

UNIVERSIDAD NACIONAL DE SAN AGUSTÍN DE AREQUIPA

ESCUELA PROFESIONAL DE CIENCIA DE LA COMPUTACIÓN
SISTEMAS OPERATIVOS



Laboratorio 4: Comunicación entre Procesos

Presentado por:

Rushell Vanessa Zavalaga Orozco

Docente :

Dra. Yessenia D. Yari R.



1. Actividades:

1.1. Librerías `sys/ipc.h` y `sys/shm.h`

`<sys/ipc.h>` Describe las estructuras que utilizan las subrutinas que realizan operaciones de comunicación entre procesos. Es utilizado por tres mecanismos para la comunicación entre procesos (IPC): mensajes, semáforos y memoria compartida. Todos utilizan un tipo de estructura común, `ipc_perm`, para pasar información utilizada para determinar el permiso para realizar una operación de IPC.

`<sys/shm.h>` contiene funciones necesarias para trabajar con memoria compartida. Funciones como `shmget()` (crea o accede a un segmento de memoria compartida), `shmat()` (adjunta el segmento a un proceso), y `shmctl()` (realiza operaciones de control sobre el segmento).

1.2. ¿Cómo se crea una tubería?

Una tubería o pipe se utiliza para permitir la comunicación entre procesos, pasando datos de un proceso a otro de manera secuencial. Para crear una tubería en C, se utiliza la función `pipe()`, que toma un array de dos enteros. Este array actúa como los descriptores de archivo para leer y escribir datos en la tubería.

```
int pipefd[2];  
int result = pipe(pipefd);
```

- `pipefd[0]`: Se utiliza para leer datos de la tubería.
- `pipefd[1]`: Se utiliza para escribir datos en la tubería.

1.3. Utilidad de la tubería en el Programa 1

Programa 1:

```
#include <stdio.h>  
#include <unistd.h>  
#include <string.h>  
  
int main() {  
    int fd[2]; // Array para el pipe  
    pid_t pid;  
    char buffer[100];
```

```
// Crear el pipe
if (pipe(fd) == -1) {
    printf("Error al crear el pipe.\n");
    return 1;
}

pid = fork(); // Crear el proceso hijo

if (pid < 0) {
    printf("Error al crear el proceso.\n");
    return 1;
} else if (pid == 0) {
    // Proceso hijo
    close(fd[1]); // Cerrar el extremo de escritura
    read(fd[0], buffer, sizeof(buffer)); // Leer desde el pipe
    printf("Proceso hijo recibió: %s\n", buffer);
    close(fd[0]); // Cerrar el extremo de lectura
} else {
    // Proceso padre
    close(fd[0]); // Cerrar el extremo de lectura
    char mensaje[] = "Hola desde el proceso padre!";
    write(fd[1], mensaje, strlen(mensaje) + 1); // Escribir en el pipe
    close(fd[1]); // Cerrar el extremo de escritura
}

return 0;
}
```

Análisis: la tubería sirve para comunicar dos procesos (padre e hijo, o cualquier otro par de procesos) de manera unidireccional o bidireccional. Permite que un proceso escriba datos que el otro proceso pueda leer. Esto es útil, por ejemplo, cuando uno de los procesos genera información que el otro necesita procesar.

- Comunicación entre procesos. Permite que el proceso padre envíe (write) un mensaje al proceso hijo el cual lo lee (read).
- Extremos de la tubería: En el programa, fd[0] se utiliza para la lectura (extremo de lectura) y fd[1] para la escritura (extremo de escritura).
- El proceso padre cierra el extremo de lectura antes de escribir y el proceso hijo cierra el extremo de escritura antes de leer, lo que asegura que solo el proceso que necesita leer o escribir esté utilizando cada extremo.

1.4. ¿Para qué creo el segmento de memoria compartida, que comparten el proceso padre e hijo?

Programa 2:

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

int main() {
    // Crear clave para la memoria compartida
    key_t key = ftok("shmfile", 65);

    // Crear segmento de memoria compartida
    int shmid = shmget(key, 1024, 0666|IPC_CREAT);

    // Adjuntar la memoria compartida al proceso
    char *str = (char*) shmat(shmid, NULL, 0);

    if (fork() == 0) {
        // Proceso hijo escribe en la memoria compartida
        sprintf(str, "Hola desde el proceso hijo.");
        shmdt(str); // Desadjuntar memoria compartida
    } else {
        // Proceso padre lee de la memoria compartida
        sleep(1); // Esperar para que el hijo escriba
        printf("Proceso padre lee: %s\n", str);
        shmdt(str); // Desadjuntar memoria compartida
        shmctl(shmid, IPC_RMID, NULL); // Eliminar el segmento de memoria compartida
    }

    return 0;
}
```

Existen varias razones para crear un segmento de memoria compartida, entre estas estan:

- Permite que los procesos padre e hijo accedan y modifiquen una misma región de memoria.
- Los datos se escriben y leen directamente desde la memoria, por lo que la comunicación es la más rápida.
- Se pierde la necesidad de duplicar la información. Los procesos acceden directamente a los mismos datos, lo que reduce la duplicación de memoria y ahorra tiempo en copiar datos.

1.5. ¿Por qué debo eliminar el segmento de memoria compartida?

Debe ser eliminado para evitar fugas de memoria y consumo innecesario de recursos del sistema. Si no se elimina, el segmento continuará ocupando espacio en la memoria del sistema. La eliminación se realiza con la función `shmctl()` usando la opción `IPC_RMID`

1.6. Comunicación por tubería vs Memoria Compartida

La elección depende del uso que les demos:

Si la comunicación es simple y secuencial (por ejemplo, un proceso produce datos que otro proceso consume inmediatamente), las tuberías son una buena opción porque son fáciles de implementar y seguras en cuanto a sincronización.

Pero si se necesita compartir grandes volúmenes de datos o si ambos procesos deben acceder frecuentemente a la misma información, la memoria compartida es más eficiente porque no hay necesidad de copiar datos entre procesos. Sin embargo, es más compleja porque necesitas mecanismos adicionales como semáforos para la sincronización.

2. Analisis de código

2.1. Programa 1:

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>

int main() {
    int fd[2]; // Array para el pipe
    pid_t pid;
    char buffer[100];

    // Crear el pipe
    if (pipe(fd) == -1) {
        printf("Error al crear el pipe.\n");
        return 1;
    }

    pid = fork(); // Crear el proceso hijo

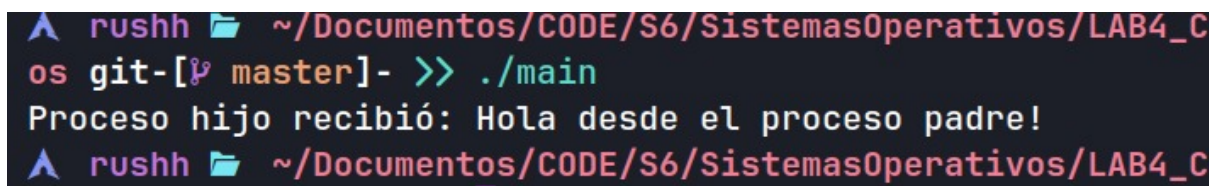
    if (pid < 0) {
```

```
    printf("Error al crear el proceso.\n");
    return 1;
} else if (pid == 0) {
    // Proceso hijo
    close(fd[1]); // Cerrar el extremo de escritura
    read(fd[0], buffer, sizeof(buffer)); // Leer desde el pipe
    printf("Proceso hijo recibió: %s\n", buffer);
    close(fd[0]); // Cerrar el extremo de lectura
} else {
    // Proceso padre
    close(fd[0]); // Cerrar el extremo de lectura
    char mensaje[] = "Hola desde el proceso padre!";
    write(fd[1], mensaje, strlen(mensaje) + 1); // Escribir en el pipe
    close(fd[1]); // Cerrar el extremo de escritura
}

return 0;
}
```

Analisis:

1. Se crea el pipe para la comunicación.
2. Se crea un proceso hijo mediante fork()
3. El proceso hijo cierra su extremo de escritura y lee datos enviados a través del pipe por el proceso padre
4. El proceso padre cierra su extremo de lectura y escribe el mensaje en el pipe.
5. El proceso hijo recibe el mensaje y lo imprime en la consola.



```
rushh ~/Documentos/CODE/S6/SistemasOperativos/LAB4_Co
os git-[master]- >> ./main
Proceso hijo recibió: Hola desde el proceso padre!
rushh ~/Documentos/CODE/S6/SistemasOperativos/LAB4_Co
```

2.2. Programa 2:

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

int main() {
    // Crear clave para la memoria compartida
```

```
key_t key = ftok("shmfile", 65);

// Crear segmento de memoria compartida
int shmid = shmget(key, 1024, 0666|IPC_CREAT);

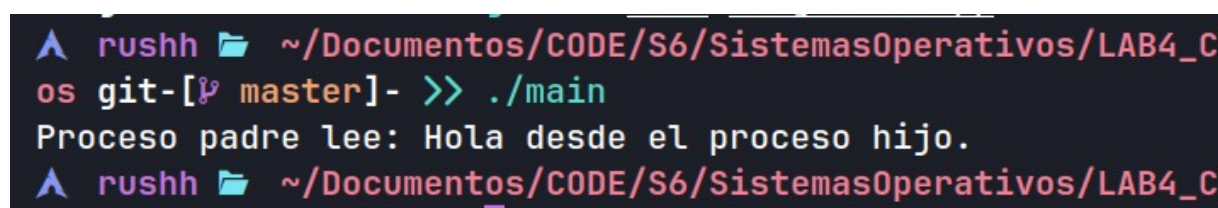
// Adjuntar la memoria compartida al proceso
char *str = (char*) shmat(shmid, NULL, 0);

if (fork() == 0) {
    // Proceso hijo escribe en la memoria compartida
    sprintf(str, "Hola desde el proceso hijo.");
    shmdt(str); // Desadjuntar memoria compartida
} else {
    // Proceso padre lee de la memoria compartida
    sleep(1); // Esperar para que el hijo escriba
    printf("Proceso padre lee: %s\n", str);
    shmdt(str); // Desadjuntar memoria compartida
    shmctl(shmid, IPC_RMID, NULL); // Eliminar el segmento de memoria compartida
}

return 0;
}
```

Analisis:

1. El proceso padre crea un segmento de memoria compartida y lo adjunta a su espacio de direcciones.
2. Se crea un proceso hijo utilizando fork()
3. El proceso hijo escribe un mensaje en la memoria compartida.
4. El proceso padre espera un segundo, luego lee el mensaje escrito por el proceso hijo desde la memoria compartida
5. Ambos procesos desvinculan la memoria compartida, y el proceso padre elimina el segmento de memoria compartida al final



```
rushh ~/Documentos/CODE/S6/SistemasOperativos/LAB4_C
os git-[master]- >> ./main
Proceso padre lee: Hola desde el proceso hijo.
rushh ~/Documentos/CODE/S6/SistemasOperativos/LAB4_C
```

3. Calculadora con tuberías y mensajes compartidos

3.1. Calculadora con tuberías:

Calculadora con tuberías de tal forma que el proceso padre envía una operación aritmética al proceso hijo, y este realice el cálculo para luego enviar el resultado de vuelta al proceso padre.

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <sys/wait.h> // Incluir para usar la función wait()

int main() {
    int fd[2]; // Array para el pipe
    pid_t pid;
    char operacion[100];
    char buffer[100];

    // Crear el pipe
    if (pipe(fd) == -1) {
        printf("Error al crear el pipe.\n");
        return 1;
    }

    pid = fork(); // Crear el proceso hijo

    if (pid < 0) {
        printf("Error al crear el proceso.\n");
        return 1;
    } else if (pid == 0) { // Proceso hijo
        close(fd[1]); // Cerrar el extremo de escritura del pipe
        read(fd[0], operacion, sizeof(operacion)); // Leer la operación enviada por el padre
        printf(" > Proceso hijo recibe: %s\n", operacion);

        // Realizar el cálculo
        char operador;
        int num1, num2, res;
        sscanf(operacion, "%d %c %d", &num1, &operador, &num2);

        switch (operador) {
            case '+':
                res = num1 + num2;
                break;
            case '-':
```



```
        res = num1 - num2;
        break;
    case '*':
        res = num1 * num2;
        break;
    case '/':
        if (num2 != 0) {
            res = num1 / num2;
        } else {
            strcpy(buffer, "Error: División por cero.");
            printf(" > Proceso hijo envía: %s\n", buffer);
            write(fd[1], buffer, strlen(buffer) + 1);
            close(fd[0]); // Cerrar el extremo de lectura
            close(fd[1]); // Cerrar el extremo de escritura antes de salir
            exit(1);
        }
        break;
    default:
        strcpy(buffer, "Error: Operación no válida.");
        printf(" > Proceso hijo envía: %s\n", buffer);
        write(fd[1], buffer, strlen(buffer) + 1);
        close(fd[0]); // Cerrar el extremo de lectura
        close(fd[1]); // Cerrar el extremo de escritura antes de salir
        exit(1);
}

// Preparar el resultado para enviarlo al padre
close(fd[0]); // Cerrar el extremo de lectura después de leer
sprintf(buffer, "Resultado: %d\n", res); // Escribir el resultado en el buffer
printf(" > Proceso hijo envía: %s", buffer);
write(fd[1], buffer, strlen(buffer) + 1); // Enviar el resultado al padre
close(fd[1]); // Cerrar el extremo de escritura después de escribir
exit(0); // Salir del proceso hijo
} else { // Proceso padre
    close(fd[0]); // Cerrar el extremo de lectura del pipe
    printf(" >> Calculadora con tuberías <<\n\n");
    printf("Operacion >> ");
    fgets(operacion, sizeof(operacion), stdin); // Leer la operación desde la entrada
    printf("\n > Proceso padre envía: %s", operacion);
    write(fd[1], operacion, strlen(operacion) + 1); // Enviar la operación al hijo

    close(fd[1]); // Cerrar el extremo de escritura después de escribir
    wait(NULL); // Esperar a que el hijo termine

    // Leer el resultado enviado por el hijo
    read(fd[0], buffer, sizeof(buffer));
    printf(" > Proceso padre recibe: %s", buffer);
```

```
        close(fd[0]); // Cerrar el extremo de lectura después de leer
        exit(0); // Salir del proceso padre
    }

    return 0;
}
```

```
^ rushh ~/Documentos/CODE/S6/SistemasOper
>> Calculadora con tuberías <<

Operacion >> 5 + 3

> Proceso padre envía: 5 + 3
> Proceso hijo recibe: 5 + 3

> Proceso hijo envía: Resultado: 8
> Proceso padre recibe: %
~/Documentos/CODE/S6/SistemasOperativos/LAB
>> Calculadora con tuberías <<

Operacion >> 5 * 15

> Proceso padre envía: 5 * 15
> Proceso hijo recibe: 5 * 15

> Proceso hijo envía: Resultado: 75
> Proceso padre recibe: %
~/Documentos/CODE/S6/SistemasOperativos/LAB
>> Calculadora con tuberías <<

Operacion >> 8 / 0

> Proceso padre envía: 8 / 0
> Proceso hijo recibe: 8 / 0

> Proceso hijo envía: Error: División por ce
> Proceso padre recibe: %
~/Documentos/CODE/S6/SistemasOperativos/LAB
>> Calculadora con tuberías <<

Operacion >> 5 - 1

> Proceso padre envía: 5 - 1
> Proceso hijo recibe: 5 - 1

> Proceso hijo envía: Resultado: 4
> Proceso padre recibe: %
^ rushh ~/Documentos/CODE/S6/SistemasOper
```

3.2. Calculadora con mensajes compartidos:

Calculadora con mensajes compartidos de tal forma que el proceso padre envía una operación aritmética al proceso hijo, y este realice el cálculo para luego enviar el resultado de vuelta al proceso padre (no pude encontrar el error, para que el proceso padre pueda imprimir el resultado).

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdlib.h>
#include <sys/wait.h>

int main() {
    pid_t pid;
    char operacion[100];

    // Crear la memoria compartida
    key_t key = ftok("shmfile", 65);
    int shmid = shmget(key, 1024, 0666 | IPC_CREAT);
    char *shm_op = (char*) shmat(shmid, NULL, 0);
    char *shm_res = shm_op + 512; // Usar la mitad de la memoria para el resultado

    pid = fork(); // Crear el proceso hijo

    if (pid < 0) {
        printf("Error al crear el proceso.\n");
        return 1;
    } else if (pid == 0) {
        // Proceso hijo: Realiza el cálculo
        sleep(3); // Esperar un poco para asegurarnos de que el padre escribe primero
        printf(" > Proceso hijo recibe: %s\n", shm_op);

        // Realizar el cálculo
        char operador;
        int num1, num2, res;
        sscanf(shm_op, "%d %c %d", &num1, &operador, &num2);

        switch (operador) {
            case '+':
                res = num1 + num2;
                break;
            case '-':
                res = num1 - num2;
                break;
        }
    }
}
```

```
        case '*':
            res = num1 * num2;
            break;
        case '/':
            if (num2 != 0)
                res = num1 / num2;
            else {
                strcpy(shm_res, "Error: División por cero.");
                shmdt(shm_op); // Desadjuntar memoria
                exit(1); // Terminar el proceso hijo con error
            }
            break;
        default:
            strcpy(shm_res, "Error: Operación no válida.");
            shmdt(shm_op);
            exit(1); // Terminar el proceso hijo con error
    }

    sprintf(shm_res, "Resultado: %d", res);
    printf(" > Proceso hijo envía: %s\n", shm_res);
    shmdt(shm_op); // Desadjuntar memoria compartida
    exit(0); // Terminar el proceso hijo correctamente

} else {
    // Proceso padre
    printf(" >> Calculadora con memoria compartida <<\n\n");
    printf(" Operación >> ");
    fgets(operacion, sizeof(operacion), stdin); // Leer la operación de la entrada
    printf("\n > Proceso padre envía: %s", operacion);
    strcpy(shm_op, operacion); // Escribir la operación en la memoria compartida

    // Esperar a que el hijo termine
    wait(NULL); // Esperar a que el hijo termine

    // Leer el resultado desde la memoria compartida
    printf(" > Proceso padre recibe: %s\n", shm_res);

    // Desadjuntar y eliminar la memoria compartida
    shmdt(shm_op);
    shmctl(shmid, IPC_RMID, NULL);
}

return 0;
}
```

```
>> Calculadora con memoria compartida <<

Operación >> 5 + 8

> Proceso padre envía: 5 + 8
> Proceso hijo recibe: 5 + 8

> Proceso hijo envía: Resultado: 13
> Proceso padre recibe: Resultado: 13
^ rushh ~/Documentos/CODE/S6/SistemasOperativos/L
ter]- >> ./main
>> Calculadora con memoria compartida <<

Operación >> 5 * 5

> Proceso padre envía: 5 * 5
> Proceso hijo recibe: 5 * 5

> Proceso hijo envía: Resultado: 25
> Proceso padre recibe: Resultado: 25
^ rushh ~/Documentos/CODE/S6/SistemasOperativos/L
ter]- >> ./main
>> Calculadora con memoria compartida <<

Operación >> 4 / 0

> Proceso padre envía: 4 / 0
> Proceso hijo recibe: 4 / 0

> Proceso padre recibe: Error: División por cero.
^ rushh ~/Documentos/CODE/S6/SistemasOperativos/L
ter]- >> ./main
>> Calculadora con memoria compartida <<

Operación >> 4 - 2

> Proceso padre envía: 4 - 2
> Proceso hijo recibe: 4 - 2

> Proceso hijo envía: Resultado: 2
> Proceso padre recibe: Resultado: 2
^ rushh ~/Documentos/CODE/S6/SistemasOperativos/L
```