Taller de Sistemas Operativos

Llamadas al Sistema

Agenda

- Conceptos generales
- La API POSIX y las System Calls
- Syscalls
- System Call Handler
- Como implementar una System Call
- Contexto de una System Call
- Utilizar una System Call
- Conclusiones

Conceptos generales

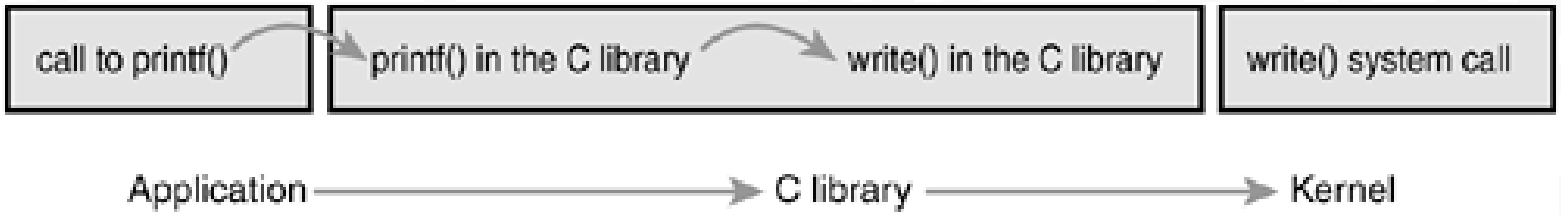
- El kernel provee un conjunto de interfaces para que procesos de usuario interactúen con él (System Calls)
  - Acceso a HW y otros recursos del SO
  - Actúan como mensajeros entre los procesos y el kernel
- Las System Calls tienen 3 propósitos principales
  - Abstraer el HW hacia el espacio de usuario
  - Al actuar el kernel como intermediario entre los recursos del sistema puede arbitrar en el acceso
  - Provee una capa de acceso única entre el espacio de usuario y el resto del sistema. Esto permite exponer un sistema “virtualizado” a los procesos

La API Posix y los System Calls

- Las aplicaciones son programadas sobre APIs y no directamente sobre system calls
- Una API define un conjunto interfaces usada por aplicaciones. Estas interfaces pueder ser programadas con 0 o más llamadas a system calls
- POSIX es una serie de estandares (IEEE) cuyo objetivo es proveer un estandar de SO portable
- POSIX es un excelente ejemplo de relación entre API y system calls
- En sistemas Unix las llamadas a funiones de la POSIX API tienen una fuerte relación con las system calls
- La Interface de system calls en Linux es provista en parte por la biblioteca de C

La API Posix y los System Calls

- La biblioteca C implementa el API principal en sistemas UNIX incluyendo la biblioteca estandar de C y la “system call interface”
- Puntos de vista
  - Para el programador el API es lo esencial
  - Para el kernel las system calls es todo



Syscalls

- Típicamente accedidas por *function calls*
  - Se definen 1 o más argumentos de entrada
  - Provocan efectos laterales
  - Pueden retornar un “long” que denota éxito o error
  - Si terminan con error modifican la variable `errno (perror())`
- Las *system calls* tienen un comportamiento definido
- El *kernel* debe proveer el comportamiento esperado sin importar la implementación siempre y cuando sea correcto

Syscalls

```
asmlinkage long sys_getpid(void)
{
return current->tgid;
}
```

- Observaciones
  - `asmlinkage` significa que el compilador debe buscar los parámetros solamente en el stack. Esto es requerido para las system calls.
  - La system call `getpid` se define como `sys_getpid` en el kernel. Esto es una convención de nombrado para todas las system calls en Linux.

Syscalls

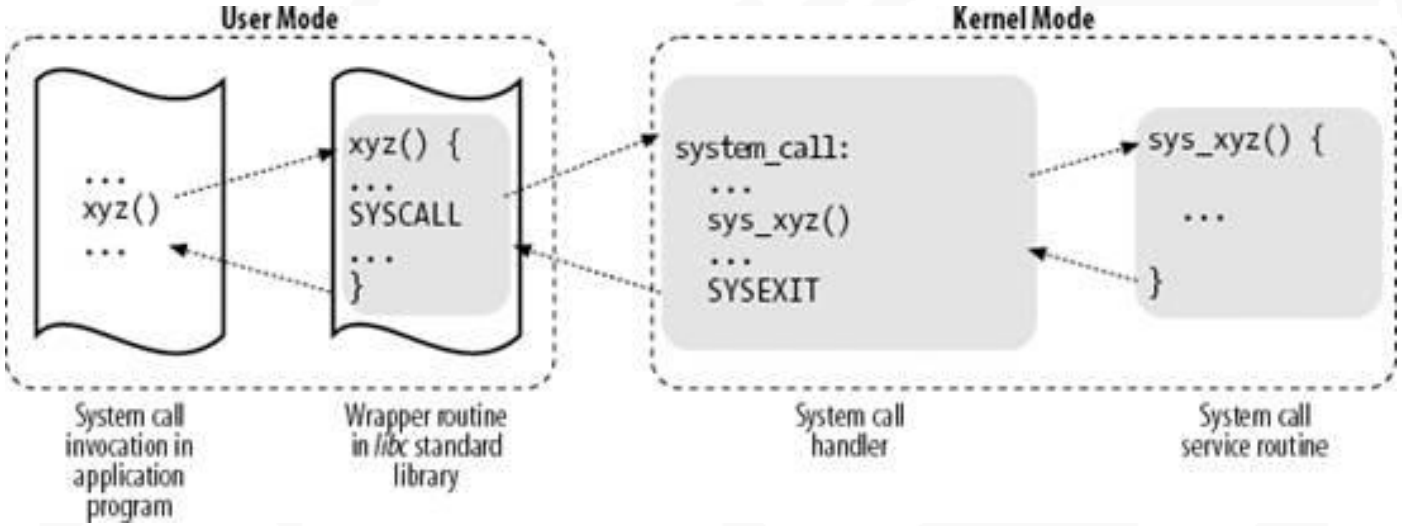
- En Linux a cada system call se le asigna un número único.
- Cuando un proceso de usuario quiere ejecutar un system call la identifica por ese número y no por su nombre.
- Si una system call es quitada del sistema, su número de system call no puede ser reciclado
- Linux provee una system call “not implemented” `sys_ni_syscall()` que no hace nada y retorna `-ENOSYS`
- El kernel tiene una lista con todas las system calls registradas llamada `sys_call_table` y definida en `syscall_32.tbl`
- Depende de la arquitectura y para x86 está en `arch/x86/syscalls/`

System call handler

- Para aplicaciones de espacio de usuario no es posible ejecutar código del kernel directamente (mem protegida)
- Las aplicaciones de espacio de usuario deben hacer un “signal” al kernel avisando que quieren ejecutar una Sys Call
- El mecanismo utilizado es una interrupción de software
- El manejador de la exepción en este caso es el sytem call handler. En x86 se dispara con la ejecución de `int $0x80`
- Dispara un switch a modo kernel y la ejecución del vector de interrupción 128 que es la función `system_call()`
- Es dependiente de la arquitectura y se define en `entry.S`

System call handler

- El número de system call debe ser pasado al kernel
- En x86, este número se pasa al kernel mediante el registro `eax`
- La función system_call () verifica la validez del número comparándolo con NR_syscalls. Si es mayor o igual, la función retorna –ENOSYS. De otra manera se llama la system call correspondiente



System call handler

- Las sycalls requieren 0 o más parámetros
- De alguna manera se deben pasar
- Se utilizan los registros del procesador
- En x86 los registros `ebx, ecx, edx, esi y edi` contienen los primeros cinco argumentos
- Si hay más se usa un solo registro con un puntero a user-space
- En x86 el para el valor de retorno se usa `eax`

Implementar una system call

- El primer paso es definir el propósito
- Definir argumentos, valor de retorno, códigos de error. Debe tener una interface simple con el menor nro de parámetros.
- Diseñe la system call para ser lo más gral posible
- Es la system call portable ?
- Es importante darse cuenta de la necesidad de portabilidad y robustez; no hoy sino para el futuro
- Las system calls básicas de Unix han sobrevivido por más de 30 años

Implementar una syscall

- Las syscalls deben verificar que los parámetros son válidos y legales. La seguridad y estabilidad del sistema están en juego
- Cada parámetro debe ser verificado para asegurarse que no sólo sea válido y legal sino correcto
- Un chequeo importante: verificar los punteros que el usuario provee. Se debe asegurar que:
  - El puntero apunta a una región en user-space
  - El puntero apunta a una región en el espacio del proceso
  - Si lee que la memoria esté habilitada para lectura; si escribe que esté habilitada para escritura
- El kernel provee funciones para hacer el check de los requisitos y copiar desde y hacia user-space

  `copy_to_user()` y `copy_from_user ()`

Contexto de una system call

- El kernel se encuentra en process context durante la ejecución de una syscall
- El puntero `current` apunta al proceso que ejecutó la syscall
- En contexto de proceso el kernel puede “dormirse” (si la syscall se bloquea) y es completamente preemptible
- El hecho que process context es preemptible implica que la “current task” puede ser preempted por otra task.
- La tarea nueva podría ejecutar la misma syscall, por lo que se debe asegurar que las syscalls sean reentrantes.
- Cuando la syscall retorna, el control continúa en system_call() quien cambia a user-space y continúa con la ejecución del proceso de usuario.

Registrar una system call

- Es trivial registrar una system call
  - Agregar una entrada al final de tabla de system calls para cada arquitectura soportada
  - Para cada arquitectura soportada, el número de system call debe ser definido en `<asm/syscalls.h>`
  - Si la system call existe en todas las arquitecturas se puede definir en `include/linux/syscalls.h`
  - La system call debe ser compilada dentro del kernel. Se debe poner en cualquier archivo relevante bajo `kernel/` como por ejemplo `sys.c`

Utilizar una system call

- Generalmente C provee mecanismos para llamar una syscall utilizando headers (.h) estandar y linkeditando con la biblioteca de C
- Linux provee una función para acceder a las syscalls
- Configura los registros e invoca las instrucciones de trap.
- Estas macros se llaman `syscall` y se encuentra en `unistd.h`
- Cada macro tiene como parámetros:
  - Número de la syscall
  - Parámetros de la system call en orden
- Retorna el `long` que retorna el system call

Conclusiones

- Ventajas de implementar una interface como syscall
  - Las system calls son fáciles de implementar
  - La performance de las system call en Linux es muy alta
- Las desventajas
  - Se necesita un número de syscall (asigando oficialmente) durante el desarrollo de una serie del kernel
  - Una vez que el kernel queda estable, ese número queda permanente
  - Cada arquitectura debe registrar ese número por separado y dar soporte a la system call
  - Las system calls no se pueden usar facilmente desde scripts ni desde el filesystem
  - Para un simple intercambio de información, una system call es demasiado

Conclusiones

- Las alternativas
  - Implementar un device node y manejarlo con `read()`, `write()` e `ioctl()`
  - Algunas interfaces como los semáforos pueden ser representadas como file descriptors y manipuladas de esa manera
  - Agregar la información como un archivo bajo `/proc`
- Para algunas interfaces, las system calls son la solución
- La baja tasa de agregado de nuevas system calls es una señal que Linux es un sistema operativo relativamente estable y completo en funcionalidades.