LABORATORIO 2

1 Introducción

En esta tarea se desea implementar un servicio sincronización de procesos a través de semáforos binarios con prioridad para el sistema operativo Linux. Para esto se deberá implementar un módulo a nivel del núcleo del sistema (dispositivo de caracteres "*char device*", so2011) y una biblioteca a nivel de usuario (*semSO*) que permitirán a los procesos sincronizarse.

El dispositivo de caracteres *so2011* administrará un conjunto de hasta 32 semáforos binarios. Los procesos a nivel de usuario podrán crear, eliminar y realizar las operaciones de sincronización. Para esto la biblioteca *semSO* brindará a los procesos una interfaz de acceso al servicio implementado a nivel del núcleo del sistema, de forma de simplificar la programación.

Los semáforos se identificarán del 0 al 31 (32 en total) y tendrán prioridades de para ser obtenidos por parte de los procesos de usuario. Las prioridades son un número de 1 a 5 donde el 1 representa la mayor prioridad y 5 la menor.

2 Componentes del sistema

▪ Biblioteca *semSO*

La biblioteca a nivel de usuario *semSO* (*semáforos Sistemas Operativos*) brinda las siguientes primitivas:

- ***semSO_init*** Inicializa estructura interna de la biblioteca.
- ***semSO_create*** Crea un semáforo binario y lo inicializa con un valor.
- ***semSO_destroy*** Destruye el semáforo.
- ***semSO_P*** Operación P.
- ***semSO_V*** Operación V.
- ***semSO_close*** Fin de uso de la biblioteca para el proceso.

Las especificaciones formales de las primitivas están en el Anexo I.
Un uso normal de la biblioteca por parte de un proceso sería invocar la siguiente secuencia de funciones: inicializar la estructura de la biblioteca a nivel del proceso (*semSO_init*), crear semáforos invocar rutinas de sincronización sobre los semáforos binarios (*semSO_P* y *semSO_V*), destruir semáforos (*semSO_destroy*) y, finalmente, cerrar el uso de la biblioteca a nivel de proceso (*semSO_close*).
Los 32 semáforos binarios se pueden comenzar a utilizar a partir de la carga del módulo (so2011), pero deben ser creados/inicializados por los procesos de usuario a través de la primitiva *semSO_create*.
Una vez que un semáforo binario es creado/inicializado por un proceso, este queda disponible para ser usado por cualquier proceso del sistema que inicialice la biblioteca (*semSO_init*).
Para destruir un semáforo que fue creado/inicializado previamente se utiliza la operación *semSO_destroy*.

La Figura 1 muestra un ejemplo de la utilización de la biblioteca *semSO* por parte de dos procesos. El ejemplo trata de mostrar un uso particular de la biblioteca donde un proceso crea/inicializa el semáforo y otro lo destruye. El semáforo utilizado para el ejemplo es el identificado con el número 12.

El proceso 2 no tiene necesidad de crear/inicializar el semáforo ya que esto fue realizado por el proceso 1.

```
// Código del proceso 1
#include <semSO.h>
#include ...
int main() {
    ...
    semSO_init();
    ...
    if (semSO_create(12,1) == -1) {
           printf("Error, el semáforo ya está creado o el servicio aún no
está disponible.\n");
           semSO_close();
           exit(-1);
     }
     ...
     semSO_P(12,5);
     // sección crítica
     semSO_V(12)
     ...
     semSO_close();
     return 0;
}
// Código del proceso 2
#include <semSO.h>
#include ...
int main() {
    ...
    semSO_init();
    ...
    semSO_P(12,5);
    // sección crítica
    semSO_V(12)
    ...
    semSO_destroy(12);
    semSO_close();
    return 0;
}
```

Figura 1 - Ejemplo de utilización de la biblioteca *semSO* por parte de dos procesos.

Notas:

- No existe restricción en el número de veces que un semáforo puede ser creado/inicializado y destruido.
- La invocación a la operación *semSO_create* sobre un semáforo que ya fue creado/inicializado debe retornar error.
- Los valores de inicialización de un semáforo serán 0 y 1.
- La ejecución de la operación *semSO_P* sobre un semáforo con valor 0 debe bloquear la ejecución del proceso.
- La ejecución de la operación *semSO_P* sobre un semáforo con valor 1 permite al proceso seguir ejecutando y el valor del semáforo cambia a 0 para bloquear a los demás procesos que hagan P sobre ese semáforo.
- La invocación de la operación *semSO_V* sobre un semáforo solo tiene efecto si el valor del semáforo es 0.
- En caso de que haya procesos bloqueados en un semáforo, la ejecución de la operación *semSO_V* sobre el semáforo debe permitir acceder al proceso con mayor prioridad de los bloqueados.

  A igual prioridad, se debe tomar un criterio FIFO (*First In – First Out*).
- La invocación de la operación *semSO_V* sobre un semáforo que no tenga procesos bloqueados debe solo cambiar el valor del semáforo a 1.
- La invocación de la operación *semSO_destroy* sobre un semáforo puede ser realizada por cualquier proceso del sistema.
- La invocación de la operación *semSO_destroy* sobre un semáforo que no está creado/inicializado no tiene efecto.
- La invocación de la operación *semSO_destroy* sobre un semáforo que tiene valor 0 debe retornar error.
- La invocación de una operación sobre un semáforo no creado/inicializado deber retornar error.
- El retorno de error de las operaciones es retornando el valor -1.

Dispositivo so2011

El módulo *so2011* será implementado como un dispositivo de caracteres (*char device*) Linux. Este dispositivo administrará unos 32 semáforos binarios.

Los dispositivos de caracteres son accedidos por el usuario a través del acceso a archivos definidos en el directorio `/dev` del sistema operativo. El acceso a estos archivos se realiza mediante los llamados a sistema de archivos (*system calls: open, read, write , close*). El desarrollo de un nuevo dispositivo de caracteres implica la implementación de un conjunto de primitivas definidas por el sistema operativo Linux. En este laboratorio se deberán implementar las siguientes primitivas requeridas por Linux para los dispositivos de caracteres:

- ***init*** Es invocada al cargar el módulo en el sistema operativo.
- ***exit*** Es invocada cuando el módulo es descargado del sistema operativo.
- ***open*** Es invocada cuando un proceso de usuario realiza el llamado a sistema *open*.
- ***write*** Es invocada cuando un proceso de usuario realiza el llamado a sistema *write*.
- ***read*** Es invocada cuando un proceso de usuario realiza el llamado a sistema *read*.
- ***release*** Es invocada cuando un proceso de usuario realiza el llamado a sistema *close*.

La comunicación con el módulo es a través de archivos en el directorio `/dev`. La biblioteca *semSO* implementará su funcionalidad a través de estos archivos. En la Figura 2 se muestra el acceso al dispositivo de caracteres abriendo el archivo `/dev/so2011`.

```
…
fd = open("/dev/so2011_3",O_RDWR);
…
If (write(fd,…))
…
```

Figura 2 - Ejemplo de acceso al módulo.

Notas:

- El módulo debe ser reentrante, por lo que se permitirá el uso de primitivas de sincronización como semáforos implementados a nivel del núcleo para permitir el correcto funcionamiento.
- **No está permitido el uso de semáforos para implementar el bloqueo de los procesos**. Para esto se debe utilizar el campo estado de los procesos (`TASK_INTERRUPTIBLE`, `TASK_RUNNING`).

  http://www.linuxjournal.com/article/8144

  http://book.opensourceproject.org.cn/kernel/kernelpri/opensource/0131181637/ch03lev1sec4.html
- No existe restricción de la cantidad de procesos que quieran operar sobre un semáforo.

3 Tarea

La tarea consiste en **diseñar, implementar y documentar la biblioteca *semSO* y el módulo del núcleo del sistema *so2011***, asegurando el correcto funcionamiento de los mismos.

4 Aclaraciones

- Se debe utilizar la primitiva *kmalloc* con la opción GFP_KERNEL para cualquier pedido de memoria dinámica que se realice en el módulo *so2011*.
  Ej: `… = kmalloc(sizeof(char) * TAMANO_BUFFER,GFP_KERNEL)`
- Las operaciones *open, read, write,* y *release* del módulo *so2011* deben retornar los valores correspondientes según la especificación de los llamados a sistema de Linux. Para esto deben utilizar el *man page* correspondiente a cada operación. Ej: *man 2 open*.

5 Consideraciones generales

5.1 Implementación

El trabajo debe realizarse utilizando el lenguaje de programación C y debe correr sobre el sistema operativo instalado en la máquina virtual distribuida al comienzo del curso (Linux Slackware 10.2.0).

- Puede utilizarse sintaxis o funciones de biblioteca de C++, pero no clases.
- Compile utilizando *make*, con un archivo *Makefile* (sin extensión). No se aceptarán entregas sin *Makefiles*, o que compilan con *scripts* o similares.
- Deberá compilarse con la opción *–Wall* (un parámetro de *g++*). Esta opción permite ver errores y advertencias (*warnings*) de compilación más detalladas.

5.2 Documentación

Debe entregarse toda la documentación impresa y en versión electrónica. En este caso se requiere el diseño de la solución tomada (las estructuras de datos, pseudocódigo, etc.).

No incluir la letra ni parte de ella en la documentación. Se penalizarán documentaciones que repitan los requerimientos o no describan lo implementado. Los códigos fuentes deben estar impresos al final de la documentación, en un anexo.

Debe especificarse en el sobre de la entrega el número de grupo e integrantes.

5.3 Plazo y Entrega

La fecha tope de entrega es el día **23 de Mayo de 2011 a las 11:30pm GMT-3** a través de la página web del curso.

De ninguna forma se aceptarán entregas fuera de la fecha de entrega estipulada.

5.4 Consultas

Se atenderán consultas en el newsgroup o en clase de consulta específicas para dicho fin.

5.5 Recomendaciones

- Utilizar los llamados a sistema (open, read, write y close) directamente para operar sobre los archivos *(/dev/so2011_XX*) y no otras librerías como *fopen*, *fclose*.
- Utilizar los semáforos del núcleo (*kernel semaphores*). Las operaciones *down_interruptible* y *up* corresponden a P y V respectivamente.
- Utilizar la función *printk* para hacer el *debug* del módulo. Se debe activar el *klog* agregando la siguiente línea '`kern.* /var/log/kern.log`' al archivo `/etc/syslog.conf`.

  Posteriormente, se debe reiniciar el proceso *syslog* mediante la ejecución del comando '`/etc/rc.d/rc.syslog restart`'. La función *printk* escribirá sobre el archivo `/var/log/kern.log`. Ej: `printk(KERN_INFO "so2011: cleanup\n");`.
- Al realizar pruebas con el módulo del kernel deberán respaldar la información ya que es común que obtengan el error "*Kernel panic*" y deban reiniciar la máquina virtual.

5.6 Condiciones del Laboratorio

El trabajo es obligatorio, la no entrega del trabajo o el mal funcionamiento implican la pérdida del mismo y por lo tanto la pérdida del curso.

Cada grupo debe realizar su trabajo en forma individual, y la detección de trabajos hechos en forma no individual (entre los grupos) implica la pérdida del mismo para los grupos involucrados. En este caso también implica la pérdida del curso para todos los participantes de los grupos.

Cada grupo es responsable de proteger su trabajo contra copia. No olvide diskettes/pendrives en la sala de máquinas, ni envíe correos o mensajes al newsgroup que permitan a otros copiar su código o parte de él.

Luego de finalizada la tarea, se realizará una prueba. Los detalles de la misma se informarán a través de la cartelera del curso y el grupo de noticias.

6 Anexo I

```
#ifndef SEMSO_H
#define SEMSO_H

/**
 *  Sistemas Operativos
 *  Curso 2011.
 *
 *  Biblioteca de semaforos binarios con prioridades.
 *
 **/

/**
 *  semSO_init
 *
 *  Incializa las estructuras internas de la biblioteca para el proceso.
 *  Parámetros:
 *     No tiene.
 *  Retorna:
 *   0 - Se inicializó la librería en forma correcta.
 *  -1 - Hubo un error.
 */
int semSO_init();

/**
 *  semSO_create
 *
 *  Crea/inicializa un semaforo binario.
 *  Parámetros:
 *  int – Identificacion del semaforo 0..31.
 *  int – Valor para inicialización (0 - bloqueado, 1 -Libre).
 *  Retorna:
 *  0 – La operacion culmino con exito.
 * -1 – Hubo un error.
*/
int semSO_create(int,int);

/**
 *  semSO_destroy
 *
 *  Destruye un semaforo binario.
 *  Parámetros:
 *  int – Identificacion del semaforo 0..31.
 *  Retorna:
 *  0 – La operacion culmino con exito.
 * -1 – Hubo un error.
*/
int semSO_destroy(int);
```

```
/**
 *  semSO_P
 *
 *  Operación P.
 *  Parámetros:
 *  int – Identificacion del semaforo 0..31.
 *  int – Prioridad 1..5.
 *  Retorna:
 *  0 – La operacion culmino con exito.
 * -1 – Hubo un error.
*/
int semSO_P(int,int);
/**
 *  semSO_V
 *
 *  Operación V.
 *  Parámetros:
 *  int – Identificacion del semaforo 0..31.
 *  Retorna:
 *  0 – La operacion culmino con exito.
 * -1 – Hubo un error.
*/
int semSO_V(int);

/**
 *  semSO_close
 *
 *  Finaliza el uso de la biblioteca.
*/
void semSO_close();


#endif
```

7 Bibliografía

- The Linux Kernel Module Programming Guide. http://www.tldp.org/LDP/lkmpg/2.4/html/lkmpg.html
- *Linux Device Drivers,* 2nd. Edition. Rubini A., Corbet J.June 2001. ISBN: 0-59600-008-1. Capítulos recomendados: 2, 3, 4 y 13.
- *Understanding the Linux Kernel*. 2nd. Edition. Bovet, P., Cesati, M.. December 2002. ISBN: 0-59600-213-8. Capítulos recomendados: 5, 7 y 8.
- Páginas del manual de Unix on-line. http://man.he.net/.
- Advanced Linux Programming, http://www.advancedlinuxprogramming.com.