**TP 3**

**Conceptos Generales**

**1. ¿Qué es una System Call?, ¿para que se utiliza?**

Son llamados al kernel para ejecutar una función específica que controla un dispositivo o ejecuta una instrucción privilegiada

* Su propósito es proveer una interfaz común para lograr portabilidad
* Su funcionalidad se ejecuta en modo Kernel pero en contexto del proceso

Las llamadas al sistema comúnmente usan una instrucción especial de la CPU que causa que el procesador transfiera el control a un código privilegiado (generalmente es el núcleo), previamente especificado. Esto permite al código privilegiado especificar donde va a ser conectado así como el estado del procesador.

Cuando una llamada al sistema es invocada, la ejecución del programa que invoca es interrumpida y sus datos son guardados, normalmente en su PCB (Bloque de Control de Proceso del inglés Process Control Block), para poder continuar ejecutándose luego. El procesador entonces comienza a ejecutar las instrucciones de código de bajo nivel de privilegio, para realizar la tarea requerida. Cuando esta finaliza, se retorna al proceso original, y continúa su ejecución. El retorno al proceso demandante no obligatoriamente es inmediato, depende del tiempo de ejecución de la llamada al sistema y del algoritmo de planificación de CPU.

## 2. ¿Para qué sirve la macro syscall?. Describa el propósito de cada uno de sus parámetros.

man syscall

[...]

SYNOPSIS

#define \_GNU\_SOURCE /\* See feature\_test\_macros(7) \*/

#include <unistd.h>

#include <sys/syscall.h> /\* For SYS\_xxx definitions \*/

long syscall(long number, ...);

[...]

La macro syscall es una función que provee la librería C, que permite hacer una system call de manera explícita.

Es una función que permite invocar una system call efectiva cuya interfaz de código assembler tiene el número especificado number con los argumentos especificados .... Se utiliza comunmente cuando se desea invocar una llamada al sistema la cual no tiene un binding directo a alguna funcionalidad de C existente.

syscall() guarda los registros de la CPU antes de realizar la llamada al sistema, restaura los registros al regresar de la llamada al sistema y almacena cualquier código de error devuelto por la llamada al sistema en errno si ocurre un error.

Las constantes simbólicas para los números de llamadas al sistema se pueden encontrar en el archivo de encabezado <sys/syscall.h>.

syscall esta declarada en unistd.h. Function: long syscall(long number, ...);

sysno es el numero de la system call. cada tipo de system call esta identificada por un numero. Macros para todas las posibles system call estan definidas en sys/syscall.h

El resto de los argumentos son argumentos para la system call, en orden, y su significado depende del tipo de system call. cada tipo de system call tiene un numero definido de argumentos, entre 0 y 5. si envias mas argumentos que los que la system call recibe, estos van a ser ignorados.

## 3. ¿Para qué sirven los siguientes archivos?

### <kernel\_code>/arch/x86/syscalls/syscall\_32.tbl

# 32-bit system call numbers and entry vectors

#

# The format is:

# <number> <abi> <name> <entry point> <compat entry point>

Posee los números de System Calls de 32bits y vectores de entrada.

El formato es <number> <abi> <name> <entry point> <compat entry point>

Ejemplo:

0 i386 restart\_syscall sys\_restart\_syscall

1 i386 exit sys\_exit

2 i386 fork sys\_fork stub32\_fork

3 i386 read sys\_read

4 i386 write sys\_write

…

…

### <kernel\_code>/arch/x86/syscalls/syscall\_64.tbl

# 64-bit system call numbers and entry vectors

#

# The format is:

# <number> <abi> <name> <entry point>

Posee los números de System Calls de 64bit y vectores de entrada.

El formato es <number> <abi> <name> <entry point>

Ejemplo:

0 common read \_\_x64\_sys\_read

1 common write \_\_x64\_sys\_write

2 common open \_\_x64\_sys\_open

…

…

Son los archivos en los cuales se declaran las system calls accesibles para 32 y 64 bist respectivamente. Si deseamos agregar una syscall nueva debemos agregar una nueva entreada al archivo. En el archivo syscall\_64.tbl se debe tener particular cuidado al momento de agregar una nueva entreada dado que existen algunas con <abi> = x32. Se deberá buscar el final de las entradas con <abi>=common y agregar la nueva ahí con ese formato.

### 4. ¿Para qué sirve la macro asmlinkage?

asmlinkage long sys\_custom\_syscall(void);

Es una macro que permite indicarle al compilador, en principio gcc, que la función que la función no debe esperar encontrar ninguno de sus argumentos en los registros sino solo en el stack de la CPU. Se realiza como idea de optimización. La syscall toma su primer argumento (número de system call) y permite hasta 4 argumentos más que se pasan al llamado del sistema real. Para lograr esto, se dejan todos los argumentos que se pasaron en la pila del CPU.

Todas las llamadas al sistema están marcadas con la etiqueta asmlinkage, por lo que todas buscan en la pila en busca de argumentos. Por supuesto, en el caso de sys\_ni\_syscall, esto no hace ninguna diferencia, porque sys\_ni\_syscall no acepta ningún argumento, pero es un problema para la mayoría de las otras system call.

La macro asmlinkage instruye al compilador a pasar parametros por stack a la hora de declarar una nueva system call.

**Asmlinkage instruye al compilador a pasar parametros por stack y no por ejemplo en registros.**

### 5. ¿Para qué sirve la herramienta strace?, ¿Cómo se usa?

strace es una herramienta que permite corroborar errores del sistema operativo GNU/Linux. Permite monitorear las system calls utilizadas por un determinado programa y las señales que este recibe. Su funcionamiento es posible por una característica del kernel de Linux llamada ptrace.

strace es una utilidad de línea de comandos para comprobación de errores en el sistema operativo GNU/Linux. Permite monitorear las llamadas al sistema usadas por un determinado programa y todas las señales que éste recibe.​ Su funcionamiento es posible por una característica del núcleo linux llamada ptrace.

Su uso más común consiste en arrancarlo junto al programa al que se le efectúa el trazado, el cual imprime una lista de llamadas al sistema que dicho programa ejecuta. Es útil para averiguar la causa del fallo de un programa determinado porque informa de situaciones en las que por ejemplo, el programa está intentando acceder a un fichero que no existe o que no tiene permiso de lectura.

## Monitoreando System Calls

## Módulos y Drivers

### 1. ¿Cómo se denomina en Gnu/Linux a la porción de código que se agrega al kernel en tiempo de ejecución? ¿Es necesario reiniciar el sistema al cargarlo?. Si no se pudiera utilizar esto. ¿Cómo deberíamos hacer para proveer la misma funcionalidad en Gnu/Linux?

Estas porciones de código son los denominados kernel modules los cuales son compilados sin estar incluidos en el código del kernel evitando que el mismo sea un kernel monolítico, en el cual los módulos formarían parte de su código.

Los módulos del kernel son pedazos de código que han sido compilados sin estar incluídos en el kernel. Cuando se compila el kernel, se puede seleccionar que determinadas funcionalidades no sean incluidas en forma nativa en el kernel, sino como módulos, y luego estos pueden ser cargados en tiempo de ejecución.

Sin módulos el kernel seria 100% monolitico. Las funcionalidades implementadas en estas deberían entonces ser incluidas dentro del código del kernel.

### 2. ¿Qué es un driver? ¿Para que se utiliza?

Un driver es un software que conecta el OS directamente con las componentes de hardware de la PC. Por ejemplo, si tenemos una placa de vídeo instalada en la computadora, esta necesita entenderse con el sistema operativo para poder recibir las instrucciones y procesar todo correctamente; y precisamente esta es la función que cumple el controlador, un puente entre ambos. El driver le da instrucciones al sistema operativo sobre cómo debe funcionar determinado hardware y de que forma el sistema debe trabajar en conjunto para suministrarte los mejores resultados.

### 3. ¿Porque es necesario escribir drivers?

La actualización de hardware y existencia de nuevas necesidades de forma constante son la razón por la cual siempre existe la necesidad de definir nuevos drivers logrando que nuevos productos sean compatibles con OSs existentes y permitiendo que el mismo llegue a más y más usuarios.

### 4. ¿Cuál es la relación entre modulo y driver en Gnu/Linux?

Un módulo del kernel es un código compilado que se puede insertar en el kernel en tiempo de ejecución, como con insmod o modprobe.

Un controlador es un poco de código que se ejecuta en el núcleo para comunicarse con algún dispositivo de hardware. Se "impulsa" el hardware. La mayoría de cada hardware en su computadora tiene un controlador asociado. Una gran parte de un núcleo en ejecución es el código del controlador.

Un controlador puede incorporarse estáticamente en el archivo del kernel en el disco. Un controlador también puede construirse como un módulo del kernel para que luego pueda cargarse dinámicamente.

La práctica estándar es construir controladores como módulos del kernel siempre que sea posible, en lugar de vincularlos estáticamente al kernel, ya que eso da más flexibilidad. Hay buenas razones para no hacerlo, sin embargo:

A veces, un controlador dado es absolutamente necesario para ayudar al arranque del sistema. Eso no sucede con la frecuencia que pueda imaginar, debido a la función initrd.

Los controladores construidos estáticamente pueden ser exactamente lo que usted desea en un sistema con un alcance estático, como un sistema integrado. Es decir, si sabe de antemano exactamente qué controladores siempre serán necesarios y que esto nunca cambiará, tiene una buena razón para no molestarse con los módulos dinámicos del kernel.

Si construye su kernel de forma estática y desactiva la función de carga dinámica del módulo de Linux, evita la modificación en tiempo de ejecución del código del kernel. Esto proporciona seguridad y estabilidad adicionales a expensas de la flexibilidad.

No todos los módulos del kernel son controladores. Por ejemplo, una característica relativamente reciente en el kernel de Linux es que puede cargar un programador de procesos diferente. Otro ejemplo es que los tipos de hardware más complejos a menudo tienen múltiples capas genéricas que se ubican entre el controlador de hardware de bajo nivel y la zona de usuario, como el controlador USB HID, que implementa un elemento particular de la pila USB, independientemente del hardware subyacente.

Un driver es un módulo que extiende funcionalidad al sistema para acceder a distintos dispositivos. La diferencia es que un módulo solo extiende funcionalidad al kernel, un driver es un módulo con capacidad de acceder a dispositivos.

**Un driver es un modulo**

**Cada dispositivo de hardware es un archivo (abstracción)**

**Ejemplo: /dev/hda. En realidad no es un archivo. Si leemos/escribimos desde el lo hacemos sobre datos “crudos” del disco (bulk data).**

**Accedemos a estos archivos mediante operaciones basicas (espacio del kernel).**

**read, write: escribir y recuperar bulk data**

**ioctl: configurar el dispositivo**

### 5. ¿Qué implicancias puede tener un bug en un driver o módulo?

“This design of modules makes it faster to load the Kernel (no need to load and initialize un-needed Modules), as well as developing Drivers (if you have a bug, just unload the Module, fix it, recompile and load it again).”

**Si ocurre un bug en un driver o modulo, solo seria necesario descargar ese modulo (rmmod), reparar el bug, recompilarlo y volverlo a cargar. No seria necesario recompilar el so completo, ni reiniciarlo.**

### 6. ¿Qué tipos de drivers existen en Gnu/Linux?

Podemos clasificar el hard en varios tipos:

* Dispositivos de acceso aleatorio(ej. discos).
* Dispositivos seriales(ej. Mouse, sonido,etc).

Acorde a esto los drivers se clasifican en:

* Drivers de bloques: son un grupo de bloques de datos persistentes. Leemos y escribimos de a bloques, generalmente de 1024 bytes.
* Drivers de carácter: Se accede de a 1 byte a la vez y 1 byte solo puede ser leıdo por única vez.
* Drivers de red: tarjetas ethernet, WIFI, etc.

### 7. ¿Que hay en el directorio /dev? ¿qué tipos de archivo encontramos en esa ubicación?

/dev es la ubicación de archivos especiales o de dispositivo. Es un directorio muy interesante que destaca un aspecto importante del sistema de archivos de Linux: todo es un archivo o un directorio. Mire a través de este directorio y esperemos que vea hda1, hda2, etc. que representan las distintas particiones en la primera unidad maestra del sistema. /dev/cdrom y /dev/fd0 representan su unidad de CD-ROM y su unidad de disquete. Esto puede parecer extraño, pero tendrá sentido si compara las características de los archivos con las de su hardware. Ambos se pueden leer y escribir . Tome /dev/dsp, por ejemplo. Este archivo representa su dispositivo de altavoz. Cualquier dato escrito en este archivo será redirigido a su orador. Si prueba 'cat /boot/vmlinuz >/dev/dsp' (en un sistema configurado correctamente) debería escuchar un sonido en el altavoz. Ese es el sonido de tu núcleo! Se imprime un archivo enviado a /dev/lp0. Enviar datos y leer desde /dev/ttyS0 le permitirá comunicarse con un dispositivo conectado allí, por ejemplo, su módem.

La mayoría de los dispositivos son dispositivos de bloque o de caracteres; sin embargo existen otros tipos de dispositivos y pueden ser creados. En general, los "dispositivos de bloque" son dispositivos que almacenan o mantienen datos, los "dispositivos de caracteres" pueden considerarse como dispositivos que transmiten o transfieren datos. Por ejemplo, las unidades de disquete, las unidades de disco duro y las unidades de CD-ROM son dispositivos de bloque, mientras que los puertos serie, los ratones y los puertos de impresora paralelos son dispositivos de caracteres. Hay un tipo de esquema de nombres, pero en la gran mayoría de los casos son completamente ilógicos.

El directorio /dev contiene los archivos de dispositivos especiales para todos los dispositivos hardware. Los archivos de dispositivos se nombran utilizando convenciones especiales, como por ejemplo sda, sdb, sdc, siendo cada letra un dispositivo distinto, y se le agrega un número como por ej sda1, sda2, a cada partición del dispositivo, en este caso es un dispositivo con 2 particiones.

### 8. ¿Para qué sirven el archivos /lib/modules//modules.dep utilizado por el comando modprobe

modprobe examina el archivo /lib/modules/version/modules.dep para ver si se deben cargar otros módulos antes de que se cargue el módulo solicitado. Este archivo es creado por depmod -a y contiene dependencias de módulo. Por ejemplo, msdos.ko requiere que el módulo fat.ko ya esté cargado en el kernel. El módulo solicitado depende de otro módulo si el otro módulo define los símbolos (variables o funciones) que utiliza el módulo solicitado.

En /lib/modules (y en sus subdirectorios) están los archivos de los módulos del kernel (creo que los que podemos cargar bajo demanda en el kernel corriendo, no los que está corriendo <<ahora mismo el kernel).

El comando modprobe emplea información para cargar un módulo específico.

El comando insmod instala los módulos específicos.

**System Calls**

**Agregando un módulo a nuestro kernel**

**Explique brevemente cual es la utilidad del archivo Makefile en la organización de un**

**proyecto de linux.**

Un archivo Makefile es un archivo de texto en el cual se distinguen cuatro tipos básico de declaraciones:

* Comentarios.
* Variables.
* Reglas explícitas.
* Reglas implícitas.

Sirven para poder ejecutar comandos como *make.*

2 - i - Makefile es una herramienta para la construcción de programas que corren en Unix, Linux y derivados. Ayuda a simplificar la construcción de programas ejecutables que pueden necesitar varios módulos.

MODULE\_LICENSES:

The big difference is the symbols that are exposed to your module when it is GPL licensed. If you do not have MODULE\_LICENSE("GPL") then symbols that are exported via EXPORT\_SYMBOL\_GPL() will not be visible to your module. This is basically how Linux enforces what work is considered derivative work of Linux and therefore must be licensed under GPL, although the subject of proprietary modules is always a touchy one and I'm not saying if you don't use any GPL exported symbols your code is not a derivative work of Linux. Furthermore, if your module is not GPL licensed via the macro, your kernel will be considered [tainted](https://unix.stackexchange.com/questions/118116/linux-what-is-a-tainted-kernel).

When the kernel is tainted, it means that **it is in a state that is not supported by the community**. Most kernel developers will ignore bug reports involving tainted kernels, and community members may ask that you correct the tainting condition before they can proceed with diagnosing problems related to the kernel. In addition, some debugging functionality and API calls may be disabled when the kernel is tainted.

**In most cases involving proprietary drivers, you can safely ignore the taint condition**, but some scenarios which cause the kernel to become tainted may be indicative of serious system problems

insmod: Trata de cargar el módulo especificado.

modprobe: Emplea la información generada por depmod e información de /etc/modules.conf para cargar el módulo especificado.

lsmod: Lista los módulos cargados (es equivalente a cat /proc/modules)

rmmod: Descarga uno o más módulos

**¿Para qué sirve la macro MODULE\_LICENSE? ¿Es obligatoria?**

MODULE\_LICENSE () es una macro que permite que los módulos de kernel cargables declaren su licencia al mundo. Su propósito es informar a los desarrolladores del núcleo cuándo se ha insertado un módulo no libre en un kernel dado. Si envía un informe ¡Vaya que muestra un kernel "contaminado", es probable que se le pida que reproduzca el problema sin cargar el módulo propietario, o que hable con el proveedor de ese módulo sobre el problema. En general, los hackers del kernel quieren saber de los problemas, pero su interés disminuye notablemente cuando no pueden llegar a la fuente para diagnosticar o /solucionar el problema.

¿Para qué sirven las funciones module\_init y module\_exit?. ¿Cómo haría para ver la  
información del log que arrojan las mismas?.

**Sirven para indicar que función se debe ejecutar cuando el módulo es cargado o descargado del kernel. Utilizando Strace.**

Clasifique los tipos de dispositivos en Linux. Explique las características de cada uno.

**Podemos clasificar el hard en varios tipos.**

**Dispositivos de acceso aleatorio(ej. discos).**

**Dispositivos seriales(ej. Mouse, sonido,etc).**