

Parte I: MUTEX

Contenido

En esta practica se continua con la sincronización de procesos, para este caso veremos otro mecanismo de sincronización, el mutex, este mecanismo nos sirve para sincronizar procesos ligeros de un proceso, esto se logra bloqueando y desbloqueando la ejecución de los hilos respectivos.

Objetivo Especifico

- a) Sincronización de hilos con mutex.
- b) Manejo de funciones y sus argumentos y variables con mutex.

Introduccion Teorica

Mutex

Un mutex consiste en una especie de semáforo binario con dos estados, cerrado y no cerrado. Un mutex es un objeto que permite a los hilos asegurar la integridad de un recurso compartido al que tienen acceso. Tiene dos estados : bloqueado y desbloqueado. Sobre un mutex se pueden realizar las siguientes operaciones:

- 1) **LOCK:** Intenta cerrar el mutex. Si el mutex no está cerrado, se cierra, todo ello en una acción atómica. Si el mutex está cerrado, el hilo se bloquea. Si dos hilos intentan cerrar el mutex al mismo tiempo, cosa que sólo puede pasar en un multiprocesador real, uno de ellos lo consigue y el otro no, bloqueándose. La forma de regular esto depende de la implementación.
- 2) **UNLOCK:** Elimina o libera el cierre del mutex. Si existe uno o más hilos esperando por el mutex, se desbloquea exactamente uno, y el resto permanece bloqueado a la espera.

Antes de acceder a un recurso compartido un hilo debe bloquear un mutex. Si el mutex no ha sido bloqueado antes por otro hilo, el bloqueo es realizado. Si el mutex ha sido bloqueado antes, el hilo es puesto a la espera. Tan pronto como el mutex es liberado, uno de los hilos en espera a causa de un bloqueo en el mutex es seleccionado para que continúe su ejecución, adquiriendo el bloqueo.

Un ejemplo de utilización de un mutex es aquél en el que un hilo A y otro hilo B están compartiendo un recurso típico, como puede ser una variable global. El hilo A bloquea el mutex, con lo que obtiene el acceso a la variable. Cuando el hilo B intenta bloquear el mutex, el hilo B es puesto a la espera puesto que el mutex ya ha sido bloqueado antes. Cuando el hilo A finaliza el acceso a la variable global, desbloquea el mutex. Cuando esto suceda, el hilo B continuará la ejecución adquiriendo el bloqueo, pudiendo entonces acceder a la variable.

Hilo A :	Hilo B :
lock(mutex)	lock(mutex)
acceso al recurso	acceso al recurso
unlock(mutex)	unlock(mutex)

Un hilo puede adquirir un mutex no bloqueado. De esta forma, la exclusin mutua entre hilos del mismo proceso est garantizada, hasta que el mutex es desbloqueado permitiendo que otros hilos protejan secciones crticas con el mismo mutex. Si un hilo intenta bloquear un mutex que ya est bloqueado, el hilo se suspende. Si un hilo desbloquea un mutex y otros hilos estn esperando por el mutex, el hilo en espera con mayor prioridad obtendr el mutex.

La tabla 1 muestra un resumen de las funciones mas importantes para la creacin y manipulacin de hilos y mutex:

<i>Gestin de hilos</i>	pthread_create pthread_exit pthread_kill pthread_join pthread_self
<i>Exclusin mutua</i>	pthread_mutex_init pthread_mutex_destroy pthread_mutex_lock pthread_mutex_trylock pthread_mutex_unlock

Tabla 1: Cuadro resumen de funciones para hilos y mutex

3 Sistemas Operativos, Guía 5

Material y Equipo

- a) Sistema operativo Linux
- b) Compilador gcc
- c) Guía de Laboratorio

Procedimiento

El siguiente programa ejecuta dos procesos ligeros, donde cada hilo imprime en pantalla cien veces el mismo carácter asignado, cada hilo imprime en pantalla un carácter distinto.

Digitar, compilar y ejecutar el programa hilos-sin-mutex.c :

Programa hilos-sin-mutex.c

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

void *funcion01(void *arg)
{
    int i;
    for(i=0;i<100;i++)
    {
        printf("*");
        fflush(stdout);
        sleep(1);
    }
    return NULL;
}

void *funcion02(void *arg)
{
    int i;
    for(i=0;i<100;i++)
    {
        printf("o");
        fflush(stdout);
        sleep(1);
    }
    return NULL;
}

int main(void)
{
    int i;
    pthread_t h1, h2;

    if(pthread_create(&h1, NULL, funcion01, NULL))
    {
        printf("error creando hilo");
        abort();
    }
```

```
if(pthread_create(&h2, NULL, funcion02, NULL))
{
    printf("error creando hilo");
    abort();
}

pthread_join(h1, NULL);
pthread_join(h2, NULL);

printf("\n");

exit(0);
}
```

```
walter@walter-desktop: ~/waltersanchez/practica09  
walter@walter-desktop:~/waltersanchez/practica09$ ./hilos-sin-mutex  
*****  
*****  
*****  
walter@walter-desktop:~/waltersanchez/practica09$
```

5 Sistemas Operativos, Guía 5

El siguiente programa presenta la versión sincronizada del programa hilos-sin-mutex.c, en esta versión se muestra el uso de mutex para sincronizar ambos procesos ligeros del manera que solo uno de los procesos ligeros pueda ejecutar su código de sección crítica a la vez y no ambos procesos ligeros al mismo tiempo.

Digitar, compilar y ejecutar el programa hilos-con-mutex.c :

Programa hilos-con-mutex.c

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

pthread_mutex_t varmutex;

void *funcion01(void *arg)
{
    int i;
    pthread_mutex_lock(&varmutex);
    for(i=0; i<100; i++)
    {
        printf("*");
        fflush(stdout);
        sleep(1);
    }
    pthread_mutex_unlock(&varmutex);
    return NULL;
}

void *funcion02(void *arg)
{
    int i;
    pthread_mutex_lock(&varmutex);
    for(i=0; i<100; i++)
    {
        printf("o");
        fflush(stdout);
        sleep(1);
    }
    pthread_mutex_unlock(&varmutex);
    return NULL;
}

int main(void)
{
    int i;
    pthread_t h1, h2;
    pthread_mutex_init(&varmutex, NULL);

    if(pthread_create(&h1, NULL, funcion01, NULL))
    {
        printf("error creando hilo");
        abort();
    }
```

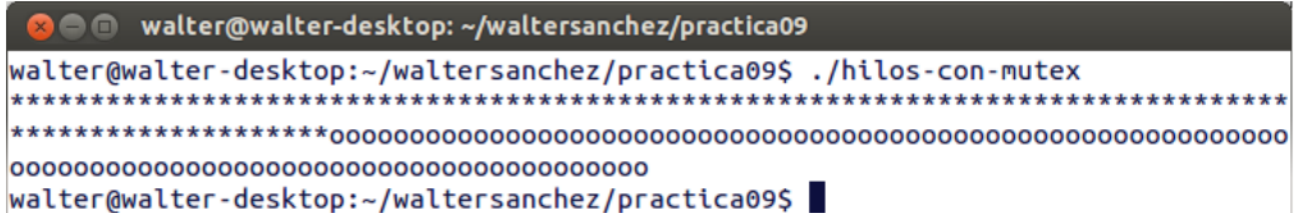
```
    if(pthread_create(&h2, NULL, funcion02, NULL))
    {
        printf("error creando hilo");
        abort();
    }

    pthread_join(h1, NULL);
    pthread_join(h2, NULL);

    pthread_mutex_destroy(&varmutex);

    printf("\n");

    exit(0);
}
```



```
walter@walter-desktop: ~/waltersanchez/practica09
walter@walter-desktop:~/waltersanchez/practica09$ ./hilos-con-mutex
*****
*****oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
walter@walter-desktop:~/waltersanchez/practica09$
```

Parte II: Gestión de Memoria

Contenido

En esta practica se abordara otra manera de comunicación entre procesos independientes entre si, los métodos antes vistos fueron las tuberías y los pasos de mensaje, estos métodos funcionan bien, pero consumen mucho tiempo de procesador dado que los datos pasan por el núcleo, para evitar este problema se usa el mentido de memoria compartida, este método evita el paso de datos por el núcleo por medio del mapeo de un mismo segmento de memoria para dos o mas procesos. Así un proceso siempre tendrá a su disposición la información generada por otros procesos que compartan el mismo segmento de memoria.

Objetivo Especifico

- a) Introducir al estudiante al concepto memoria compartida.
- b) Familiarizarse con las funciones Sistema V de memoria compartida y proyección de archivos en memoria.
- c) Resolver problemas relacionados con la comunicación y sincronización de procesos utilizando memoria compartida.

Introduccion Teorica

Memoria Compartida

Memoria compartida es la forma mas rápida de comunicación entre procesos disponible, una vez la memoria esta mapeada dentro del espacio de direcciones de los procesos que están compartiendo la región de memoria. El núcleo no se involucra cuando se da el paso de datos entre los procesos, es decir, los procesos no ejecutan ninguna llama dentro del núcleo para el paso de datos.

Obviamente el núcleo debe establecer el mapeo de memoria que permite a los procesos compartir la memoria, y entonces administra esta memoria en el lapso que dura el paso de datos.

El problema con la comunicación entre procesos que involucran tuberías y paso de mensajes es que para los procesos involucrados en el intercambio de información, la información tiene que pasar por el núcleo.

La memoria compartida provee una manera en que dos o mas procesos comparten una región de memoria. Los procesos por supuesto deben sincronizar el uso de la memoria compartida entre ellos.

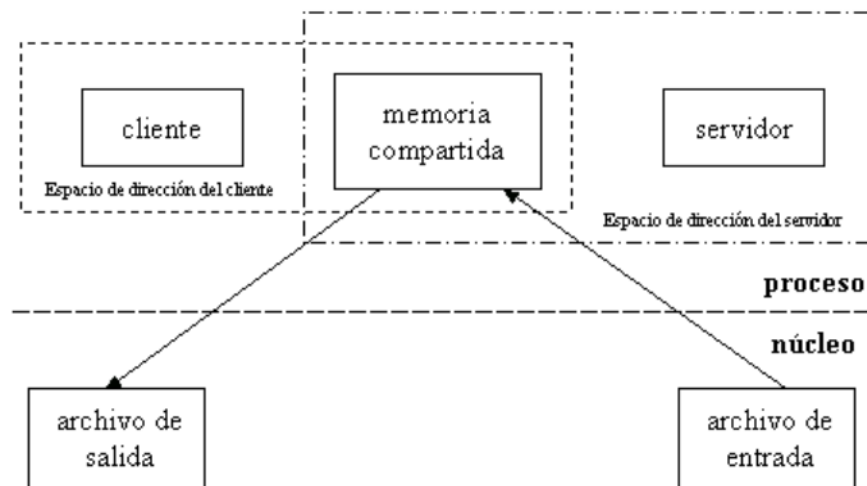


Figura 1. Copia de un archivo desde el servidor al cliente usando memoria compartida

En la figura 1 los datos son copiados solo dos veces. Desde el archivo de entrada dentro de la memoria compartida y desde la memoria compartida al archivo de salida. La memoria compartida aparece en el espacio de direcciones de ambos, el cliente y el servidor.

Servicios Sistema V Memoria Compartida

Por cada segmento de memoria compartida, el kernel mantiene la siguiente estructura de información:

```
<sys/shm.h>
struct shmid_ds{
    struct ipc_perm    shm_perm;
    size_t             shm_segsz;
    pid_t              shm_lpid;
    pid_t              shm_cpid;
    shmatt_t           shm_cnattch;
    time_t             shm_atime;
    time_t             shm_dtime;
    time_t             shm_ctime;
};

#include<sys/shm.h>
int shmget(key_t key, size_t size, int oflag);

#include<sys/shm.h>
void *shmat(int shmid, const void *shmaddr, int flag);

#include<sys/shm.h>
int shmdt(const void *shmaddr);

#include<sys/shm.h>
int shmctl(int shmid, int cmd, struct shmid_ds *buff);
```

Servicios POSIX Proyección de archivos en Memoria

```
#include<sys/mman.h>
void *mmap(void *addr, size_t len, int prot, int flags, int fd, off_t offset)

#include<sys/mman.h>
int munmap(void *addr, size_t len);
```


9 Sistemas Operativos, Guía 5

Material y Equipo

- a) Sistema operativo Linux
- b) Compilador gcc
- c) Guía de laboratorio

Procedimiento

1) Digitar el código de los programas productor y consumidor siguientes. El productor escribirá en memoria compartida una serie de números aleatorios y el consumidor los leerá de dicha memoria compartida.

productor.c

```
/* Productor */
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/shm.h>
#define TAMANIO 64
#define CLAVE (key_t) 1000

main(){
    int shmid;
    int *adr;
    int i;
    if((shmid=shmget(CLAVE,TAMANIO*sizeof(int), IPC_CREAT|0666))== -1){
        perror("shmget");
        exit(2);
    }
    if((adr=shmat(shmid,0,0)) == (int *) -1){
        perror("shmat");
        exit(2);
    }
    srand(getpid());
    for(i=0;i<TAMANIO;i++){
        printf("%d ",adr[i]=rand()%100);
        putchar('\n');
    }
}
```

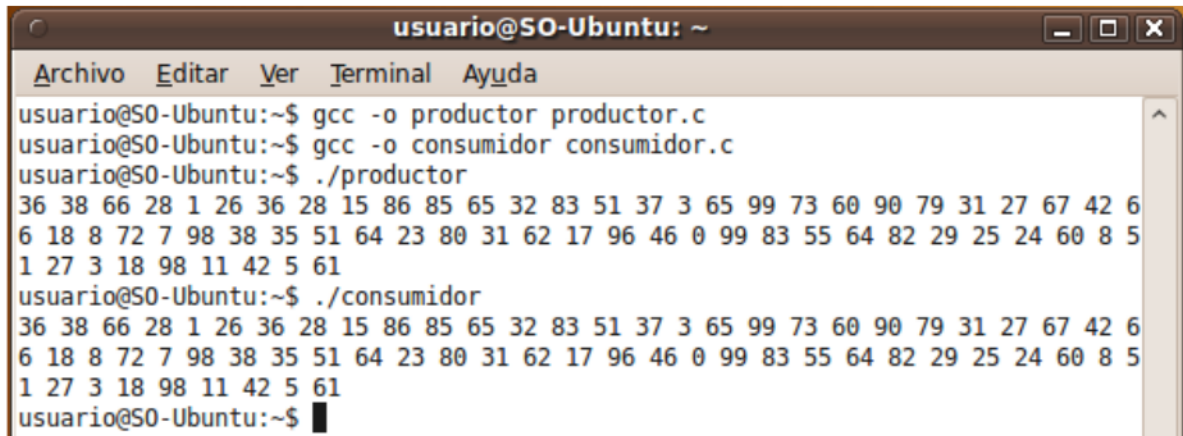
consumidor.c

```
/* consumidor */
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/shm.h>
#define TAMANIO 64
#define CLAVE (key_t) 1000

main(){
    int shmid;
    int *adr;
    int i;
    if((shmid=shmget(CLAVE,TAMANIO*sizeof(int), IPC_CREAT|0666))== -1){
        perror("shmget");
        exit(2);
    }
    if((adr=shmat(shmid,0,0)) == (int *) -1){
        perror("shmat");
        exit(2);
    }
}
```

```
}  
srand(getpid());  
for(i=0;i<TAMANIO;i++)  
    printf("%d ",adr[i]);  
putchar('\n');  
}
```

Compilación y ejecución del código fuente:



```
usuario@SO-Ubuntu: ~  
Archivo  Editar  Ver  Terminal  Ayuda  
usuario@SO-Ubuntu:~$ gcc -o productor productor.c  
usuario@SO-Ubuntu:~$ gcc -o consumidor consumidor.c  
usuario@SO-Ubuntu:~$ ./productor  
36 38 66 28 1 26 36 28 15 86 85 65 32 83 51 37 3 65 99 73 60 90 79 31 27 67 42 6  
6 18 8 72 7 98 38 35 51 64 23 80 31 62 17 96 46 0 99 83 55 64 82 29 25 24 60 8 5  
1 27 3 18 98 11 42 5 61  
usuario@SO-Ubuntu:~$ ./consumidor  
36 38 66 28 1 26 36 28 15 86 85 65 32 83 51 37 3 65 99 73 60 90 79 31 27 67 42 6  
6 18 8 72 7 98 38 35 51 64 23 80 31 62 17 96 46 0 99 83 55 64 82 29 25 24 60 8 5  
1 27 3 18 98 11 42 5 61  
usuario@SO-Ubuntu:~$
```

11 Sistemas Operativos, Guía 5

2) El siguiente programa es un ejemplo del uso de las funciones de proyección de archivos en memoria. Al programa se le debe pasar como parámetros el carácter a buscar y el archivo donde se realizara la búsqueda, como resultado se mostrara en pantalla el numero de ocurrencias del carácter en mención en el contenido del archivo.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    int i, fd, contador=0;
    char character;
    char *org, *p;
    struct stat bstat;

    if(argc!=3) {
        fprintf (stderr, "Uso: %s caracter archivo\n", argv[0]);
        return(1);
    }

    character=argv[1][0];

    if ((fd=open(argv[2], O_RDONLY))<0){
        perror("No puede abrirse el archivo");
        return(1);
    }

    if (fstat(fd, &bstat)<0){
        perror("Error en fstat del archivo");
        close (fd);
        return(1);
    }

    if((org=mmap((caddr_t)0, bstat.st_size, PROT_READ, MAP_SHARED, fd, 0)) == MAP_FAILED){
        perror("Error en la proyeccion del archivo");
        close (fd);
        return(1);
    }

    close(fd);

    p=org;
    for(i=0; i<bstat.st_size; i++)
        if(*p++==character) contador++;

    munmap(org, bstat.st_size);

    printf("%d\n", contador);
    return (0) ;
}
```

Ejemplo de compilación y ejecución del código fuente:

```
walter@walter-desktop: ~/waltersanchez/practica10
walter@walter-desktop:~/waltersanchez/practica10$ gcc proymemoria.c -o proymemoria
walter@walter-desktop:~/waltersanchez/practica10$
walter@walter-desktop:~/waltersanchez/practica10$ cat introduccion.txt
En esta practica se abordara otra manera de comunicación entre procesos independientes
entre si, los métodos antes vistos fueron las tuberías y los pasos de mensaje, estos
métodos funcionan bien, pero consumen mucho tiempo de procesador dado que los datos pa
san por el núcleo, para evitar este problema se usa el mentido de memoria compartida,
este método evita el paso de datos por el núcleo por medio del mapeo de un mismo segme
nto de memoria para dos o mas procesos. Así un proceso siempre tendrá a su disposición
la información generada por otros procesos que compartan el mismo segmento de memoria
.
walter@walter-desktop:~/waltersanchez/practica10$
walter@walter-desktop:~/waltersanchez/practica10$ ./proymemoria a introduccion.txt
45
walter@walter-desktop:~/waltersanchez/practica10$ ./proymemoria f introduccion.txt
3
walter@walter-desktop:~/waltersanchez/practica10$ ./proymemoria j introduccion.txt
1
walter@walter-desktop:~/waltersanchez/practica10$ ./proymemoria e introduccion.txt
67
walter@walter-desktop:~/waltersanchez/practica10$ █
```