Semáforos en Minix

M. Besio P. García H.Rajchert

22 de Noviembre de 2006

Resumen

En este trabajo expondremos la información y los resultados obtenidos al realizar una modificación al sistema operativo Minix.

1. Enunciado: Modificaciones a Minix

1.1. Objetivo

El proyecto a realizar consiste en agregar al Sistema Operativo Minix 2.0.0, una característica o herramienta, que dispongan los Sistemas Operativos más elaborados y que Minix no tenga. El proyecto está centrado en la investigación del funcionamiento de Minix, manejo de pasaje de mensajes, servidores, task (drivers), manejo de system calls e interrupciones. La primera parte consiste en el estudio de las posibles implementaciones a realizar, para luego comenzar con las modificaciones necesarias.

1.2. Temas propuestos

- Archivos ocultos: Se deben poder ocultar archivos, según el dueño, grupos o otros, una estructura similar a lectura-escritura-ejecución.
- Cola de mensajes: Debe cumplir con el estándar de POSIX.
- Semáforos: Debe cumplir con el estándar de POSIX.
- Mount de FAT 12 (read): Se debe poder hacer un mount de un diskete con FAT 12 y ver su contenido, pero no modificarlo.

Se aceptará incluir otros temas además de los aquí presentados. La propuesta de los alumnos debe ser aprobada por los integrantes de la cátedra en función de la complejidad del tema elegido y de la cantidad de alumnos que integren el grupo.

1.3. Contenido

1.3.1. Script de instalación y desinstalación

El grupo debe desarrollar un script que realice lo siguiente:
En la instalación:

- Backup de archivos fuentes que se van a modificar.
- Copiar archivos fuentes modificados.
- Compilar el kernel.

En la desinstalación:

- Restaurar los arhivos fuentes resguardados.
- Compilar el kernel.

El objetivo del script es no modificar la instalación de Minix existente.

1.3.2. Programas de prueba

Se deberán desarrollar y documentar debidamente los programas de testeo utilizados por el grupo, explicando en cada uno su función.

1.3.3. Páginas de manuales

Cada grupo deberá incluir en el sistema los manuales (programa man) de las nuevas funcionalidades que hayan implementado.

1.3.4. Informe

El informe debe contener las siguientes secciones como mínimo:

- Carátula.
- Enunciado del TP.
- Tema elegido.
- Listado de los fuentes de Minix modificados y los fuentes nuevos, comentando brevemente en cada caso la modificación realizada y el objetivo de la misma. (En el informe impreso, en tipo de letra "Bold" o "Negrita").
- Manual del usuario. Comandos. Forma de compilar.
- Programas de test. Explicación de su funcionamiento.
- Funcionamiento interno de Minix analizado.

Todos estos ítem serán evaluados.

1.4. Consideraciones

- Es posible utilizar en el trabajo rutinas o librerías ya desarrolladas consultando previamente al docente a cargo la inclusión de las mismas. No está permitido incluir rutinas o librerías desarrolladas por integrantes de otros grupos.
- El tipo de diseño y la forma de implementación serán discutidos entre el grupo y la cátedra durante las clases de laboratorio, dejando la posibilidad de modificar éste enunciado escrito, previo acuerdo entre el docente y los integrantes del grupo.
- Para la evaluación se tendrá en cuenta no sólo el funcionamiento del programa sino también el informe y la investigación realizada sobre el funcionamiento interno del Sistema operativo Minix.

1.5. Material a entregar

Cada grupo deberá entregar:

- 1 Diskette con los archivos fuentes y el informe del trabajo realizado (FAT 12).
- Informe impreso.

1.6. Integrantes del grupo

El trabajo debe ser realizado en grupos de 3 integrantes o menos. La nota definitiva se compone de un coloquio oral y demostración del trabajo funcionando en el que deben estar presentes, sin excepciones, todos los integrantes del grupo para su defensa el día de la entrega.

1.7. Fecha de entrega y defensa del trabajo final

Miércoles 22 de Noviembre a las 18 hs en el Laboratorio LI1.

2. Tema elegido

El tema que elegimos es semáforos, dado que Minix no tiene un soporte nativo de los mismos y proporcionan ventajas significativas cuando se trata con problemas de concurrencia. Lo implementamos bajo el estándar POSIX.

3. Cambios en los fuentes de Minix

Se agregó una biblioteca con las funciones que indica el estándar de POSIX para implementar semáforos. Puede obtener una lista de las mismas en la próxima sección. La misma fue incorporada a Minix como una biblioteca estándar. Se modificó el servidor de FS para agregarle un system call que invoca a estas funciones. Para lograrlo se agregó el número de la nueva system call en `callnr.h` y en `table.c` de `fs` se la implementó, mientras que en el de `mm` se colocó la contrapartida de la system call indicando que no se use.

Los archivos de Minix modificados o agregados son los siguientes:

- `fs/Makefile` - Cambiado para compilar, se agregaron entradas para compilar el semáforo y para sus dependencias.
- `fs/misc.c` - Agregada una función que captura el system call y llamadas a las funciones del semáforo que se deben llamar cuando los procesos mueren o hacen fork o exec.
- `fs/proto.h` - Agregados prototipos de funciones del semáforo que usa el fs pero no son de la biblioteca.
- `fs/main.c` - Agregada la inicialización de las funciones del semáforo en la inicialización del servidor.
- `fs/table.c` - Agregado el system call a la tabla de llamadas del fs.

- `mm/table.c` - Agregado el `no_sys` a la tabla de llamadas del mm (el mm no responde a esta llamada).
- `fs/my_param.h` - Archivo con macros para enmascarar los campos de la estructura de los mensajes.
- `fs/semaphore.c` - Implementación de las funciones del semáforo en el servidor.
- `fs/semaphore2.c` - Funciones del semáforo que abstraen a la implementación del semáforo del sistema en que corre e interactúan con Minix.
- `/usr/src/lib/libsem/semaphore.c` - Agregado el número de system call.
- `/usr/src/lib/libsem/syscall/sem_close.s` - System call para close.
- `/usr/src/lib/libsem/syscall/sem_destroy.s` - System call para destroy.
- `/usr/src/lib/libsem/syscall/sem_getvalue.s` - System call para getvalue.
- `/usr/src/lib/libsem/syscall/sem_init.s` - System call para init.
- `/usr/src/lib/libsem/syscall/sem_open.s` - System call para open.
- `/usr/src/lib/libsem/syscall/sem_post.s` - System call para post.
- `/usr/src/lib/libsem/syscall/sem_trywait.s` - System call para trywait.
- `/usr/src/lib/libsem/syscall/sem_wait.s` - System call para unlink.
- `/usr/include/minix/callnr.h` - Agregado el número de system call.
- `/usr/include/semaphore.h` - Prototipos del semáforo.
- `/usr/lib/i386/libsem.a` - Biblioteca de funciones del semáforo.

4. Manual del usuario

Para instalar esta modificación en Minix Ud. deberá ejecutar el script `tpminix.sh` seguido de las palabras clave `install` o `uninstall`. Con esto se modificará el sistema agregando la funcionalidad del trabajo.

Una vez instalada la modificación, usted podrá acceder al manual online de la biblioteca y cada una de las funciones implementadas por medio del comando `man`. Las funciones corresponden al estándar POSIX y son las siguientes:

- `int sem_close(sem_t *sem);`
- `int sem_destroy(sem_t *sem);`

- `int sem_getvalue(sem_t *sem, int *sval);`
- `int sem_init(sem_t *sem, int pshared, unsigned value);`
- `sem_t *sem_open(const char *name, int oflag, ...);`
- `int sem_post(sem_t *sem);`
- `int sem_trywait(sem_t *sem);`
- `int sem_unlink(const char *name);`
- `int sem_wait(sem_t *sem);`

5. Programas de prueba

El problema que resuelve la aplicación `prod_cons` es el clásico de consumidor-productor. Para ello se requieren dos o más procesos, al menos uno productor y otro consumidor. Los primeros producen mensajes y los segundos los consumen.

Dado que la implementación de la system call fue a destiempo, las funciones de la biblioteca en sí no podían ser probadas durante su producción misma (en Linux). Por lo tanto, se ha desarrollado una aplicación en Linux entera cliente-servidor TCP/IP que emula la IPC de Minix con pasaje de mensajes por sockets.

Para compilarla se debe acceder a `src/tests/linux` y ejecutar el script `compilar_ejemplos.sh`. Una vez completada la compilación se podrá iniciar el productor con `prod_cons p` y el consumidor con `prod_cons c`. Se pueden lanzar varios consumidores y varios productores a la vez (debe lanzarse desde varias terminales dado que el programa se bloquea esperando teclas). Una vez dentro, por cada caracter que se ingrese (incluido ENTER) se le estará indicando que consuma (o produzca, según sea el caso) un elemento. Por ejemplo, si se ingresa `aaa` en el consumidor, se consumirán (o se intentará consumir) 4 elementos (3 letras + ENTER). De esta manera podrá apreciarse cómo es la dinámica de los semáforos detrás de esta lógica, y probar su funcionamiento en Linux.

En Minix, el mismo programa se ejecuta nativamente y se encuentra en `/usr/ast/sem/`.

También se incluye otro programa (`pc_init`) que funciona de la misma manera que el anterior, sólo que no realiza `sem_open` y `sem_close` sino que realiza `sem_init` y `sem_destroy`.

6. Funcionamiento interno de Minix

Minix tiene una estructura interna dividida en cuatro niveles. En su base se encuentra el administrador de procesos, seguido de las tareas (drivers), los servidores y finalmente los procesos de usuario. La capa de administración de procesos se encarga de atrapar todas las interrupciones, alternar entre tareas y proveer al sistema del pasaje de mensajes entre procesos como forma de intercomunicación. La siguiente capa contiene los procesos que manejan la comunicación con los dispositivos de distintos tipos. Dentro de ellos se encuentra el System Task,

que si bien no interactúa con ningún dispositivo, existe para proveer servicios como copiar partes entre diferentes regiones de memoria. La capa 3, la de servicios, permite al nivel inferior de la cadena tener la asistencia necesaria para su correcto funcionamiento. Por ejemplo, aquí se ubica el MM (Memory Manager) que proporciona llamadas al sistema tan importantes como fork, exec, etc. En la última capa se ubican los procesos de usuario, como el shell, editores de texto, compiladores y programas escritos por el usuario.

Particularmente, en nuestro interés cayó el manejo de IPCs en Minix. Este cuenta con tres primitivas para enviar y recibir mensajes. Ellas son:

- `send(dest, &message)`
- `receive(source, &message)`
- `send_rec(src_dst, &message)`

Para implementar el semáforo, tomamos ventaja de la existencia de las system calls de Minix. Cuando un proceso en Minix realiza una system call, utiliza un wrapper de la función `send_rec`. Por motivos de seguridad, Minix no permite a los procesos de usuario realizar otro tipo de mensajería, ya que podría se que enviaran mensajes indiscriminadamente a los procesos de sistema, atareándolos inútilmente en forma abusiva. Esta implementación de semáforos, como decíamos, toma ventaja de el uso de esta función de envío-recepción porque bloquea al proceso esperando una respuesta a la llamada al sistema. Así, cuando un proceso entra en un semáforo y pide esperar en él, se ejecuta la llamada, se envía el mensaje al servidor y el proceso se bloquea en ese punto. Mientras tanto, el servidor, atiende el mensaje, y llama a la función correspondiente del semáforo, que según corresponda, enviará o no una respuesta al proceso invocador y así manejará el estado de bloqueo del mismo. La implementación de la llamada al sistema usa mensajes de tipo 2, e incluye en sus campos los valores que debe recibir la función correspondiente del semáforo. Además incluye un código que permite identificar la función del semáforo que se quiere invocar, lo que nos permite mejorar el diseño al no tener que agregar en la tabla de llamadas al sistema una entrada por cada función del semáforo, y en cambio poner un único punto de entrada al kernel para toda la biblioteca. El único parámetro que no viaja por mensajes es el nombre del semáforo, que es copiado desde el servidor con una función de copia de memoria del kernel. El uso de esta función nos permite limitar el largo del nombre a 255 caracteres, y evitar intentos de buffer overflow añadiendo seguridad al sistema. Luego de procesar el requerimiento del proceso de usuario dentro del servidor, se le responde através de un mensaje de tipo 1 con los valores de retorno de la función y valores que deben ser copiados por recibir parámetros por referencia.

Con respecto a nuestras pruebas, el desarrollo de la aplicación cliente-servidor no sólo resultó provechoso para detectar bugs de antemano, sino que sirvió para dejar muy en claro la arquitectura interna de microkernel de Minix. El mismo Tanenbaum aclara que la estructura de Minix puede pensarse como una aplicación de este tipo, dado

que el pasaje de mensajes es análogo a una transacción en TCP/IP.

A modo de prueba incluímos una consola que interpreta comandos posix de semáforos que utilizamos para durante el desarrollo para probar el funcionamiento interno de las distintas funciones del semáforo. Cuando se ejecuta el programa, este ofrece varias acciones, una por cada función posix, más una que libera el espacio interno. Hay una macro que define la cantidad máxima de semáforos a usar por ese proceso. Cada vez que se crea un semáforo con open o init se consume uno de los semáforos, en caso de quedarse el proceso sin semáforos (no es lo mismo a que el sistema se quede sin recursos para semáforo). Por cada función que requiere un `sem_t *` se le pide al usuario que diga cual es el semáforo con su nombre.

Cabe destacar que la biblioteca implementada no soporta la reentrancia. Si esta misma biblioteca se hubiera implementado para Linux debería haberse creado con soporte para la reentrancia, dado que allí se pueden producir interrupciones a la ejecución de la misma. En cambio, Minix entre niveles distintos no presenta problema dado que los procesos se comunican mediante colas de mensajes, y hasta que el nivel superior no termine de atender el pedido, no se pasará a otro.

7. Man Pages

7.1. Semaphore

```
semaphore(3)              Minix programmer's manual             semaphore(3)

NOMBRE
       semaphore - Biblioteca que implementa semáforos en Minix.

SINOPSIS
       #include <semaphore.h>

       int sem_close(sem_t *sem);

       int sem_destroy(sem_t *sem);

       int sem_getvalue(sem_t *sem, int *sval);

       int sem_init(sem_t *sem, int pshared, unsigned int value);

       sem_t *sem_open(const char *name, int oflag);

       sem_t *sem_open(const char *name, int oflag, mode_t mode, unsigned int value);

       int sem_post(sem_t *sem);

       int sem_trywait(sem_t *sem);

       int sem_unlink(const char *name);

       int sem_wait(sem_t *sem);

DESCRIPCION
       Las  funciones  de  semáforo  permiten  hacer  operaciones  seguras sobre recursos compartidos. Vea las
       páginas individuales para descripciones más detalladas sobre cada función. Un  semáforo  es  un  número
       entero que nunca toma valores negativos. Sobre él se pueden realizar 2 operaciones básicas: incrementar
       en uno (con la función sem_post) y decrementar en uno (con la función sem_wait). Si  el  semáforo  vale
       cero al llamar a sem_wait, el proceso se bloquea hasta que el semáforo incremente su valor.

       POSIX define dos tipos de semáforos: con y sin nombre.

       Los  semáforos  con nombre se pueden identificar con una cadena de la forma /nombre. Dos o más procesos
       pueden esperar en el mismo semáforo usando la misma cadena al llamar a la función sem_open.

VER TAMBIÉN
       sem_close (3), sem_destroy (3), sem_getvalue (3), sem_init (3), sem_open (3), sem_post (3),  sem_timed-
       wait (3), sem_trywait (3), sem_unlink (3), sem_wait (3)
```

7.2. Close

```
sem_close(3)              Minix programmer's manual             sem_close(3)

NOMBRE
       sem_close - cerrar un semáforo nombrado

SINOPSIS
       #include <semaphore.h>

       int sem_close(sem_t *sem);

DESCRIPCION
       La  función  sem_close()  indica que el proceso invocador ha terminado de utilizar el semáforo indicado
       por sem. Los efectos de llamar a sem_close() por un semáforo no nombrado (uno creado por sem_init()) no
       están  definidos.  La  función  sem_close()  liberará  (esto  es,  hará  disponible para reusarse en un
       sem_open() subsecuente de este proceso) cualquier recurso del sistema utilizados por el mismo por  este
       proceso  para este semáforo. El efecto de subsecuentes usos del semáforo indicado por sem por este pro-
       ceso no está definido. Si el semáforo no ha sido removido exitosamente por sem_unlink(), sem_close() no
       tendrá  efecto  sobre  el  estado  del  semáforo.  Si sem_unlink() se ha invocado exitosamente sobre el
       semáforo, cuando todos los procesos que lo hayan abierto lo cierren el semáforo dejará de  ser  accesi-
       ble.

VALOR DE RETORNO
       En  una  ejecución  exitosa  se devolverá el valor cero. En caso de error se devolverá -1 y la variable
       errno será actualizada para indicarlo.

CONFORME A
       POSIX.

VER TAMBIÉN
       sem_destroy (3), sem_getvalue (3),  sem_init  (3),  sem_open  (3),  sem_post  (3),  sem_timedwait  (3),
       sem_trywait (3), sem_unlink (3), sem_wait (3)

Versión 1.0                22 de Noviembre de 2006               sem_close(3)#
```

7.3. Destroy

```
sem_destroy(3)            Minix programmer's manual           sem_destroy(3)

NOMBRE
       sem_destroy - destruye un semáforo no nombrado

SINOPSIS
       #include <semaphore.h>

       int sem_destroy(sem_t *sem);

DESCRIPCION
       La  función  sem_destroy  destruirá  el  semáforo  indicado por sem. Sólo un semáforo creado utilizando
       sem_init puede ser destruido por sem_destroy; el efecto de llamar sem_destroy con un semáforo  no  nom-
       brado  es indefinido, al igual que el uso subsecuente del semáforo hasta que se lo re-inicialice nueva-
       mente.

       Es seguro destruir un semáforo inicializado sobre el cual no haya threads  actualmente  bloqueados.  El
       efecto de destruir un semáforo donde hayan threads bloqueados es indefinido.

VALOR DE RETORNO
       Ambas  funciones, en una ejecución exitosa se devolverá el valor cero. En caso de error se devolverá -1
       y la variable errno será actualizada para indicarlo.

CONFORME A
       POSIX.

VER TAMBIÉN
       sem_close (3), sem_getvalue (3), sem_init (3), sem_open (3), sem_post (3), sem_timedwait (3),  sem_try-
       wait (3), sem_unlink (3), sem_wait (3)

Versión 1.0                22 de Noviembre de 2006             sem_destroy(3)
```

7.4. Getvalue

```
sem_getvalue(3)           Minix programmer's manual          sem_getvalue(3)

NOMBRE
       sem_getvalue - obtener el valor de un semaforo

SINOPSIS
       #include <semaphore.h>
```

```
       int sem_getvalue(sem_t *sem, int *sval);

DESCRIPCION
       La  función  sem_getvalue() actualizará el contenido apuntado por sval para tener el valor del semáforo
       referenciado por sem sin afectar el estado del mismo. El valor representa el valor actual del  semáforo
       presente  en  algún momento no especificado durante la llamada, y no necesariamente el valor actual del
       semáforo al producirse la devolución del valor al proceso que invoca.

       Si sem está bloqueado, sval apuntará a un número negativo cuyo módulo indicará el  número  de  procesos
       esperando al semáforo en un momento no especificado durante la llamada.

CONFORME A
       POSIX.

VALOR DE RETORNO
       En  una  ejecución  exitosa  se devolverá el valor cero. En caso de error se devolverá -1 y la variable
       errno será actualizada para indicarlo.

VER TAMBIÉN
       sem_close (3), sem_destroy (3), sem_init (3), sem_open (3), sem_post (3), sem_timedwait  (3),  sem_try-
       wait (3), sem_unlink (3), sem_wait (3)

Versión 1.0                22 de Noviembre de 2006            sem_getvalue(3)
```

7.5. Init

```
sem_init(3)               Minix programmer's manual               sem_init(3)

NOMBRE
       sem_init - inicializar un semáforo no nombrado

SINOPSIS
       #include <semaphore.h>

       int sem_init(sem_t *sem, int pshared, unsigned value);

DESCRIPCION
       La  función  sem_init inicializa el semáforo sin nombre referido por sem, utilizando como valor inicial
       value. Luego de una exitosa invocación de sem_init, el semáforo podrá ser  utilizado  en  subsiguientes
       llamadas a sem_wait, sem_trywait, sem_post y sem_destroy.

       Si  el argumento pshared tiene un valor que no sea cero, el semáforo será compartido entre procesos. En
       este caso, cualquier proceso que pueda acceder al semáforo  sem  podrá  utilizarlo  para  realizar  las
       operaciones  antes  mencionadas.  Sólo  sem  podrá  ser utilizado para realizar las operaciones de sin-
       cronización. El resultado de referir a copias de sem en llamadas a esas funciones es indefinido.

       Si el argumento pshared es cero, el semáforo será compartido entre hilos del proceso,  los  que  podrán
       efectuar cualquiera de las operaciones que figuran más arriba.

       El intentar inicializar un semáforo ya inicializado resulta en un comportamiento no definido.

VALOR DE RETORNO
       En  una  ejecución  exitosa  se devolverá el valor cero. En caso de error se devolverá -1 y la variable
       errno será actualizada para indicarlo.

CONFORME A
       POSIX.

VER TAMBIÉN
       sem_close (3), sem_destroy (3), sem_getvalue  (3),  sem_open  (3),  sem_post  (3),  sem_timedwait  (3),
       sem_trywait (3), sem_unlink (3), sem_wait (3)

Versión 1.0                22 de Noviembre de 2006               sem_init(3)
```

7.6. Open

```
sem_close(3)              Minix programmer's manual              sem_close(3)

NOMBRE
       sem_open - inicializar y abrir un semáforo nombrado

SINOPSIS
       #include <semaphore.h>

       sem_t *sem_open(const char *name, int oflag);

       sem_t *sem_open(const char *name, int oflag, mode_t mode, unsigned int value);

DESCRIPCION
       La  función sem_open() crea un nuevo semáforo POSIX o abre uno existente. Este semáforo es identificado
```

```
       por nombre.

       El argumento oflag especifica los flags que controlan la operación de la llamada. Si O_CREAT está
       especificado en oflag, el semáforo es creado si no existía previamente. Si O_CREAT y O_EXCL están
       especificados en oflag, se devuelve un error si el semáforo con el nombre dado ya existe.

       Si O_CREAT está definido en oflag, se deben proporcionar dos argumentos adicionales. El argumento mode
       especifica los permisos asociados al nuevo semáforo, de la misma manera en que lo hace open(2). El
       argumento value indica el valor inicial para el nuevo semáforo. Si se define O_CREAT y un semáforo con
       el nombre dado ya existe, los argumentos mode y value son ignorados.

VALOR DE RETORNO
       En una ejecución exitosa se devolverá la dirección del nuevo semáforo para facilitar la invocación de
       las demás funciones con relación a semáforos. En caso de error se devolverá SEM_FAILED y la variable
       errno será actualizada para indicarlo.

CONFORME A
       POSIX.

VER TAMBIÉN
       sem_close (3), sem_destroy (3), sem_getvalue (3), sem_init (3), sem_post (3), sem_timedwait (3),
       sem_trywait (3), sem_unlink (3), sem_wait (3)

Versión 1.0                22 de Noviembre de 2006              sem_close(3)
```

7.7. Post

```
sem_post(3)               Minix programmer's manual               sem_post(3)

NOMBRE
       sem_post - desbloquear un semáforo

SINOPSIS
       #include <semaphore.h>

       int sem_post(sem_t *sem);

DESCRIPCION
       La función sem_post() incrementa (desbloquea) el semáforo apuntado por sem. Si en consecuencia el valor
       del semáforo se hace mayor que cero, otro proceso o hilo bloqueado en un sem_wait(3) será despertado y
       procederá a bloquear el semáforo.

VALOR DE RETORNO
       En una ejecución exitosa se devolverá el valor cero. En caso de error no se alterará el semáforo, se
       devolverá -1 y la variable errno será actualizada para indicarlo.

CONFORME A
       POSIX.

VER TAMBIÉN
       sem_close (3), sem_destroy (3), sem_getvalue (3), sem_init (3), sem_open (3), sem_timedwait (3),
       sem_trywait (3), sem_unlink (3), sem_wait (3)

Versión 1.0                22 de Noviembre de 2006               sem_post(3)
```

7.8. Wait y Trywait

```
sem_wait, sem_trywait(3)   Minix programmer's manual  sem_wait, sem_trywait(3)

NOMBRE
       sem_wait, sem_trywait - desbloquear un semáforo

SINOPSIS
       #include <semaphore.h>

       int sem_trywait(sem_t *sem);

       int sem_wait(sem_t *sem);

DESCRIPCION
       La función sem_trywait() desbloquea el semáforo referenciado por sem sólo si el mismo no está actual-
       mente bloqueado. Si estuviera bloqueado no lo desbloquea.

       sem_wait() bloqueará el semáforo referenciado por sem. Si el valor del mismo es cero, el hilo o proceso
       que invoque la función no volverá de la función hasta que el semáforo sea desbloqueado. En tal caso
       volverá a bloquearse pero pasará el control a esta función.

VALOR DE RETORNO
       Ambas funciones, en una ejecución exitosa devolverán el valor cero. En caso de error no se alterará el
       semáforo, se devolverá -1 y la variable errno será actualizada para indicarlo.
```

```
CONFORME A
       POSIX.

VER TAMBIÉN
       sem_close  (3), sem_destroy (3), sem_getvalue (3), sem_init (3), sem_open (3), sem_post (3), sem_timed-
       wait (3), sem_unlink (3)

Versión 1.0                22 de Noviembre de 2006   sem_wait, sem_trywait(3)
```

7.9. Unlink

```
sem_unlink(3)              Minix programmer's manual             sem_unlink(3)

NOMBRE
       sem_unlink - remover un semáforo nombrado

SINOPSIS
       #include <semaphore.h>

       int sem_unlink(const char *name);

DESCRIPCION
       La  función sem_unlink removerá el semáforo correspondiente al nombre name. Si el semáforo está actual-
       mente referenciado por otro proceso, sem_unlink marcará el semáforo para posponer su destrucción  hasta
       que  todas las referencias al mismo hayan sido destruidas (sem_close). Las llamadas a sem_open para re-
       crear o re-conectarse al semáforo se referirán a un nuevo semáforo luego de que  sem_unlink  haya  sido
       llamada.

VALOR DE RETORNO
       En  una ejecución exitosa se devolverá el valor cero. En caso de error el semáforo no se modificará, se
       devolverá -1 y la variable errno será actualizada para indicarlo.

CONFORME A
       POSIX.

VER TAMBIÉN
       sem_close (3), sem_destroy (3), sem_getvalue (3), sem_init (3), sem_open (3), sem_post (3),  sem_timed-
       wait (3), sem_trywait (3), sem_wait (3)

Versión 1.0                22 de Noviembre de 2006             sem_unlink(3)
```

Referencias

[1] "Sistemas Operativos: Diseño e implementación, Segunda Edición", Tanenbaum & Woodhull, Ed. Pearson, 1998, New Jersey.

[2] http://www.opengroup.org/onlinepubs/000095399/ basedefs/semaphore.h.html

[3] http://www.midnightsoret.com.ar/personales/ alejandrovaldez/minix/SystemCall.html

Índice

Made with LaTeX