SOA2 (12/11/2024)

Nombre:
DNI:

Primer control de teoría

Responde brevemente a las siguientes preguntas justificando tus respuestas. Una respuesta sin justificar no será dada como válida.

1. (2 puntos) Syscall

Los procesadores Intel disponen de una 3era instrucción, a parte de INT y de SYSENTER, para forzar un cambio de modo de privilegios: SYSCALL. Cuando esta instrucción se ejecuta hace lo siguiente:
-En el registro ECX, guarda la dirección de la siguiente instrucción a SYSCALL.
-Modifica EIP con el valor del registro MSR LSTAR que guarda la dirección del handler de las llamadas al sistema.
-Cambia el nivel de privilegios a 0.

Supón que implementamos las llamadas al sistema mediante SYSCALL y que el registro MSR LSTAR tiene la dirección del handler de las llamadas al sistema.

a) Al ejecutar la primera instrucción del handler, ¿qué pila se está usando, la de modo usuario o la de modo sistema?

b) ¿Qué es lo primero que tiene que hacer el handler antes de guardar el contexto software?

c) Dentro del handler, ¿cómo se puede obtener la dirección de la pila de sistema del proceso actual?

d) ¿Qué se tiene que modificar del boot del sistema operativo para poder utilizar la instrucción en ensamblador SYSCALL?

e) A parte de sustituir SYSENTER por SYSCALL en los wrappers, y suponiendo que estás trabajando con un sistema operativo basado en Linux, ¿se tendría que hacer alguna modificación en los wrappers para el correcto funcionamiento de las llamadas al sistema?

SOA2 (12/11/2024)

Nombre:
DNI:

1. (2 puntos) Funciones auxiliares

Escribe en C el código de las siguientes funciones. Estas funciones se pueden llamar desde cualquier parte del código de sistema. Puedes suponer que estás trabajando con las estructuras y funciones de ZeOS (que tienes listadas al final del enunciado del control).

a) unsigned long ret_address(); Devuelve la direccion de retorno al código de usuario.

```
unsigned long ret_address()
{



}
```

b) unsigned long GetPhysicalAddressPCB(); Devuelve la direccion fisica de comienzo del union task_union del proceso actual.

```
unsigned long GetPhysicalAddressPCB()
{



}
```

c) unsigned long CalculateStackSize(); Devuelve el número de posiciones de la pila de sistema que están ocupadas.

```
unsigned long CalculateStackSize()
{



}
```

d) int IsFirstExecution(struct task_struct *pcb); Devuelve si el proceso pcb, que esta en Ready, se ha ejecutado alguna vez o no.

```
int IsFirstExecution(struct task_struct *pcb)
{



}
```

2. (3 puntos) Gestión de memoria:

Supón que tenemos una máquina con un procesador con una MMU que incluye un TLB y que usa tabla de páginas de un solo nivel. Considera que el número de entradas de esta tabla de páginas es T y que cada entrada hace referencia a una página de 4KBs. Sobre esta máquina montamos un sistema operativo que podemos modificar completamente.

En este sistema operativo, todos los procesos son iguales y su imagen en memoria está compuesta de las siguientes secciones:
-K páginas de kernel que incluyen el código, datos y pilas de kernel. Siendo K > 0. El kernel siempre está mapeado desde la página lógica 1 a la K, ambas inclusive.
-C páginas de código de usuario. Siendo C > 0.
-D páginas de datos y pila de usuario. Siendo D > 0.

Sys_fork está codificado para que utilice la primera sección libre (sin tener en cuenta la página lógica 0) dentro de la imagen del proceso en memoria para realizar la copia de la sección de datos del proceso de padre a hijo.

SOA2 (12/11/2024)

Nombre:
DNI:

Nos planteamos como influye la distribución de secciones (código, datos y pila) de la imagen del proceso en el rendimiento del bucle de copia de la sección de datos en sys_fork.

a) Describe como se tendrían que distribuir las secciones de la imagen del proceso para reducir el tiempo de ejecución del bucle de copia de la zona de datos en sys_fork

b) Indica cuantos flushes de TLB se tienen que realizar en el bucle de copia de la zona de datos con la distribución de secciones del apartado anterior

c) Describe como se tendrían que distribuir las secciones de la imagen del proceso para tener el peor tiempo de ejecución posible del bucle de copia

d) Indica cuantos flushes de TLB se tienen que realizar en el bucle de copia de la zona de datos con la distribución anterior.

Ahora modificamos sys_fork para que realice la copia en la sección libre más grande dentro de la imagen del proceso.

e) Indica que características deben tener K, C, D y S para tener un resultado similar al del apartado d.

Ahora suponemos que la imagen del proceso es la siguiente:
-de la página 1 a la K está el kernel.
-a continuación, se encuentra la sección de código de usuario.
-a continuación, existe una sección libre de tamaño D páginas.
-y después, secuencialmente, la sección de datos y la pila de usuario.

Nombre:
DNI:

f) ¿Cuál es la primera dirección lógica libre en esta imagen? Llama a esta dirección PLIF.

g) (1 punto) Escribe el bucle de copia de la zona de datos de sys_fork con esta imagen del proceso en memoria. Puedes asumir que en el vector int frames[D] tienes los identificadores de las páginas físicas para la zona de datos del proceso hijo.

3. (3 puntos) fork_func

El Señor Baka Baka, gran amante de los sistemas operativos, después de pagar sus deudas con la justicia, ha decidido seguir haciendo la vida imposible a los estudiantes de SOA2 con nuevas e inútiles modificaciones de ZeOS. La nueva modificación que propone es crear una nueva llamada al sistema, fork_func, que cree un nuevo proceso, como fork, pero que en vez de ejecutar la siguiente instrucción después de fork, ejecute, en el proceso hijo, una función que se pasará como parámetro. El padre continuará su ejecución como en fork. Así, la nueva llamada al sistema es: `int fork_func(void (*func)(void *param), void *param);` donde func es la función que se tiene que ejecutar y param es el parámetro que se puede pasar a esta función. La cabecera de func es: `void func(void *param);`

Para implementarla, se basará en el wrapper de fork, en el handler de las llamadas al sistema y en sys_fork (hará una copia de las 3 funciones añadiendo al nombre _func):

Wrapper de fork_func

```
01 ENTRY(fork_func)
02   push %ebp
03   mov %esp, %ebp
04   push %ebx
05   mov 8(%ebp), %ebx
06   mov 12(%ebp), %ecx
07   mov $33, %eax
08   int $0x80
09   cmp $0, %eax
10   jl fin
11   neg %eax
12   mov %eax, errno
13   mov $-1, %eax
14 fin:
15   pop %ebx
16   pop %ebp
17   ret
```

Syscall_handler

```
01 ENTRY(syscall_handler)
02   SAVE_ALL
03   cmp $0, %eax
04   jl err
05   cmp $MAX_SYSCALL, %eax
06   jg err
07   call *syscall_table(, %eax, 4)
08   jmp fin
09 err:
10   mov $-ENOSYS, %eax
11 fin:
12   mov %eax, 0x18(%esp)
13   RESTORE_ALL
14   iret
```

Nombre:
DNI:

a) Dibuja la pila de usuario antes de ejecutar la línea 08 del wrapper de fork_func incluyendo su frame de activación e indicando a donde apunta el registro esp del procesador.

b) Dibuja el contenido de la pila de sistema antes de la instrucción 07 del syscall_handler

c) Para conseguir el objetivo, decide modificar la pila del usuario cambiando la dirección de retorno del wrapper fork_func por la dirección de func. Dibuja como tiene que quedar la pila de usuario.

Después de pensarlo mucho, el señor Baka Baka ve tres posibles implementaciones para esta solución: (opción A) modificando solamente sys_fork_func, (opción B) modificando solamente ret_from_fork_func y (opción C) modificando solamente el wrapper fork_func.

d) (Opción A) Escribe en pseudocódigo las modificaciones que se tienen que hacer en sys_fork_func indicando en qué parte de sys_fork_func se tienen que incluir.

e) (Opción B) Escribe en pseudocódigo la nueva función ret_from_fork_func teniendo en cuenta que es la única función que se modifica.

Nombre:
DNI:

f) (Opción C) Escribe en ensamblador las modificaciones que se tienen que hacer en el wrapper fork_func indicando entre qué líneas se tienen que añadir y suponiendo que ni sys_fork ni ret_from_fork se modifican.

Información del Sistema Operativo

El sistema dispone de las siguientes rutinas:

- ***struct task_struct *current():*** Retorna la task_struct del proceso actual.
- ***int alloc_frame()*:** Reserva un frame de memoria física.
- ***void free_frame (int frame)***: Libera un frame de memoria.
- ***page_table_entry * get_PT(struct task_struct *t):*** Retorna la tabla de páginas del proceso *t*.
- ***page_table_entry * get_DIR(struct task_struct *t):*** Retorna el directorio de páginas del proceso *t*.
- ***set_CR3 (page_table_entry * dir):*** Sobreescribe el registro CR3 con el nuevo directorio de páginas *dir*, provoca una invalidación de la TLB.
- ***int get_frame (page_table_entry *pt, int logical_page)***: Retorna el frame asignado a una página lógica en la tabla de páginas de un proceso determinado.
- ***int set_ss_page (page_table_entry *pt, int logical_page, int frame):*** Asigna un *frame* a una página lógica de la tabla de páginas *pt*.
- ***int del_ss_page (page_table_entry *pt, int logical_page)*:** Borra una entrada de la tabla de páginas.