ÍNDICE

1. USO DE SEMÁFOROS IPC EN PROGRAMACIÓN DISTRIBUIDA

1.1. Introducción

Se pueden definir operaciones atómicas como aquellas instrucciones individuales que no son interrumpidas, es decir el proceso que las ejecuta no puede ser desplanificado de la CPU, sin haberlas ejecutado.

Una sección crítica es un conjunto de instrucciones de código que debe ejecutarse sin ser interrumpidas, al menos sin que otro proceso interfiera en el resultado. Un ejemplo de una sección crítica puede ser la actualización del valor de una variable en memoria compartida.

Una simple operación como: "contador ++;" no es atómica desde el punto de vista de la CPU. Esta instrucción podría ser descompuesta en otras que sí son atómicas:

(1) acumulador = contador
(2) acumulador = acumulador + 1
(3) contador = acumulador

Si "contador" es una variable compartida por dos procesos y ambos ejecutan la instrucción "contador ++;" a la vez, se debe garantizar que cualquiera que sea el orden en que se planifiquen los procesos para ejecutar las correspondientes instrucciones atómicas que actualizan la variable, el resultado final debe ser el mismo, en caso contrario nos encontraríamos ante una sección crítica. Veamos que sucede:

P1	P2	acumulador	Valor de contador
–	–	–	3
(1)	–	acumulador = contador = 3	3
(2)	–	acumulador +1 = 4	3
–	(1)	acumulador = contador = 3	3
–	(2)	acumulador +1 = 4	3
(3)	–	4	contador = acumulador = 4
–	(3)	4	contador = acumulador = 4
–	–	–	4

Si inicialmente "contador" tiene el valor '3', cuando los dos procesos acaben de incrementarla debería tener el valor '5'. Sin embargo, en el ejemplo propuesto, el valor final de "contador" es '4'.

Por tanto es necesario un mecanismo que permita el acceso a secciones críticas de forma segura.

Un semáforo es un mecanismo estándar propuesto por Dijisktra y diseñado para prevenir el acceso simultáneo por más de un proceso a recursos compartidos como puede ser zonas de memoria compartida, es decir el acceso a una sección crítica. Los semáforos puede ser de dos tipos:

- **Binarios**: puede tomar valores '0' o '1'. Cuando el semáforo tiene valor '0', el recurso asociado está siendo utilizado por un proceso, mientras que si toma el valor '1' dicho recurso se encuentra disponible para los procesos que lo comparten.
- **De cuenta**: El valor del semáforo podrá ser '0' o un número entero positivo. Mientras que el valor sea '0' ningún proceso puede acceder al recurso o sección crítica asociada y cuando tenga un valor entero positivo los procesos sí podrían acceder al recurso.

Las operaciones básicas que se definen sobre un semáforo son:

- **wait**: el proceso que realiza esta operación comprueba el valor del semáforo:
  - ✓ Si es igual a '0': se suspende el proceso y se coloca en la cola de espera del semáforo.
  - ✓ Si es mayor que '0': el valor se decrementa y el proceso puede entrar en la sección crítica.
- **signal**: se comprueba la cola de espera del semáforo:
  - ✓ Si es está vacía: se incrementa el valor del semáforo.
  - ✓ Si no está vacía, es decir existen procesos suspendidos en ella, se despierta a uno de ellos.

La implementación de semáforos no puede realizarse exclusivamente con código de usuario, sino que se apoya directamente en el hardware de la máquina (instrucciones máquina especiales de inhabilitación de interrupciones).

Con el uso de semáforos se ha de tener en cuenta el interbloqueo entre procesos. Dos procesos están bloqueados esperando cada uno por un recurso que el otro tiene bloqueado. Es responsabilidad del programador ubicar de forma adecuada las operaciones wait y signal en las secciones críticas de su código para evitar los interbloqueos.

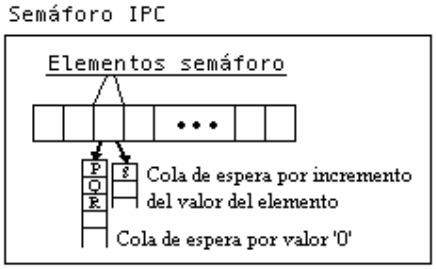
1.2. Semáforos IPC. Principales características

Semáforo IPC o semáforo del kernel es un mecanismo IPC para intercomunicación de procesos, además de colas de mensajes y memoria compartida.

Son más complejos que los conceptos básicos comentados en el apartado de introducción por las siguientes razones:

- Se puede crear un conjunto de semáforos con una simple llamada al sistema.
- Son semáforos multivaluados, como los semáforos de cuenta: con esta característica se permitirá restringir el acceso a un recurso de forma simultánea a un número máximo de procesos. También, el valor del semáforo puede indicar el número de unidades disponibles del recurso en cada momento. Pueden trabajar como semáforos binarios si el valor del semáforo igual a '1' indica que el recurso está libre y '0' que está ocupado.
- Es posible especificar un array de operaciones sobre un conjunto de semáforos.
- Se proporcionan facilidades que permiten automáticamente desbloquear todos los procesos bloqueados en los semáforos.
- Los semáforos presentan propiedades, modos de acceso y otro tipo de información similar al resto de mecanismos IPC, es decir a las colas de mensajes y memoria compartida.

Cuando se habla de un semáforo IPC realmente se trata de un grupo de semáforos, donde cada elemento semáforo consiste en un número entero.



Los semáforos IPC tienen una representación interna a la que el usuario no tiene acceso directamente. Para cada elemento semáforo se incluye al menos la siguiente información:

- Número entero que representa el valor del elemento semáforo.
- Identificador del último proceso que accedió a dicho elemento semáforo.
- Número de procesos que están esperando a que se incremente el valor del elemento semáforo.
- Número de procesos que esperan a que el valor del elemento semáforo sea '0'.

Existen dos colas en las que los procesos se encuentran bloqueados:

- Una para los procesos bloqueados a la espera de que se incremente el valor del elemento semáforo, ya que esperarán a que el valor sea un entero positivo.
- Otra para los procesos que están bloqueados esperando que el valor del elemento semáforo sea '0'.

En la siguiente tabla se muestran los límites del sistema que afectan a los semáforos IPC. Dichos valores dependen de la implementación del S.O. y los que mostramos aquí son los correspondientes a la máquina centauro (OSF/Digital Unix v4.0E).

Nombre	Descripción	Valor
SEMVMX	Máximo valor de cualquier semáforo.	32767
MSGMNI	El máximo número de semáforos IPC en todo el sistema	10
SEMMNS	El máximo número de elementos semáforo en todo el sistema.	60
SEMMSL	El máximo número de elementos semáforo por semáforos IPC.	25
SEMOPM	El máximo número de operaciones por llamada a semop	10

1.3. Operaciones sobre semáforos IPC

Como se puede observar con las características de los semáforos IPC, éstos proporcionan un mecanismo que en ocasiones es mucho más complejo de lo que, en general, se va a necesitar. Por esto, las operaciones se van a ver sobre semáforos IPC son un resumen de las disponibles.

Para más información se recomienda consultar las páginas del man.

1.3.1. Creación de un semáforo IPC: semget()

Esta es la primera función que se invoca normalmente para crear un semáforo IPC e inicializar el valor de cada elemento semáforo '0' o utilizar un semáforo IPC ya existente. La sintaxis de esta función es la siguiente:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int semget (key_t key, int nsems, int semflg);
```

Donde:

Argumento	Descripción
key	Identificador numérico para el semáforo IPC. Debe ser un valor entero distinto de '0' o la constante simbólica **IPC_PRIVATE**
nsems	Número de elementos semáforo. En funciones posteriores, los elementos semáforo serán referenciados por números enteros entre 0 y nsems-1.
semflg	**IPC_CREAT | PERMISOS**: flags para crear el semáforo IPC y asignar los permisos adecuados de acceso, en notación octal, para las tres categorías existentes en el entorno Unix (ugo, usuario-grupo-otros). Para combinar los permisos de creación del semáforo con el resto de flags se utiliza la operación OR a nivel de bits. **0**: Para el acceso a un conjunto ya creado anteriormente por otro proceso.

Si esta función tiene éxito devolverá el handler del nuevo semáforo IPC creado o accedido y será el valor utilizado para referenciarlo en el resto de funciones. En caso de error, devolverá –1.

La función puede fallar debido a que no existan suficientes recursos en el sistema que se puedan asignar o porque alguno de los argumentos no sea correcto.

En el caso de haber especificado el flag **IPC_CREAT** e **IPC_EXCL** y estar otro proceso utilizando esa clave, la llamada **semget()** fallaría y devolvería un error.

Hay que tener en cuenta que los semáforos seguirán existiendo aún después de que el proceso que los cree finaliza, obviamente si no los destruye antes.

El siguiente código es un ejemplo de creación de un conjunto de semáforos de dos elementos con clave 6043 y permisos de lectura y escritura para el propietario y lectura tanto para el grupo del propietario como para el resto.

```
#define CLAVE 6043
#define PERMISOS 0644
#define SIZE 2

…
semid = semget(CLAVE, SIZE, PERMISOS | IPC_CREAT);
...
```

El siguiente ejemplo muestra el acceso a un conjunto de semáforos de dos elementos con clave 6043 desde un proceso distinto al que lo crea.

```
#define CLAVE 6043

...
semid = semget(CLAVE,0);
...
```

1.3.2. Conjunto de operaciones sobre un semáforo IPC: semop()

Mediante esta función se podrán realizar varias operaciones para incrementar o decrementar el valor de varios elementos de un conjunto de semáforos. La sintaxis de esta función es la siguiente:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semop (int semid, struct sembuf * sops, int nsops);
```

Donde:

Argumento	Descripción
semid	Identificador del conjunto de semáforos

sops	Apunta a un conjunto de operaciones atómicas que se realizan sobre el grupo de semáforos. Cada operación está representada por la siguiente estructura:
	`struct sembuf {`
	`short sem_num;`
	`short sem_op;`
	`short sem_flg;`
	`};`
	• **sem_num**: número de elemento del conjunto sobre el que actuará la operación.
	• **sem_op**: operación particular que se va a realizar sobre el elemento. En este campo se pueden dar los siguentes casos:
	✓ Número positivo: la operación es sumar dicho número al valor del correspondiente elemento del semáforo. Además se despiertan todos los procesos que están esperando porque el elemento sea incrementado.
	✓ Número igual a '0': Si el elemento del semáforo no tiene valor '0', el proceso se bloquea hasta que se tenga dicho valor.
	✓ Número negativo: la operación es sumar al valor del elemento del semáforo correspondiente dicho número siempre que el resultado no sea negativo. Si la operación da como resultado un valor negativo el proceso se bloquea hasta que elemento semáforo se incremente. Si el valor resultante es '0', se despiertan todos los procesos que esperan por el valor '0' de dicho elemento.
	• **sem_flg**: opciones de la operación. Por defecto este valor es '0'. Si es **IPC_NOWAIT** el proceso nunca se bloquea, y si no puede realizar la operación la función devuelve –1, con **errno** con valor **EAGAIN**.

nsops	Número de operaciones del array

Si cualquiera de las operaciones que se indican en la función hace que el proceso se bloquee, ninguna de las operaciones se ejecutará.

En el siguiente ejemplo se muestra cómo implementar las funciones wait (Lock) y signal (Unlock) sobre un elemento semáforo binario de un semáforo IPC.

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

#define CLAVE 6043
#define PERMISOS 0644
#define SIZE 2

// realiza una operación wait sobre el elemento semáforo n del semáforo IPC sem
void Lock (int sem, int n)
{
  struct sembuf sop;            // struct para una operación
  sop.sem_num = n;              // la operación se aplicará sobre el elemento n
  sop.sem_op = -1;              // operación de decremento de una unidad
  sop.sem_flg = 0;              // Se bloquea hasta que el recurso esté libre
  semop (sem, &sop, 1); // Llamada a semop().
}

// realiza una operación signal sobre el elemento semáforo n del semáforo IPC sem
void Unlock (int sem, int n)
{
  struct sembuf sop;            // struct para una operación
  sop.sem_num = n;              // se aplicará la operación sobre el elemento n
  sop.sem_op = 1;               // operación de incremento en una unidad
  sop.sem_flg = 0;
  semop (sem, &sop, 1); // Llamada a semop ()
}
…
int semid;
semid = semget(CLAVE, SIZE, PERMISOS | IPC_CREAT);
for (int i = 0; i<SIZE; i++)
  Unlock (semid, i);    // Necesario desbloquear el recurso.
                // Inicialmente los elementos tienen valor 0
...
```

1.3.3. Operaciones de control de un conjunto de semáforos: semctl()

Esta función nos va a permitir realizar operaciones de control de los semáforos, entre las que se va a destacar su eliminación y la asignación de valores iniciales a un grupo de semáforos o de forma individual a un elemento del conjunto. Su sintaxis es la siguiente:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semctl (int semid, int semnum, int cmd, /* union semun arg*/ …);
```

Donde:

Argumento	Descripción
semid	Identificador del grupo de semáforos
semnum	Indica el elemento del semáforo dentro del grupo cuando el argumento **cmd** sea un comando referido a un elemento individual
cmd	Comando a ejecutar. Entre los valores que puede tomar destacan: • SETVAL: Inicializa el valor específico de un elemento al valor de **arg.val** • GETVAL: Devuelve el valor del elemento del conjunto de semáforos. • SETALL: Asigna los valores de **arg.array** a los elementos del conjunto de semáforos. (Utilizar esta función y este flag es más eficiente que asignar uno a uno los valores a los elementos de un grupo de semáforos). • GETALL: Devuelve todos los valores de los elementos del conjunto de semáforos en **arg.array** • IPC_RMID: Elimina el conjunto de semáforos identificado por **semid** y se libera el

Argumento	Descripción
	espacio de memoria que el kernel utiliza para él. Esta operación esta restringida, es decir, el usuario efectivo del proceso que la invoca debe de ser el root o bien ser el propietario o el creador del semáforo.
Arg	Argumento opcional que dependerá del valor de **cmd** y será usado de diferentes formas. El tipo de datos es:

```
union semun {
int val;                // valor del semáforo si cmd es SETVAL
struct semid_ds * buf;
ushort * array;         // puntero a un array de valores.
                        // Si cmd es SETALL o GETALL
};
```

La función, en general, si tuvo éxito devuelve '0', excepto si **cmd** es **GETVAL** en cuyo caso devuelve el valor del elemento del conjunto de semáforos.

En el siguiente ejemplo se muestra cómo implementar la función que asigne el valor inicial a un elemento semáforo (InicializaElemento) de un semáforo IPC y cómo implementar la función que asigna valores a todos los elementos semáforo (InicializaSemaforo) de un semáforo IPC a la vez.

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

#define CLAVE 6043
#define PERMISOS 0644
#define SIZE 2

// realiza la operación de inicialización sobre el elemento semáforo n del semáforo IPC sem
int InicializaElemento (int sem, int n, int valor)
{
 union semun arg;              // union para los argumentos de la operación
 if (valor < 0)
        return -1;
 arg.val = valor;              // El elemento n se inicializará a valor
 semctl (sem, n, SETVAL, arg); // Llamada a semctl().
 return 0;
}

// realiza la operación de inicialización sobre todos los elementos semáforo n del semáforo
IPC sem
int InicializaSemaforo (int sem, int valores [], int nval)
{
 union semun arg;              // union para los argumentos de la operación
 int i;
 for (int i = 0; i < nval; i++)
        if (valores [i] < 0)
                return -1;
 arg->array = valores;         // El array de valores está en elemento n se inicializará a
valor
 semctl (sem, 0, SETALL, arg); // Llamada a semctl().
 return 0;
}

…
int semid1, semid2;
int ValoresIniciales [] = {1, 1};

semid1 = semget(CLAVE, SIZE, PERMISOS | IPC_CREAT);
for (int i = 0; i<SIZE; i++)
  InicializaElemento (semid1, i, 1);   // Inicializar cada elemento semáforo a 1.

semid2 = semget(CLAVE, SIZE, PERMISOS | IPC_CREAT);
InicializaSemaforo (semid2, ValoresIniciales, SIZE); // Inicializar todos los elementos.

...
```

1.4. Semáforos y señales

Podemos definir una señal como una notificación de un evento UNIX.

Durante la ejecución de llamadas al sistema, la acción de la señal dependerá de las acciones específicas establecidas ante la llegada de la señal (bloquear, atrapar, ignorar). Con semáforos, se puede salir de una operación "wait" sin que se hubiera decrementado el semáforo: si **semop()** es interrumpida por una señal, devuelve –1 y **errno** vale **EINTR**. Para evitar que esto pueda suceder, cada vez que realicemos una llamada a **semop()** es necesario comprobar cuál es el valor de retorno de la función.

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <errno.h>


int semop_restart (int semid, struct sembuf * sops, int nsops)
{
  int retval;

  while ( ((retval = semop (semid, sops, nsops)) == -1) && (errno == EINTR) ) ;

  return retval;
}
```

1.5. Gestión de los semáforos desde la línea de comando

En Unix el comando **ipcs** puede utilizarse para obtener estadísticas de funcionamiento de varios mecanismos IPC entre los que se encuentran los semáforos. Mediante este comando y con la opción **–s** se podrá observar el estado de los semáforos del kernel. Así mismo, las máquinas Unix disponen de otro comando, **ipcrm**, que puede utilizarse para eliminar semáforos IPC, con la opción **–s** y donde se puede indicar el identificador de un semáforo concreto.

Para más información, consultar las páginas del man.

ANEXO A

```
/* El fichero cabecera line.h */
struct info {
char c;
int length;
};
union semun {
int val;
struct semid_ds *buf;
u_short *array;
};

#define KEYSHMEM        ((key_t) (1234))
#define KEYSEM  ((key_t) (4321))
#define SEGSIZE sizeof(struct info)
#define SEMSIZE 1

void Lock (int, int); // Funcion que realiza wait sobre un elemento del semaforo IPC
void Unlock (int, int);// Funcion que realiza signal sobre un elemento del semaforo IPC
int InicializaElemento (int, int, int);  // Funcion que inicializa un elemento un semaforo
IPC a un valor
```

```
/* cline.c
  Este es un ejemplo de uso de memoria compartida y semaforos IPC.
  El programa opera en conjunto con el programa pline.
  Este programa permite modificar la longitud de la línea
  y el carácter escribiendo en un segmento de memoria compartido.
 */
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include "line.h"

main(int argc, char *argv[])
{
  int i, shmemid, semid;
  struct info *ctrl;
  struct shmid_ds shmbuf;
  if (argc!= 3) {
          fprintf(stderr, "uso: cline <char> <length>\n");
          exit(3);
  }
  shmemid = shmget(KEYSHMEM, SEGSIZE, 0);
  if (shmemid < 0) {
          perror("cline: shmget faild:");
          exit(1);
  }
  ctrl = (struct info *)shmat(shmemid, 0, 0);
  if (ctrl <= (struct info *)(0)) {
          perror("cline: shmat failed:");
          exit(2);
  }
          semid = semget(KEYSEM,0);
  if (semid < 0) {
          perror("cline: semget failed:");
          exit(1);
  }

          /* Intenta entrar en la seccion critica */
          Lock (semid, 0);
  /*Copia los datos de la linea de comandos en el segmento de memoria compartida */
  ctrl->c         = argv[1][0];
  ctrl->length    = atoi(argv[2]);
          /* Sale de la seccion critica */
          Unlock (semid, 0);

  exit(0);
}
```

```
/*
  pline.c
```

```
  Este es un ejemplo de uso de memoria compartida y semaforos IPC.
  El programa imprime una linea de texto cada cuatro segundos.
  La longitud de la línea, y el carácter impreso se controlan
  mediante dos valores incluidos en una pequeña estructura de datos
  en un segmento de memoria compartida. Cualquier otro programa puede
  vincular dicho segmento y modificar los campos de la estructura de datos.
*/

#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include "line.h"

main()
{
 int i, shmemid, semid, local;
 struct info *ctrl;
 struct shmid_ds shmbuf;

 shmemid = shmget(KEYSHMEM, SEGSIZE, IPC_CREAT | 0666);
 if (shmemid < 0) {
        perror("pline: shmget failed:");
        exit(1);
 }

 ctrl = (struct info *)shmat(shmemid, 0, 0);
 if (ctrl <= (struct info *)(0)) {
        perror("pline: shmat failed:");
        exit(2);
 }

        /* crea el semaforo IPC de un elemento */
        semid = semget (KEYSEM, SEMSIZE, IPC_CREAT | 0666);
 if (semid < 0) {
        perror("pline: semget failed:");
        exit(1);
 }

        /* Inicializa a 1 el elemento 0 del semaforo IPC */
        if (InicializaElemento (semid, 0, 1) < 0) {
                perror("pline: InicializaElemento failed:");
                exit (5);
        }

        /* Intenta entrar en la seccion critica */
        Lock (semid, 0);

 /*Inicializa los parámetros por defecto */
 ctrl->c        = 'a';
 ctrl->length   = 10;
        local = ctrl->length;

        /* Sale de la seccion critica*/
        Unlock (semid, 0);

 /*Este bucle imprime una linea cada 4 segundos */
 while (local > 0) {
                /* Intenta entrar en la seccion critica */
                Lock (semid, 0);

                for(i = 0; i < ctrl->length; i++)
                putchar(ctrl->c);
        putchar('\n');

                /* Sale de la seccion critica */
                Unlock (semid, 0);

        sleep(4);

        /* De nuevo entra en la seccion critica*/
                Lock (semid, 0);
                local = ctrl->length;
        /* Sale de la seccion critica */
                Unlock (semid, 0);
 }

 if (shmctl(shmemid,IPC_RMID,(struct msqid_ds *) NULL)<0) {
        perror("pline: shmctl failed:");
        exit(3);
 }
        if (semctl(semid,0,IPC_RMID)<0) {
        perror("pline: semctl failed:");
```

```
        exit(3);
  }
  printf ("Final de pline, adios!!!");
  exit(0);
}
```

```
/*
  funcsem.c
  Fichero en el que se implementan funciones de semaforos IPC:
   Lock
   Unlock
   InicializaElemento
*/

#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include "line.h"

void Lock (int sem, int n)
{
  struct sembuf sop;     // struct para una operación
  sop.sem_num = n;       // la operación se aplicará sobre el elemento n
  sop.sem_op = -1;       // operación de decremento de una unidad
  sop.sem_flg = 0;       // Se bloquea hasta que el recurso esté libre
  semop (sem, &sop, 1); // Llamada a semop().
}

void Unlock (int sem, int n)
{
  struct sembuf sop;     // struct para una operación
  sop.sem_num = n;       // se aplicará la operación sobre el elemento n
  sop.sem_op = 1;               // operación de incremento en una unidad
  sop.sem_flg = 0;
  semop (sem, &sop, 1); // Llamada a semop ()
}

int InicializaElemento (int sem, int n, int valor)
{
  union semun arg;       // union para los argumentos de la operación
  if (valor < 0)
          return -1;
  arg.val = valor;       // El elemento n se inicializará a valor
  semctl (sem, n, SETVAL, arg); // Llamada a semctl().
  return 0;
}
```

```
# Escuela Técnica Superior de Ingeniería Informática (Gijon)
# Sistemas de Computación (Curso 5)
# Aplicaciones Distribuidas en Unix con Sun RPC
#************************************************
# Variables
#************************************************

CC=gcc
CFLAGS= -Wall
CLIENTE = cliente
SERVIDOR= servidor

todo : $(CLIENTE) $(SERVIDOR)

#************************************************
# Generacion del cliente y del servidor
#************************************************
funcsem.o : funcsem.c line.h
        $(CC) -c funcsem.o funcsem.c

$(CLIENTE) : cliente.c ej1.h
        $(CC) $(CFLAGS) -o $(CLIENTE) cliente.c

$(SERVIDOR) : servidor.c ej1.h
        $(CC) $(CFLAGS) -o $(SERVIDOR) servidor.c

#************************************************
# Utilidades
#************************************************
limpiaobjs :
        rm -f *.o
```

BIBLIOGRAFÍA

Unix. Distributed Programming

Autor: Chris Brown
Editorial: Prentice-Hall
ISBN: 0-13-075896-5

Advanced Programming in the Unix Environment

Autor: W. Richard Stevens
Editorial: Addison-Wesley
ISBN: 0-201-56317-1

Unix Network Programming

Autor: W. Richard Stevens
Editorial: Prentice-Hall
ISBN: 0-13-949876-1

UNIX Programación práctica

Autor: Kay A. Robbins
Steven Robbins
Editorial: Prentice-Hall
ISBN: 968-880-959-4

Unix. Manual Pages