Programación Concurrente, Mayo 2023

Práctica 6

Martínez Buenrostro Jorge Rafael.
Universidad Autónoma Metropolitana
Unidad Iztapalapa, México
molap96@gmail.com

Actividad B

El objetivo de esta actividad es deducir la capacidad de un pipe modificando el programa de ejemplo. Para esto habrá que llenar byte por byte el pipe hasta que se llene, haremos las siguientes dos modificaciones:

1. Se agrega en el proceso emisor un ciclo en el cual se agrega un byte hasta que ya no se pueda escribir más en el pipe.

```
if (pid == 0) {
    close(hijo_padre[READ]); // Cierra el que no va a usar
    do{
        printf(" Hijo: Escribo en la tuberia el caracter %d\n", iteracion);
        iteracion++;
    }while(write(hijo_padre[WRITE],"0",1));
    close(hijo_padre[WRITE]);
    printf(" Hijo: termino\n");
    exit(0);
```

2. Para que el pipe se pueda llenar se comenta la línea en el proceso receptor que se encarga de leer datos del pipe.

```
} else {
    close(hijo_padre[WRITE]); // Cierra el que no va a usar
    printf("Padre: leyendo de la tuberia\n");
    //read(hijo_padre[READ], buf, 5);
    printf("Padre lei: read \"%s\"\n", buf);
    printf("Padre : termino\n");
    wait(NULL);
    close(hijo_padre[READ]);
}
```

Al ejecutar este programa en la terminal se muestra una cuenta de la cantidad de bytes que se van agregando al pipe; esta cuenta se detiene en el número **65,536.** Convirtiendo este número en potencias de **2** podemos ver que el tamaño del pipe es de 2^{16} bytes.

Enlace GDB

Código Fuente

```c
/*
** pipe.c -- un ejemplo de uso de pipe para
**            comunicar procesos por env�o de mensajes
*/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

#define WRITE 1
#define READ  0

int main(){
   int hijo_padre[2];
   char buf[30];
   int status, pid, iteracion;
   iteracion = 0;

   status = pipe(hijo_padre);
   if (status == -1 ){
       printf("ERROR al crear el pipe");
       exit(1);
   }

   if ((pid = fork()) == -1 ) {
       printf("ERROR al clonar el proceso");
       exit(1);
   }

   if (pid == 0) {
       close(hijo_padre[READ]); // Cierra el que no va a usar
       do{
           printf(" Hijo: Escribo en la tuberia el caracter %d\n", iteracion);
           iteracion++;
       }while(write(hijo_padre[WRITE],"0",1));
       close(hijo_padre[WRITE]);
```

```
        printf(" Hijo: termino\n");
        exit(0);
    } else {
        close(hijo_padre[WRITE]); // Cierra el que no va a usar
        printf("Padre: leyendo de la tuberia\n");
        //read(hijo_padre[READ], buf, 5);
        printf("Padre lei: read \"%s\"\n", buf);
        printf("Padre : termino\n");
        wait(NULL);
        close(hijo_padre[READ]);
    }
    return 0;
}
```

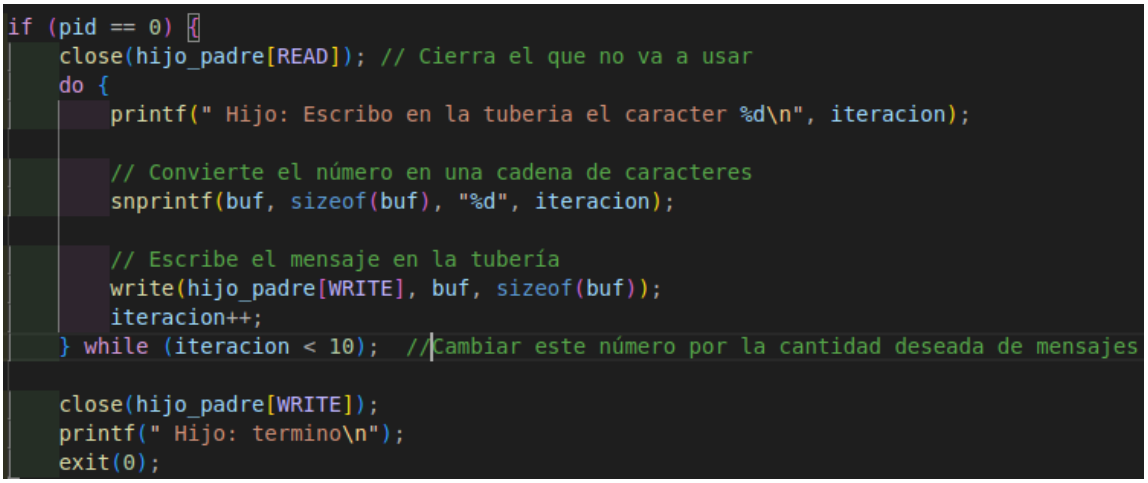
Actividad C

Esta actividad requiere que modifiquemos el programa ejemplo para que el proceso emisor le envíe varios mensajes al proceso receptor. Para esto habrá que realizar las siguientes modificaciones:

3. Primero se declara una variable *iteracion* poder seleccionar la cantidad deseada de mensaje a enviar. Al proceso emisor se le agrega un ciclo que se repite el el número de veces determinadas por la variable *iteracion;* dentro de este ciclo se encuentran las líneas encargadas de: mostrar en la terminal el número que se enviará, la línea encargada de convertir el número a enviar en una cadena de caracteres y la línea encargada de escribir en el pipe el número a enviar.

```
if (pid == 0) {
    close(hijo_padre[READ]); // Cierra el que no va a usar
    do {
        printf(" Hijo: Escribo en la tuberia el caracter %d\n", iteracion);

        // Convierte el número en una cadena de caracteres
        snprintf(buf, sizeof(buf), "%d", iteracion);

        // Escribe el mensaje en la tubería
        write(hijo_padre[WRITE], buf, sizeof(buf));
        iteracion++;
    } while (iteracion < 10);  //Cambiar este número por la cantidad deseada de mensajes

    close(hijo_padre[WRITE]);
    printf(" Hijo: termino\n");
    exit(0);
```

4. En el proceso receptor se agrega un ciclo que se encarga de leer el pipe hasta que este se encuentre vacío, dentro de este ciclo se muestra en la terminal lo que se leyó del pipe.



```c
if (pid == 0) {
    close(hijo_padre[READ]); // Cierra el que no va a usar
    do {
        printf(" Hijo: Escribo en la tuberia el caracter %d\n", iteracion);

        // Convierte el número en una cadena de caracteres
        snprintf(buf, sizeof(buf), "%d", iteracion);

        // Escribe el mensaje en la tubería
        write(hijo_padre[WRITE], buf, sizeof(buf));
        iteracion++;
    } while (iteracion < 10);  //Cambiar este número por la cantidad deseada de mensajes

    close(hijo_padre[WRITE]);
    printf(" Hijo: termino\n");
    exit(0);
```

Enlace GDB

Código Fuente

```c
/*
** pipe.c -- un ejemplo de uso de pipe para
**            comunicar procesos por envío de mensajes
*/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

#define WRITE 1
#define READ  0

int main() {
   int hijo_padre[2];
   char buf[30];
   int status, pid, iteracion;
   iteracion = 0;

   status = pipe(hijo_padre);
   if (status == -1) {
```

```
        printf("ERROR al crear el pipe");
        exit(1);
    }

    if ((pid = fork()) == -1) {
        printf("ERROR al clonar el proceso");
        exit(1);
    }

    if (pid == 0) {
        close(hijo_padre[READ]); // Cierra el que no va a usar
        do {
            printf(" Hijo: Escribo en la tuberia el caracter %d\n", iteracion);

            // Convierte el número en una cadena de caracteres
            snprintf(buf, sizeof(buf), "%d", iteracion);

            // Escribe el mensaje en la tubería
            write(hijo_padre[WRITE], buf, sizeof(buf));
            iteracion++;
        } while (iteracion < 10);  //Cambiar este número por la cantidad deseada de
mensajes
        close(hijo_padre[WRITE]);
        printf(" Hijo: termino\n");
        exit(0);
    } else {
        close(hijo_padre[WRITE]); // Cierra el que no va a usar
        printf("Padre: leyendo de la tuberia\n");
        while (read(hijo_padre[READ], buf, sizeof(buf)) > 0) {
            printf("Padre lei: read \"%s\"\n", buf);
        }
        printf("Padre: termino\n");
        wait(NULL);
        close(hijo_padre[READ]);
    }
    return 0;
}
```

Actividad D

Esta actividad nos pide que agreguemos lo que sea necesario para que la comunicación entre el proceso emisor y el proceso receptor sea en ambos sentidos.

1. Primero se crean dos pipes: ***receptor_emisor*** y ***emisor_receptor***. Con estos nuevos pipes el proceso emisor puede enviar mensajes por medio del pipe ***emisor_receptor*** y recibir respuestas del proceso receptor por medio del pipe ***receptor_emisor***. Mientras que el proceso receptor lee mensajes del pipe ***emisor_receptor*** y envía respuestas al proceso emisor por medio del pipe ***receptor_emisor.***

```c
int emisor_receptor[2];
int receptor_emisor[2];
```

2. En el proceso emisor se agregan un ciclo que permite que pueda mandar mensajes hasta que se quiera salir. Dentro del ciclo se pide el mensaje que se quiere mandar y se escribe dentro del pipe ***emisor_receptor***. Se evalua el mensaje para saber si contiene la palabra reservada para terminar el envío de mensajes; para terminar se lee del pipe ***receptor_emisor*** la respuesta del proceso receptor y se muestra en pantall

```c
close(emisor_receptor[READ]);
close(receptor_emisor[WRITE]);

while (1) {
    // Leer mensaje desde la entrada estándar
    printf("Emisor: ingrese un mensaje ('terminar' para salir): ");
    fgets(buf, sizeof(buf), stdin);
    buf[strcspn(buf, "\n")] = '\0';  // Eliminar el salto de línea

    // Enviar mensaje al proceso receptor
    write(emisor_receptor[WRITE], buf, sizeof(buf));

    // Verificar si el mensaje es "exit"
    if (strcmp(buf, "terminar") == 0) {
        break;
    }

    // Leer respuesta del proceso receptor
    read(receptor_emisor[READ], buf, sizeof(buf));
    printf("Emisor: respuesta recibida \"%s\"\n", buf);
}
```

3. En el proceso receptor se agrega un ciclo que permite que pueda mandar mensajes hasta que se quiera salir. Dentro del ciclo primero lee el pipe ***emisor_receptor*** en búsqueda de la respuesta del emisor; en caso que haya una respuesta se compara con la palabra reservada para terminar con la comunicación. Al final escribe dentro del pipe ***receptor_emisor*** la confirmación de recepción del mensaje.

```
close(emisor_receptor[WRITE]);
close(receptor_emisor[READ]);

while (1) {
    // Leer mensaje del proceso emisor
    read(emisor_receptor[READ], buf, sizeof(buf));
    printf("Receptor: recibido \"%s\"\n", buf);

    // Verificar si el mensaje es "exit"
    if (strcmp(buf, "terminar") == 0) {
        break;
    }

    // Enviar respuesta al proceso emisor
    write(receptor_emisor[WRITE], "Mensaje recibido", sizeof("Mensaje recibido"));
}

close(emisor_receptor[READ]);
close(receptor_emisor[WRITE]);
printf("Receptor: termino\n");
exit(0);
```

Enlace GDB

Código Fuente

```
/*
** pipe.c -- un ejemplo de uso de pipe para
**           comunicar procesos por envío de mensajes
*/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <string.h>

#define WRITE 1
#define READ  0

int main() {
```

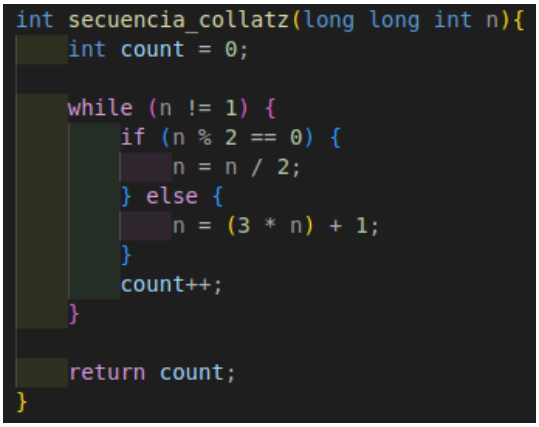
```c
int emisor_receptor[2];
int receptor_emisor[2];
char buf[30];
int status, pid;

status = pipe(emisor_receptor);
if (status == -1) {
    printf("ERROR al crear el pipe emisor_receptor");
    exit(1);
}

status = pipe(receptor_emisor);
if (status == -1) {
    printf("ERROR al crear el pipe receptor_emisor");
    exit(1);
}

if ((pid = fork()) == -1) {
    printf("ERROR al clonar el proceso");
    exit(1);
}

if (pid == 0) {
    close(emisor_receptor[WRITE]);
    close(receptor_emisor[READ]);

    while (1) {
        // Leer mensaje del proceso emisor
        read(emisor_receptor[READ], buf, sizeof(buf));
        printf("Receptor: recibido \"%s\"\n", buf);

        // Verificar si el mensaje es "exit"
        if (strcmp(buf, "terminar") == 0) {
            break;
        }

        // Enviar respuesta al proceso emisor
```

```
            write(receptor_emisor[WRITE], "Mensaje recibido", sizeof("Mensaje
recibido"));
        }

        close(emisor_receptor[READ]);
        close(receptor_emisor[WRITE]);
        printf("Receptor: termino\n");
        exit(0);
    } else {
        close(emisor_receptor[READ]);
        close(receptor_emisor[WRITE]);

        while (1) {
            // Leer mensaje desde la entrada estándar
            printf("Emisor: ingrese un mensaje ('terminar' para salir): ");
            fgets(buf, sizeof(buf), stdin);
            buf[strcspn(buf, "\n")] = '\0';  // Eliminar el salto de línea

            // Enviar mensaje al proceso receptor
            write(emisor_receptor[WRITE], buf, sizeof(buf));

            // Verificar si el mensaje es "exit"
            if (strcmp(buf, "terminar") == 0) {
                break;
            }

            // Leer respuesta del proceso receptor
            read(receptor_emisor[READ], buf, sizeof(buf));
            printf("Emisor: respuesta recibida \"%s\"\n", buf);
        }
        close(emisor_receptor[WRITE]);
        close(receptor_emisor[READ]);
        printf("Emisor: termino\n");
        wait(NULL);
    }
    return 0;
}
```

Actividad E

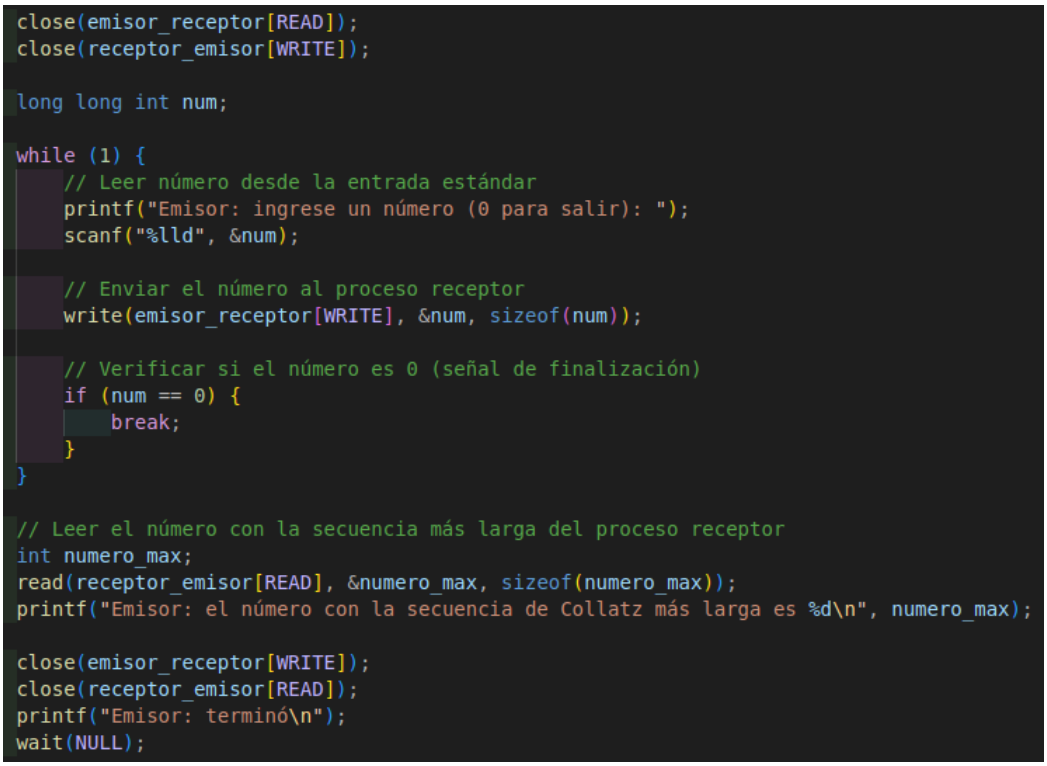
Esta actividad nos pide una versión del programa anterior para que dada una secuencia de números, encuentre aquel número cuya secuencia de Collatz sea la más larga.

1. Primero se crea la función con la que se calculará la secuencia de Collatz de un número dado.



```c
int secuencia_collatz(long long int n){
    int count = 0;

    while (n != 1) {
        if (n % 2 == 0) {
            n = n / 2;
        } else {
            n = (3 * n) + 1;
        }
        count++;
    }

    return count;
}
```

2. Para el proceso emisor primero se cierran los pipes que no se usarán. Se crea un ciclo que se termina una vez que se ha ingresado el número reservado; se pide el número a evaluar y se escribe dentro del pipe ***emisor_receptor*** y se verifica que el número ingresado no sea la palabra reservada para terminar la solicitud de números. Al final se lee del pipe ***receptor_emisor*** el número con la secuencia de Collatz más larga, y se cierran los pipes.



```c
close(emisor_receptor[READ]);
close(receptor_emisor[WRITE]);

long long int num;

while (1) {
    // Leer número desde la entrada estándar
    printf("Emisor: ingrese un número (0 para salir): ");
    scanf("%lld", &num);

    // Enviar el número al proceso receptor
    write(emisor_receptor[WRITE], &num, sizeof(num));

    // Verificar si el número es 0 (señal de finalización)
    if (num == 0) {
        break;
    }
}

// Leer el número con la secuencia más larga del proceso receptor
int numero_max;
read(receptor_emisor[READ], &numero_max, sizeof(numero_max));
printf("Emisor: el número con la secuencia de Collatz más larga es %d\n", numero_max);

close(emisor_receptor[WRITE]);
close(receptor_emisor[READ]);
printf("Emisor: terminó\n");
wait(NULL);
```

3. Para el proceso receptor se cierra los pipes que no usará; y se declaran las variables que se usarán para guardar el número con la secuencia de Collatz más larga. Luego se crea un ciclo para poder realizar operaciones hasta que se lea la palabra reservada para terminar; dentro de este ciclo se lee del pipe ***emisor_receptor*** el número que se procesará y se verifica si es la palabra reservada se termina, se asigna a una variable el valor de la secuencia de Collatz del número, y se evalúa si es la más larga hasta el momento en caso de serlo se guarda. Para terminar este proceso se escribe en el pipe ***receptor_emisor*** el número con la secuencia de Collatz más larga, y se cierran los pipes.

```
close(emisor_receptor[WRITE]);
close(receptor_emisor[READ]);

long long int num;
int longitud_max = 0;
int numero_max = 0;

while (1) {
    // Leer número del proceso emisor
    read(emisor_receptor[READ], &num, sizeof(num));

    // Verificar si el número es 0 (señal de finalización)
    if (num == 0) {
        break;
    }

    // Calcular la longitud de la secuencia de Collatz para el número
    int length = secuencia_collatz(num);

    // Verificar si la longitud actual es la máxima
    if (length > longitud_max) {
        longitud_max = length;
        numero_max = num;
    }
}
```

Enlace GDB

Código Fuente

```c
/*
** pipe.c -- un ejemplo de uso de pipe para
**            comunicar procesos por envío de mensajes
*/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <string.h>

#define WRITE 1
#define READ  0

int secuencia_collatz(long long int n);

int main() {
   int emisor_receptor[2];
   int receptor_emisor[2];
   char buf[30];
   int status, pid;

   status = pipe(emisor_receptor);
   if (status == -1) {
       printf("ERROR al crear el pipe emisor_receptor");
       exit(1);
   }

   status = pipe(receptor_emisor);
   if (status == -1) {
       printf("ERROR al crear el pipe receptor_emisor");
       exit(1);
   }

   if ((pid = fork()) == -1) {
       printf("ERROR al clonar el proceso");
       exit(1);
```

```
}

if (pid == 0) {
    close(emisor_receptor[WRITE]);
    close(receptor_emisor[READ]);

    long long int num;
    int longitud_max = 0;
    int numero_max = 0;

    while (1) {
        // Leer número del proceso emisor
        read(emisor_receptor[READ], &num, sizeof(num));

        // Verificar si el número es 0 (señal de finalización)
        if (num == 0) {
            break;
        }

        // Calcular la longitud de la secuencia de Collatz para el número
        int length = secuencia_collatz(num);

        // Verificar si la longitud actual es la máxima
        if (length > longitud_max) {
            longitud_max = length;
            numero_max = num;
        }
    }

    // Enviar el número con la secuencia más larga al proceso emisor
    write(receptor_emisor[WRITE], &numero_max, sizeof(numero_max));

    close(emisor_receptor[READ]);
    close(receptor_emisor[WRITE]);
    printf("Receptor: terminó\n");
    exit(0);
} else {
```

```
        close(emisor_receptor[READ]);
        close(receptor_emisor[WRITE]);

        long long int num;

        while (1) {
            // Leer número desde la entrada estándar
            printf("Emisor: ingrese un número (0 para salir): ");
            scanf("%lld", &num);

            // Enviar el número al proceso receptor
            write(emisor_receptor[WRITE], &num, sizeof(num));

            // Verificar si el número es 0 (señal de finalización)
            if (num == 0) {
                break;
            }
        }

        // Leer el número con la secuencia más larga del proceso receptor
        int numero_max;
        read(receptor_emisor[READ], &numero_max, sizeof(numero_max));
        printf("Emisor: el número con la secuencia de Collatz más larga es %d\n",
numero_max);

        close(emisor_receptor[WRITE]);
        close(receptor_emisor[READ]);
        printf("Emisor: terminó\n");
        wait(NULL);
    }

    return 0;
}

int secuencia_collatz(long long int n){
    int count = 0;
```

```
    while (n != 1) {
        if (n % 2 == 0) {
            n = n / 2;
        } else {
            n = (3 * n) + 1;
        }
        count++;
    }

    return count;
}
```