5
  .size array, .-array
```

```
extern int array[];
extern int blob(int *a, long b, short c);

int main(int argc, char **argv)
{
  return blob(array, 2, 0x2a);
```

```
}
```

Solution:

```
   .global main
   .type main, @function

main:
  push %rbp
  mov %rsp, %rbp

  lea array, %rdi
  mov $2, %rsi
  mov $0x2a, %rdx

  call blob
  leave
  ret

  .size main, .-main
```