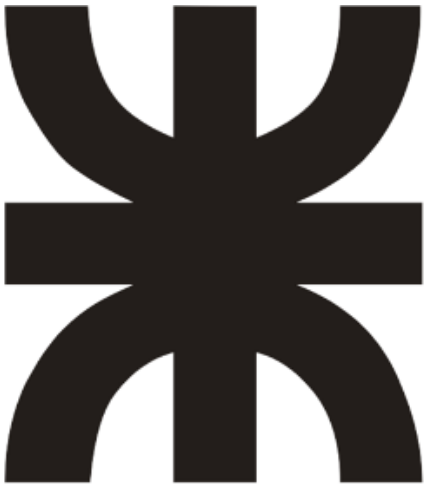
UNIVERSIDAD TECNOLOGICA NACIONAL



Ingeniería en sistemas de información

Asignatura: Sistemas Operativos
Comision: B
Segundo Cuatrimestre
Año: 2023

Trabajo Práctico 3
Tema: Comunicación entre procesos

Grupo: 11

Alumnos:
Meyer, Nahuel
nahuelmeyer16@gmail.com
Pividori, Marcos
mpividori3@gmail.com
Simonsini, Juan Pablo
juanpablosimonsini@gmail.com

TABLA DE CONTENIDOS

SECCION PRINCIPAL

MEMORIA COMPARTIDA Y SEMAFOROS

Ejercicio 1 Club 1:

```
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <linux/shm.h>
#include <linux/ipc.h>
#include <linux/sem.h>

#define SEGSIZE 128

int main(int argc, char *argv[]){

        //Club 1
        key_t keyClub1 = ftok(".", '1');
        if (keyClub1 == -1) exit(1);
        int semClub1 = semget(keyClub1, 1, 0666|IPC_CREAT);
        if(semClub1 == -1) exit(1);
        struct sembuf semb1 = {0, -1, 0};
        semctl(semClub1, 0, SETVAL, 0);

        //Club 2
        key_t keyClub2 = ftok(".", '2');
        if (keyClub2 == -1) exit(1);
        int semClub2 = semget(keyClub2, 1, 0666|IPC_CREAT);
        if (semClub2 == -1) exit(1);
        struct sembuf semb2 = {0, -1, 0};
        semctl(semClub2, 0, SETVAL, 0);
```

```
//MUTEX
key_t keyMutex = ftok(".", 'M');
if (keyMutex == -1) exit(1);
int mutex = semget(keyMutex, 1, 0666|IPC_CREAT);
if (mutex == -1) exit(1);
struct sembuf sembM = {0, -1, 0};
semctl(mutex, 0, SETVAL, 1);

//Memoria compartida
key_t keyMC = ftok(".",'E');
int MC = shmget(keyMC, SEGSIZE ,IPC_CREAT | 0666);
char *mcptr = (char *) shmat(MC, NULL, 0);

//Reserva memoria compartida y estampa color
semop(mutex, &sembM, 1);
strcpy(mcptr, "Rojo - ");
sembM.sem_op = 1;
semop(mutex, &sembM, 1);

//Libera al club2
semb2.sem_op = 1;
semop(semClub2, &semb2, 1);
semop(semClub1, &semb1, 1);

//Reserva memoria compartida y estampa color
semop(mutex, &sembM, 1);
strcat(mcptr, "Amarillo - ");
sembM.sem_op = 1;
semop(mutex, &sembM, 1);

//Libera al club2
semb2.sem_op = 1;
semop(semClub2, &semb2, 1);
```

```
	semop(semClub1, &semb1, 1);

	//Reserva memoria compartida y estampa color
	semop(mutex, &sembM, 1);
	strcat(mcptr, "Azul - ");
	sembM.sem_op = 1;
	semop(mutex, &sembM, 1);

	//Libera al club2
	semb2.sem_op = 1;
	semop(semClub2, &semb2, 1);

	semctl(semClub1, 0, IPC_RMID, 0);
	shmdt(mcptr);

	return 0;
}
```

Ejercicio 1 Club 2:

```
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <linux/shm.h>
#include <linux/ipc.h>
#include <linux/sem.h>

#define SEGSIZE 128

int main(int argc, char *argv[]){

	key_t keyClub2 = ftok(".", '2');
```

```
//Abre el semÃ¡foro del club2
int semClub2 = semget(keyClub2, 1, 0666);
if(semClub2 == -1) exit(1);
struct sembuf semb2 = {0, -1, 0};

//Abre el semÃ¡foro del club1
key_t keyClub1 = ftok(".", '1');
int semClub1 = semget(keyClub1, 1, 0666);
struct sembuf semb1 = {0, -1, 0};

//Se abre el semÃ¡foro MUTEX
key_t keyMutex = ftok(".", 'M');
if (keyMutex == -1) exit(1);
int mutex = semget(keyMutex, 1, 0666|IPC_CREAT);
if (mutex == -1) exit(1);
struct sembuf sembM = {0, -1, 0};

//Se abre el segmento de memoria compartida
key_t keyMC = ftok(".",'E');
int MC = shmget(keyMC, SEGSIZE , 0666);
char *mcptr = (char *) shmat(MC, NULL, 0);

//Se bloquea
semop(semClub2, &semb2, 1);

//Reserva memoria compartida y estampa color
semop(mutex, &sembM, 1);
strcat(mcptr, "Naranja - ");
sembM.sem_op = 1;
semop(mutex, &sembM, 1);

//Libera club1
semb1.sem_op = 1;
semop(semClub1, &semb1, 1);
```

```
        semop(semClub2, &semb2, 1);

        //Reserva memoria compartida y estampa color
        semop(mutex, &sembM, 1);
        strcat(mcptr, "Verde - ");
        sembM.sem_op = 1;
        semop(mutex, &sembM, 1);

        //Libera club1
        semb1.sem_op = 1;
        semop(semClub1, &semb1, 1);
        semop(semClub2, &semb2, 1);

        //Reserva memoria compartida y estampa color
        semop(mutex, &sembM, 1);
        strcat(mcptr, "Violeta");
        sembM.sem_op = 1;
        semop(mutex, &sembM, 1);

        //Muestra el resultado final
        printf("Bandera: %s\n", mcptr);

        //Elimina semÃ¡foro club2
        semctl(semClub2,0,IPC_RMID,0);
        //Elimina semÃ¡foro MUTEX
        semctl(mutex, 0, IPC_RMID, 0);
        //Se desliga de la memoria compartida
        shmdt(mcptr);

        return 0;
}
```

CLIENTE Y SERVIDOR CON MENSAJES

Ejercicio 2 Cliente:

```
#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/sem.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <unistd.h>


int main()

{

        key_t key;

        int semid;

        struct sembuf sb = {0, -1, 0};
        key = ftok(".", 'J');
        semid = semget(key, 1, 0);


        int fifo;


        // FIFO file path

        char clientefifo[50] = "/tmp/";

        int pid = getpid();
```

```
char strpid[10];

sprintf(strpid, "%d", pid);

strcat(clientefifo, strpid);
strcat(clientefifo, ".fifo");


// Crea la FIFO del cliente

mkfifo(clientefifo, 0666);


// serverFIFO file path

char * serverfifo = "/tmp/datetime.fifo";


// Genera la FIFO del server

mkfifo(serverfifo, 0666);

//Reserva el FIFO del server
semop(semid, &sb, 1);

//Abre la FIFO del server, envÃa el pedido y cierra

fifo = open(serverfifo,O_WRONLY);

write(fifo, strpid, strlen(strpid)+1);
close(fifo);

//Libera el FIFO del server
sb.sem_op = 1;
semop(semid, &sb, 1);
```

```
	//Abre la FIFO del cliente y lee la respuesta

	fifo = open(clientefifo,O_RDONLY);

	char array[80];

	read(fifo, array, sizeof(array));

	//Muestra la respuesta, cierra la serverFIFO, y borra su FIFO

	puts(array);

	close(fifo);

	unlink(clientefifo);


	return 0;
}
```

Ejercicio 2 Server:

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

#include <string.h>

#include <fcntl.h>
#include <time.h>
#include <errno.h>
#include <sys/ipc.h>
#include <sys/sem.h>

#include <sys/stat.h>

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

```
#include <unistd.h>


void crearSemaforo(){

	key_t key = ftok(".", 'J');

	int semid;
	semid=semget(key,1,0666|IPC_CREAT);
	semctl(semid,0,SETVAL,1);
}

void eliminarSemaforo(){

	key_t key= ftok(".", 'J');

	int semid;
	semid=semget(key,1,0);
	semctl(semid,0,IPC_RMID,0);
}

void responderSolicitud(char pid[20]){

	int fifo;
	char * dt_ptr;
	time_t t_secs;
	char respuesta[80] = "Fecha y Hora: ";

	//Crea la respuesta para el cliente
	t_secs = time(NULL);
	dt_ptr = ctime(&t_secs);
	strcat(respuesta,dt_ptr);

	printf("Cliente PID: %s", pid);
	printf(" %s\n", respuesta);
```

```
	//Genera el file path de la FIFO del cliente
	char clientefifo[50] = "/tmp/";

	strcat(clientefifo, pid);
	strcat(clientefifo, ".fifo");

	// Abre la FIFO del cliente

	mkfifo(clientefifo, 0666);

	// Abre la FIFO del cliente, escribe la respuesta y cierra

	fifo = open(clientefifo,O_WRONLY);

	write(fifo, respuesta, strlen(respuesta)+1);

	close(fifo);


}

int main()

{

	//Crea el semÃ¡foro que utilizarÃ¡ para su FIFO
	crearSemaforo();


	int fifo, ret;


	// serverFIFO file path

	char * serverfifo = "/tmp/datetime.fifo";
```

```
	// Crea la serverFIFO
	ret = mkfifo(serverfifo, 0666);

	char clientepid[20];

	while (1)
	{

		// Abre la FIFO del server, lee y cierra
		fifo = open(serverfifo,O_RDONLY);
		read(fifo, clientepid, sizeof(clientepid));

		if(strcmp(clientepid,"CERRAR") == 0)break;

		close(fifo);

		responderSolicitud(clientepid);


	}

	unlink("/tmp/datetime.fifo");

	eliminarSemaforo();

	return 0;
}
```

Ejercicio 2 Parar:

```
#include <stdio.h>

#include <stdlib.h>
#include <fcntl.h>
#include <string.h>
#include <sys/stat.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

        int fifo;

        fifo = open("/tmp/datetime.fifo",O_WRONLY);

        write(fifo, "CERRAR", strlen("CERRAR")+1);
        close(fifo);


return 0;

}
```

Ejercicio 3 Cliente:

```
#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/sem.h>
```

```
#include <sys/stat.h>

#include <sys/types.h>

#include <unistd.h>

struct msgbuf{

        long mtype;
        char mensaje[80];
} msg;


int main()

{

        key_t key = ftok(".", 'J');
        int msgid = msgget(key, 0666|IPC_CREAT);

        int pid = getpid();
        char strpid[10];

        sprintf(strpid, "%d", pid);


        //Prepara la solicitud

        msg.mtype = 1;
        strcpy(msg.mensaje, strpid);

        //Envia mensaje al server
        msgsnd(msgid, &msg, sizeof(msg), 0);

        //Recibe respuesta del server
        msgrcv(msgid, &msg, sizeof(msg), atoi(strpid),0);

        //Muestra la respuesta
```

```
	printf("%s\n", msg.mensaje);


	return 0;
}
```

Ejercicio 3 Server:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <time.h>
#include <errno.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

struct msgbuf{

	long mtype;
	char mensaje[80];
} msg;

int main()
{

	//Crea la cola de mensajes
```

```
key_t key = ftok(".", 'J');
int msgid = msgget(key, 0666|IPC_CREAT);


char clientepid[20];


while (1)
{


        msgrcv(msgid, &msg, sizeof(msg), 1, 0);

        if(strcmp(msg.mensaje,"CERRAR") == 0)break;

        char * dt_ptr;
        time_t t_secs;
        char respuesta[80] = "Fecha y Hora: ";

        //Crea la respuesta para el cliente
        t_secs = time(NULL);
        dt_ptr = ctime(&t_secs);
        strcat(respuesta,dt_ptr);

        printf("Cliente PID: %s", msg.mensaje);
        printf(" %s\n", respuesta);

        msg.mtype = atoi(msg.mensaje);
        strcpy(msg.mensaje, respuesta);

        msgsnd(msgid, &msg, sizeof(msg), 0);


}
```

```
	msgctl(msgid,IPC_RMID, NULL);


	return 0;
}
```

Ejercicio 3 Parar:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <time.h>
#include <errno.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>


struct msgbuf{

	long mtype;
	char mensaje[80];
} msg;


int main()
{

	//Crea la cola de mensajes
```

```
key_t key = ftok(".", 'J');
int msgid = msgget(key, 0666|IPC_CREAT);


char clientepid[20];


while (1)
{


        msgrcv(msgid, &msg, sizeof(msg), 1, 0);

        if(strcmp(msg.mensaje,"CERRAR") == 0)break;

        char * dt_ptr;
        time_t t_secs;
        char respuesta[80] = "Fecha y Hora: ";

        //Crea la respuesta para el cliente
        t_secs = time(NULL);
        dt_ptr = ctime(&t_secs);
        strcat(respuesta,dt_ptr);

        printf("Cliente PID: %s", msg.mensaje);
        printf(" %s\n", respuesta);

        msg.mtype = atoi(msg.mensaje);
        strcpy(msg.mensaje, respuesta);

        msgsnd(msgid, &msg, sizeof(msg), 0);


}
```

```
        msgctl(msgid,IPC_RMID, NULL);


        return 0;
}
```

SECCION DE DESCARGOS

Ejercicio 1:

Se debe ejecutar el proceso ./ejercicio1Club1 antes del ./ejercicio1Club2 ya que el ejercicio1Club1 es el encargado de crear los semáforos y memoria compartida.

Ejercicio 2:

El orden de ejecución es indiferente.
No puede existir más de un proceso ./ejercicio2Cliente al momento de ejecutar ./ejercicio2Server ya que sólo se responderá al primer proceso cliente creado.
El proceso ./ejercicio2Parar se utiliza para terminar la ejecución del server. Elimina las tuberías y semáforos automáticamente.

Ejercicio 3:

El orden de ejecución es indiferente.
Puede haber más de un ./ejercicio3Cliente esperando antes de ejecutar ./ejercicio3server.
El proceso ./ejercicio3Parar cumple la misma función que ./ejercicio2Parar en el enunciado anterior.

BIBLIOGRAFIA