1. UNIDAD 1: INTRODUCCION

Escuela de Ingeniería de Ciencias Y Sistemas
Facultad de Ingeniería
Universidad de San Carlos de Guatemala

Anuncios Importantes

Tema 1
Tema 2
Tema 3
Tema 4

AGENDA

1. Personalizando el kernel de Linux
2. Depuración y prueba del kernel
3. Definición y función de las llamadas al sistema
4. Flujo de llamadas del sistema
5. Llamadas comunes al sistema en Linux
6. Desarrollo de llamadas al sistema

COMPETENCIA(S) QUE DESARROLLAREMOS

- Entender el marco de referencia o estructura lógica general de un sistema operativo, que le permita la utilización, análisis y diseño de sistemas operativos.
- Desarrollar e implementar nuevos sistemas operativos y modificar funcionalidades de sistemas operativos existentes

PERSONALIZAR EL KERNEL DE LINUX

Configuración:

El proceso comienza con la configuración, un paso que determina las características y módulos que incluirá el kernel. Herramientas como make menuconfig ofrecen una interfaz gráfica para alternar opciones, guiando a los usuarios a través de la infinidad de posibilidades.

Compilando el kernel:

Con la configuración establecida, la siguiente fase es la compilación, que transforma el código fuente en un kernel ejecutable. Este proceso se inicia con comandos como make y make modules_install, un procedimiento meticuloso que puede llevar tiempo, dependiendo de las capacidades del sistema y la complejidad del kernel.

PERSONALIZAR EL KERNEL DE LINUX

Instalación del kernel personalizado

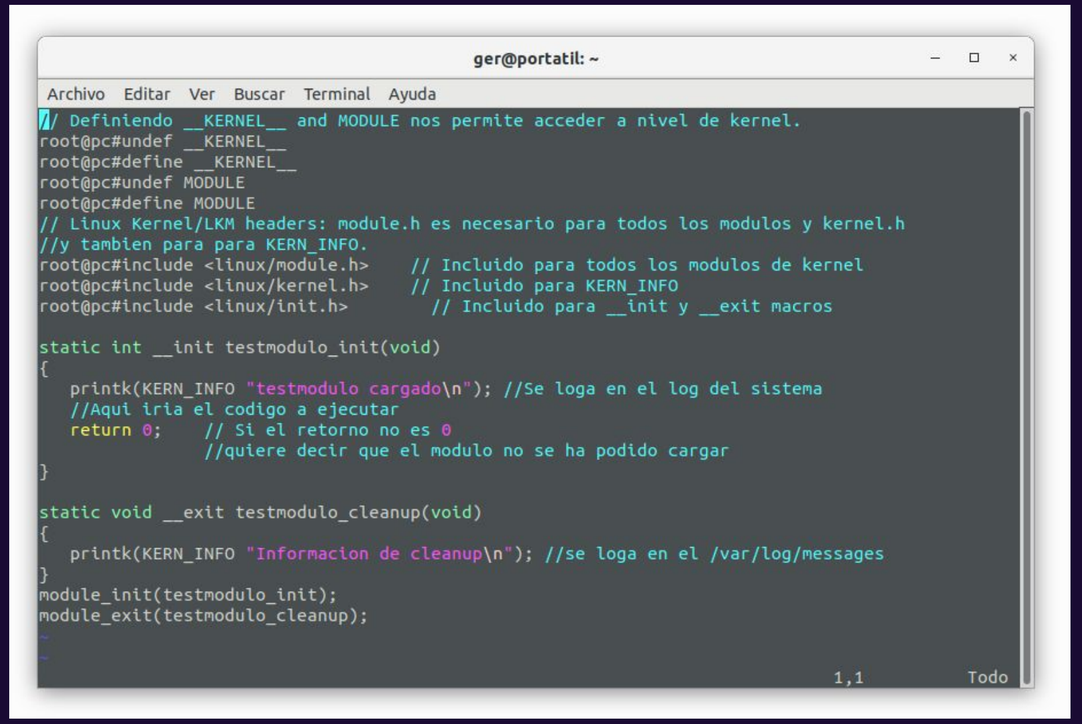
Tras la compilación, se instala el nuevo kernel. Esto implica copiar la imagen del kernel al directorio de arranque y, posiblemente, actualizar la configuración del gestor de arranque para incluir el nuevo kernel como opción de arranque, garantizando así que el sistema pueda arrancar con este núcleo personalizado.

Pruebas y validación

El último paso es la prueba, donde se reinicia el sistema con el nuevo kernel. Esta fase es crucial para verificar que el sistema funciona como se espera, con todo el hardware reconocido y funcionando correctamente.

Depuración y Prueba

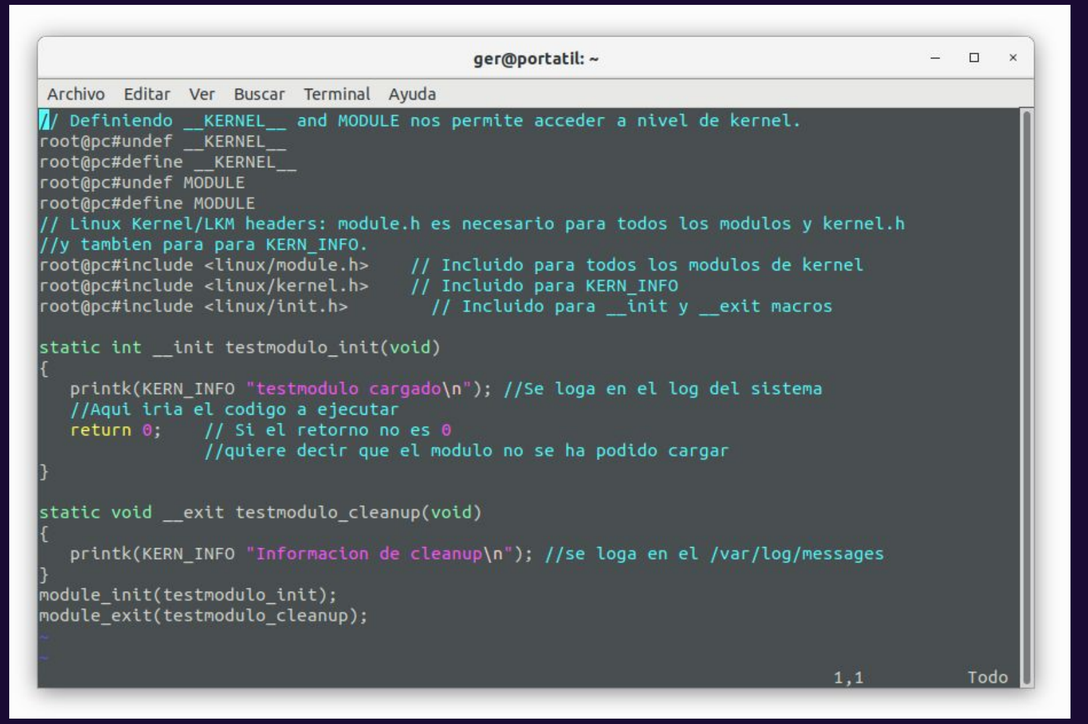
Los logs del kernel son mensajes que el sistema operativo genera en tiempo de ejecución para informar sobre eventos importantes, advertencias, errores, o simplemente para ayudar en el seguimiento del estado del sistema. En el contexto de desarrollo del kernel, los logs son vitales para entender qué está ocurriendo dentro del núcleo sin necesidad de herramientas invasivas.



¿Cómo se generan los logs?

Se utiliza la función printk(), análoga a printf() en espacio de usuario:

```
printk(KERN_INFO "Este es un mensaje de prueba: %d\n", variable);
```



NIVELES DE SEVERIDAD

Log Level	Base	For Kernel	For Device Drivers
0	printk(KERN_EMERG…)	pr_emerg()	dev_emerg()
1	printk(KERN_ALERT…)	pr_alert()	dev_alert()
2	printk(KERN_CRIT…)	pr_crit()	dev_crit()
3	printk(KERN_ERR…)	pr_err()	dev_err()
4	printk(KERN_WARNING…)	pr_warning() pr_warn()	dev_warn()
5	printk(KERN_NOTICE…)	pr_notice()	dev_notice()
6	printk(KERN_INFO…)	pr_info()	dev_info()
7	printk(KERN_DEBUG…)	pr_debug() pr_devel()	dev_dbg() dev_vdbg()
N/A	printk(KERN_CONT…)	pr_cont()	N/A

Nivel	Constante	Descripción
0	KERN_EMERG	Emergencia, el sistema no sirve
1	KERN_ALERT	Se requiere acción inmediata
2	KERN_CRIT	Estado crítico del sistema
3	KERN_ERR	Error
4	KERN_WARNING	Advertencia
5	KERN_NOTICE	Información importante
6	KERN_INFO	Información general
7	KERN_DEBUG	Información para depuración

¿Donde ver estos mensajes?

Los logs del kernel se almacenan en un buffer circular de memoria accesible mediante herramientas del espacio de usuario:

a. dmesg
Muestra el buffer de logs del kernel:

```
dmesg | grep mi_syscall
```

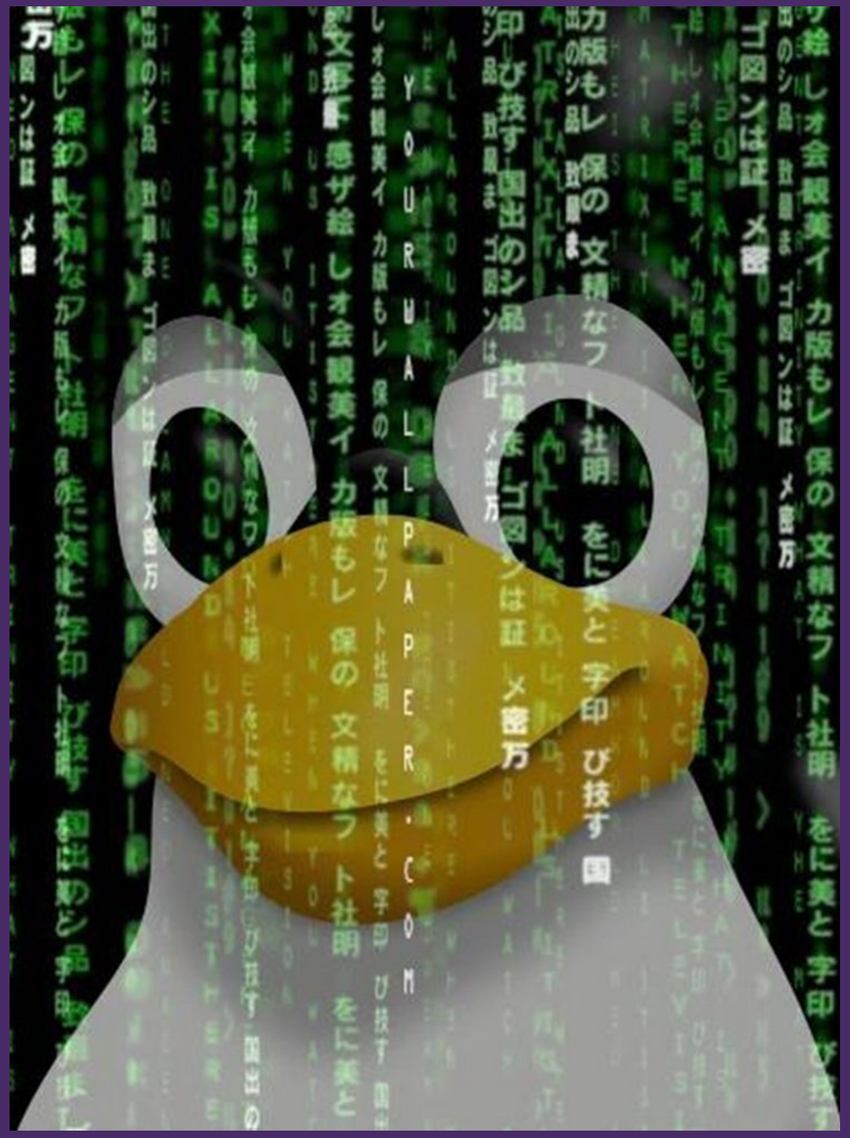
b. /var/log/kern.log o /var/log/messages
Dependiendo de la configuración del sistema (y de rsyslog o journald), los logs del kernel también se guardan en archivos persistentes.

c. journalctl (para sistemas con systemd)

```
journalctl -k    # Solo logs del kernel
journalctl -k -p debug  # Solo nivel debug
```

LABORATORIO SISTEMAS OPERATIVOS 2

LLAMADAS AL SISTEMA



LLAMADAS AL SISTEMA

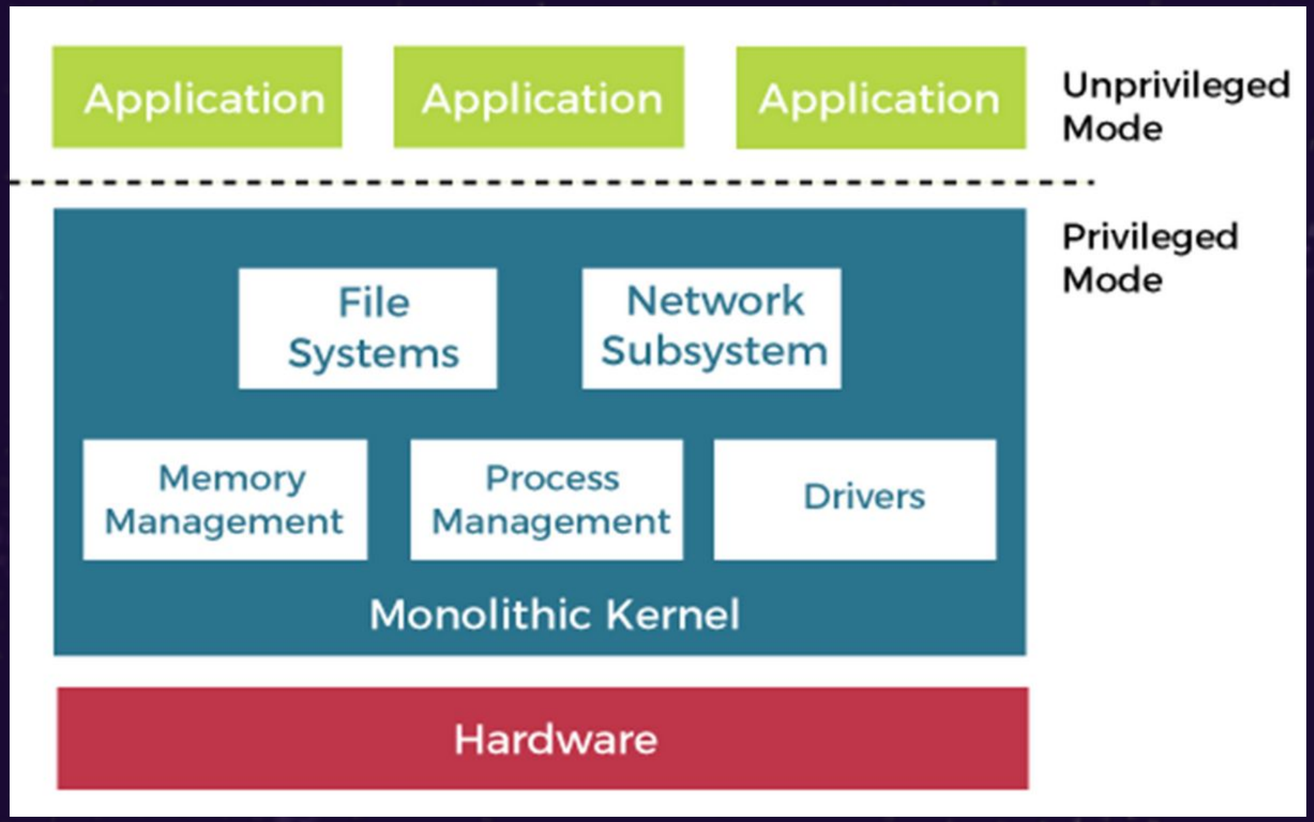
Una llamada al sistema, o syscall,es una rutina que permite a una aplicación de usuario solicitar acciones que requieren privilegios especiales. La adición de llamadas al sistema es una de varias maneras de ampliar las funciones proporcionadas por el kernel.

En los sistemas operativos modernos, este método se utiliza si una aplicación o proceso de usuario necesita pasar información al hardware, a otros procesos o al propio kernel, o si necesita leer información de estas fuentes.

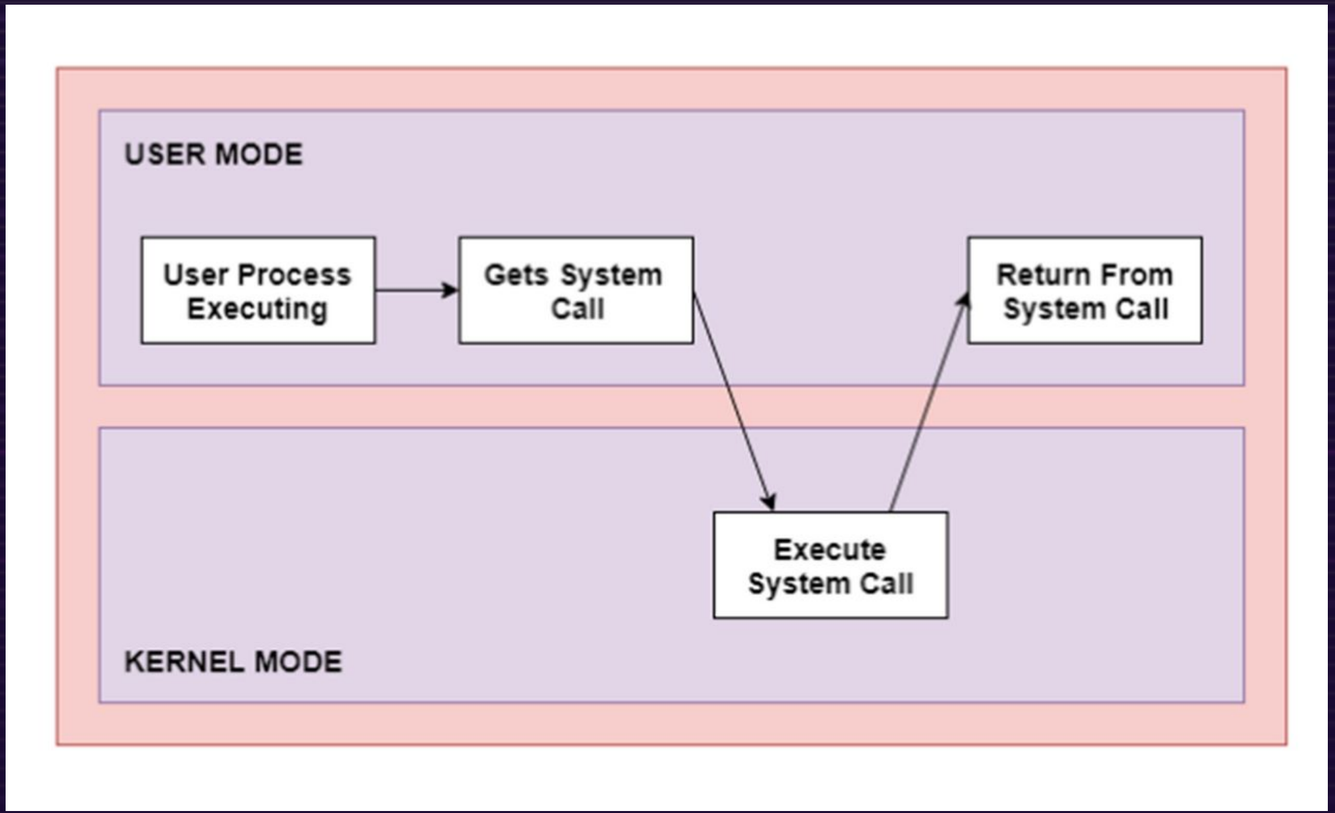
Esto hace que estas llamadas sean un vínculo entre el modo de usuario y el modo kernel

La necesidad de llamadas al sistema está estrechamente ligada al modelo de sistema operativo moderno con modo de usuario y modo kernel (monolítico).

Procesos en modo usuario se utilizan normalmente para realizar cálculos, acceder a recursos a nivel de usuario, como archivos y memoria, y gestionar el control de procesos. Estas tienen acceso limitado a los recursos del sistema y garantiza que los procesos no puedan interferir entre sí.



El modo kernel es el sistema de control fundamental donde se ejecutan todoslos servicios y procesos del sistema, asicomo realizar las acciones críticas del sistema por parte de los programas de aplicación que están bloqueados en el modo de usuario, comoconfigurar asignaciones de memoria o acceder a dispositivos de E/S

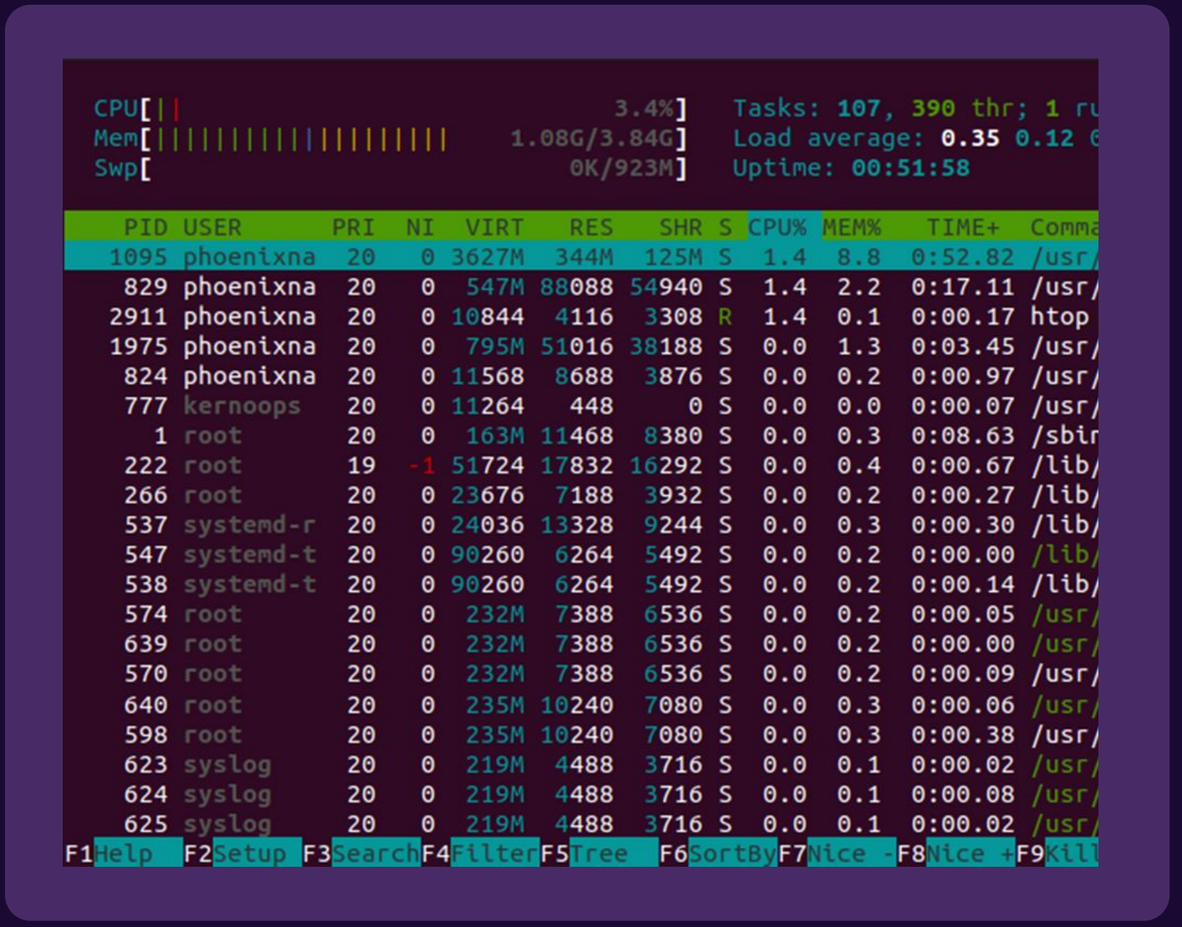


En Linux, realizar una llamada al sistema implica transferir el control del modo de usuario sin privilegios al modo de kernel privilegiado; Los detalles de esta transferencia varían de una arquitectura a otra. Las bibliotecas se encargan de recopilar los argumentos de la llamada al sistema y, si es necesario, de organizarlos en la forma especial necesaria para realizar la llamada al sistema.

LLAMADAS AL SISTEMA PARA GESTIÓN DE PROCESOS

La interfaz de llamadas de usuario suministra al desarrollador de software herramientas tanto para la creación, sincronización y comunicación de nuevos procesos, como la capacidad de ejecutar nuevos programas.

Las llamadas al sistema para gestion de procesos de Linux son fork(), exit(), exec(), aunque tambien exploraremos las señales entre procesos, utilizadas para la comunicación entre procesos.



FORK( )

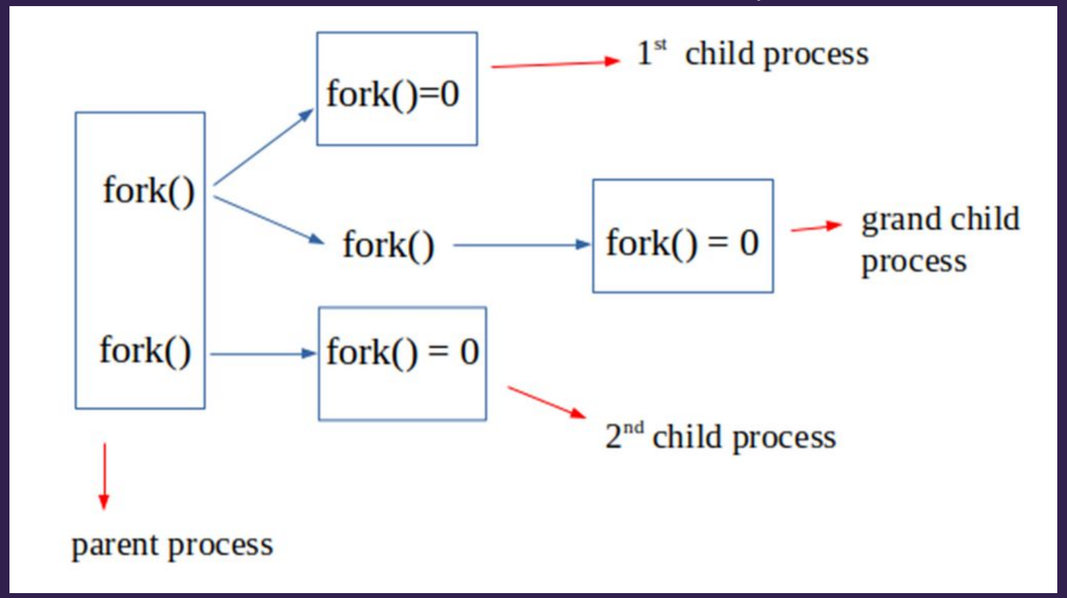
El kernel crea un nuevo proceso (proceso hijo) realizando una copia del proceso que realiza la llamada al sistema fork (proceso padre). Así, salvo el PID y el PPID los dos procesos serán inicialmente idénticos. De esta forma los nuevos procesos obtienen una copia de los recursos del padre (heredan el entorno).

El proceso hijo tendrá copias de los descriptores, si los utiliza el proceso padre. Sin embargo, las copias de los descriptores harán referencia a los mismos objetos subyacentes.

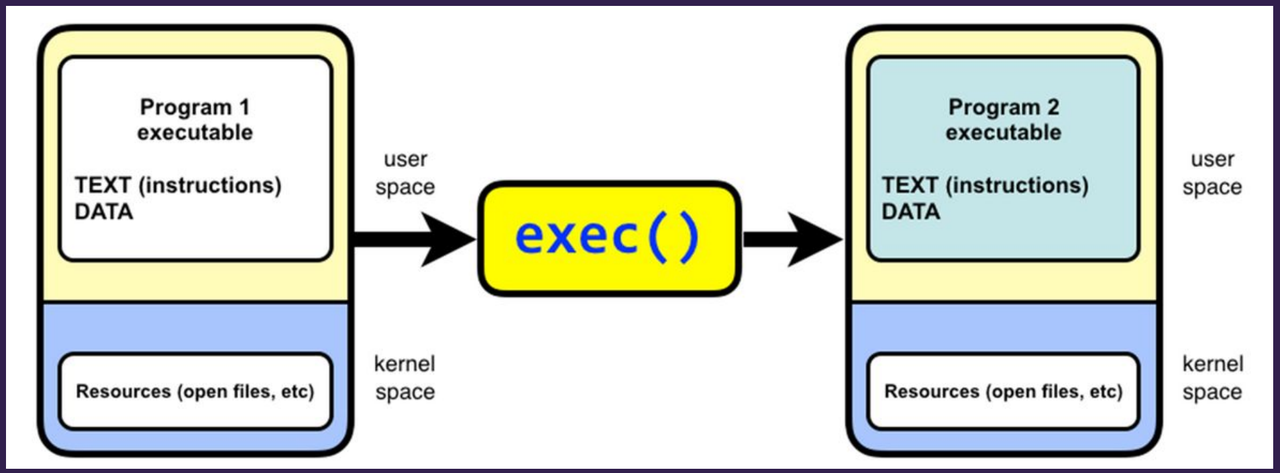
Su sintaxis es:
int pid = fork ();

Este retornará un valor diferente dependiendo de donde sea ejecutado, sus posibles valores son:

- 0 cuando se ejecuta desde el proceso hijo
- PID del nuevo proceso cuando se ejecuta del proceso padre
- -1 si ocurre un error al crear el nuevo proceso



EXEC( )

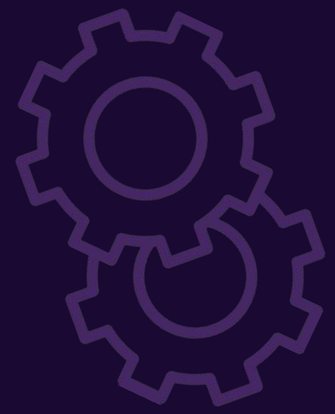


La llamada al sistema exec permite remplazarlos segmentos de instrucciones y de datos de usuario por otros nuevos a partir de un archivo ejecutable en disco, con lo que se consigue que un proceso deje de ejecutar instrucciones de un programa y comience a ejecutar instrucciones de un nuevo programa. exec() no crea ningún proceso nuevo.

El programa que se está ejecutando actualmente finaliza inmediatamente y el nuevo programa comienza aejecutarse en el contexto del proceso existente.

Hay 6 formas de realizar unallamada al sistema exec:

- int execl (char *path, char *arg0, char *arg1, . . . ,char *argN, char *null)
- int execlp (char *file, char *arg0, char *arg1, . . . ,char *argN, char *null)
- int execle (char *path, char *arg0, char *arg1, . . . ,char *argN, char *null, char *envp[])
- int execv (char *path, char *argv[])
- int execvp (char *file, char *argv[])
- int execve (char *path, char *argv[], char *envp[])



Donde:

- path = ruta del ejecutable del nuevo programa a ejecutar.
- file = nombre del nuevo programa a ejecutar.
- arg0 = primer argumento del programa. Por convención suele asignarse el nombre del programa sin la trayectoria.
- arg1 ... argN = Conjunto de parámetros que recibe el programa para su ejecución.
- argv = Matriz de punteros a cadenas de caracteres. Estas cadenas de caracteres constituyen la lista de argumentos disponibles para el nuevo programa.
- envp = Matriz de punteros a cadenas de caracteres. Estas cadenas de caracteres constituyenelentorno de ejecución del nuevo programa.

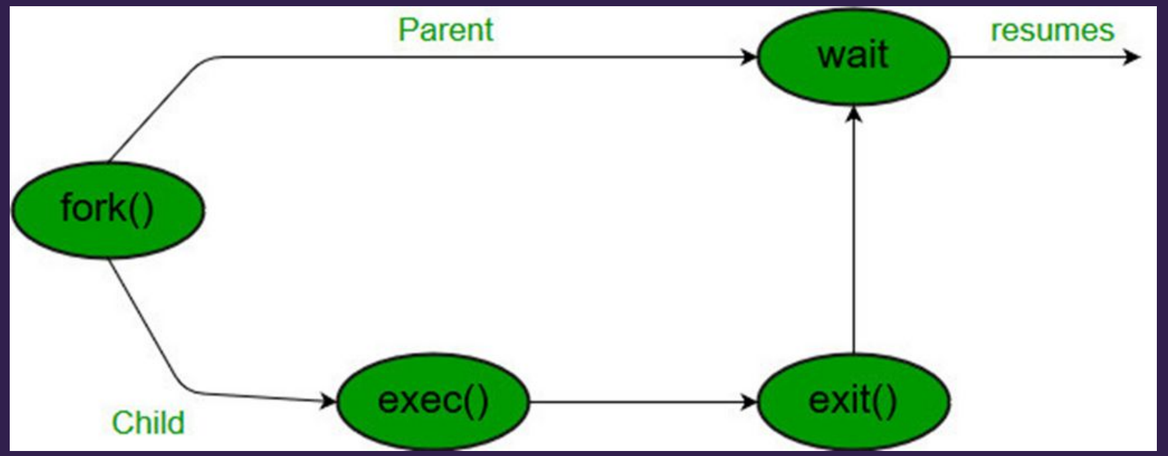
WAIT( )

La llamada al sistema wait() suspende la ejecución del hilo que lo llama hasta que uno de sus hijos termine.

Se utiliza un parámetro ‘status’ para comunicar al proceso padre la forma en que el proceso hijo termina. Por convenio, este valor suele ser 0 si el proceso termina correctamente y cualquier otro valor en caso de terminación anormal.

EXIT( )

Un programa utiliza la llamada al sistema exit() para finalizar su ejecución. El sistema operativo recupera recursos que fueron utilizados por el proceso después de la llamada al sistema exit().



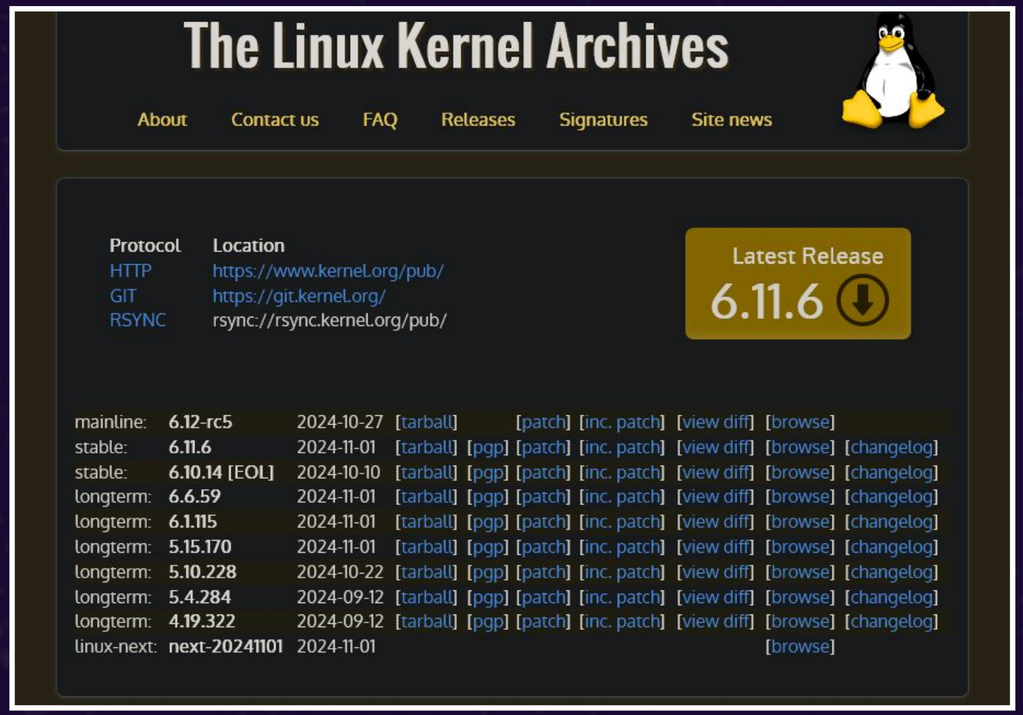
MODIFICANDO EL KERNEL

Obtener el Código Fuente del Kernel

- Comenzar descargando la versión más reciente del código fuente del kernel desde ***kernel.org***. Esto se puede hacer mediante comandos de terminal o desde el navegador. Una vez descargado, se debe descomprimir el archivo para acceder a los archivos de código fuente.

Instalar Dependencias Necesarias:

- Antes de compilar el kernel, es esencial contar con todas las herramientas necesarias, incluyendo compiladores y bibliotecas




```
> Instalar dependencias
$ sudo apt-get install build-essential libncurses-dev bison flex libssl-dev libelf-dev
```

Configurar el Kernel:

Se gestiona a través de un archivo .config. Para facilitar este proceso, se puede utilizar el comando localmodconfig, que ajusta automáticamente el archivo para incluir solo los módulos del kernel actualmente cargados en el sistema. Sin embargo, es importante destacar que esta configuración hará que el kernel compilado sea específico para la máquina utilizada.

Compilar el Kernel:

Usar el comando:
$ fakeroot make

fakeroot es una herramienta esencial que permite ejecutar comandos en un entorno simulado con permisos de superusuario, sin necesidad de tener realmente dichos permisos. Esto es particularmente útil al usar make, ya que facilita la creación de archivos con las propiedades y permisos de superusuario, simplificando así el proceso de construcción y empaquetado de software.

Instalar el Kernel:

Primero se instalan los módulos del kernel ejecutando:
$ sudo make modules_install

Luego instalamos el kernel:
$ sudo make install

Después de eso, reiniciamos la computadora para que se complete la instalación.

```
> Configurar el kernel
$ make localmodconfig

> Deshabilitar llaves privadas
$ scripts/config --disable SYSTEM_TRUSTED_KEYS
$ scripts/config --disable SYSTEM_REVOCATION_KEYS
$ scripts/config --set-str CONFIG_SYSTEM_TRUSTED_KEYS ""
$ scripts/config --set-str CONFIG_SYSTEM_REVOCATION_KEYS ""

> Compilar el kernel
$ fakeroot make -j4

> Instalar el kernel
$ sudo make modules_install
$ sudo make install
$ sudo reboot
```

Ejemplo Guiado:

Crear una nueva llamada al sistema

CONCEPTOS CLAVE APRENDIDOS

- Personalizar, compilar e instalar el kernel de linux
- Llamadas al Sistema
- Llamadas al sistema comunes

VALOR DE LA SEMANA

- **Pensamiento Crítico:** Es posible que como estudiante nos enfrentemos por primera vez al reto de explorar y modificar el código del kernel de linux, o bien a modificarlo tan a fondo, es importante que fortalecer el pensamiento crítico para este nuevo reto.

¡GRACIAS POR LA ATENCIÓN!

¿Dudas?