ARQUITECTURAS PARALELAS

3º CURSO – GRADO DE INGENIERÍA EN INFORMÁTICA (2023/2024)

PRÁCTICA 6

ALUMNOS:

VICTOR GONZALEZ DEL CAMPO

MARIO MARTINEZ LA FUENTE

Contenido PRACTICA 6

ENUNCIADO:

- Vamos a suponer el siguiente escenario para calcular el factorial de un número: el proceso 0 se encarga de recibir los números que le va introduciendo el usuario. El cálculo se lo encarga al proceso 1 que se supone que reside en una máquina con mayo r potencia de cálculo. Para evitar el colapso de las peticiones, deberá hacer esperar al usuario para que no introduzca otro dato hasta que se haya completado el envío ant erior. Para ello mostrará un mensaje de espera. Esta situación es poco probable con un hardware de una mínima potencia, pero la incluimos como ejemplo para **aprov echar** la potencia de las nuevas funciones. También será necesario incorporar una con dición de salida. En este caso, el envío del dato 0 indicará que se desea terminar la ope ración.

FUNCIONES UTILIZADAS:

```
int MPI_Isend(void *buf, int count, MPI_Datatype dtype, int dest, int
   tag, MPI_Comm comm, MPI_Request *request)

int MPI_Irecv(void *buf, int count, MPI_Datatype dtype, int source, int
   tag, MPI_Comm comm, MPI_Request *request)
```

```
int MPI_Wait(MPI_Request *request, MPI_Status *status)
```

```
int MPI_Test(MPI_Request *request, int *flag, MPI_Status *status)
```

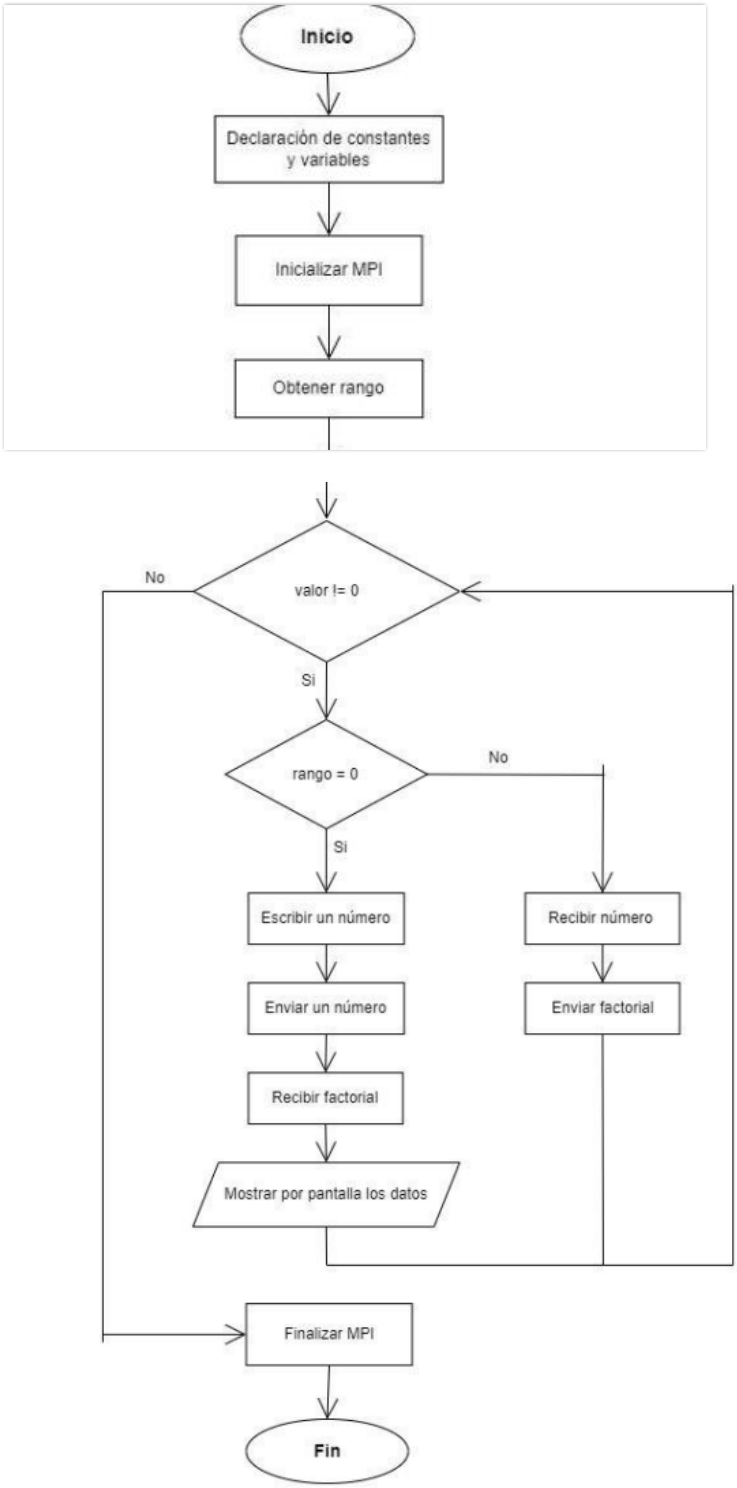
MPI_Isend: MPI_Isend es una función de la biblioteca MPI que se utiliza para enviar datos de manera asíncrona desde un proceso emisor a uno o varios procesos receptores en un entorno de cómputo paralelo. A diferencia de MPI_Send, MPI_Isend no bloquea la ejecución del proceso emisor, lo que significa que el programa puede continuar su ejecución sin esperar a que se complete la operación de envío. Esto es útil cuando se necesita superponer la comunicación con el cálculo.

MPI_Irecv: MPI_Irecv es la contraparte de MPI_Isend y se utiliza para recibir datos de manera asíncrona en un proceso receptor. Al igual que MPI_Isend, MPI_Irecv no bloquea la ejecución del proceso receptor y permite que el programa continúe su ejecución mientras se espera la llegada de los datos.

MPI_Wait: MPI_Wait es una función que se utiliza para esperar hasta que se completen todas las operaciones de envío y recepción asincrónica iniciadas previamente con MPI_Isend y MPI_Irecv. Es importante para garantizar que los datos estén disponibles para su procesamiento antes de continuar con otras partes del programa. MPI_Wait bloquea el proceso hasta que las operaciones asincrónicas se completen.

MPI_Test: MPI_Test es una función que se utiliza para verificar si las operaciones asincrónicas iniciadas con MPI_Isend y MPI_Irecv se han completado. A diferencia de MPI_Wait, MPI_Test no bloquea el proceso y devuelve un valor que indica si las operaciones asincrónicas se han completado o no. Esto permite a los programadores realizar otras tareas mientras esperan la finalización de las comunicaciones asincrónicas y verificar el estado de las mismas en momentos convenientes.

DIAGRAMA DE FLUJO :



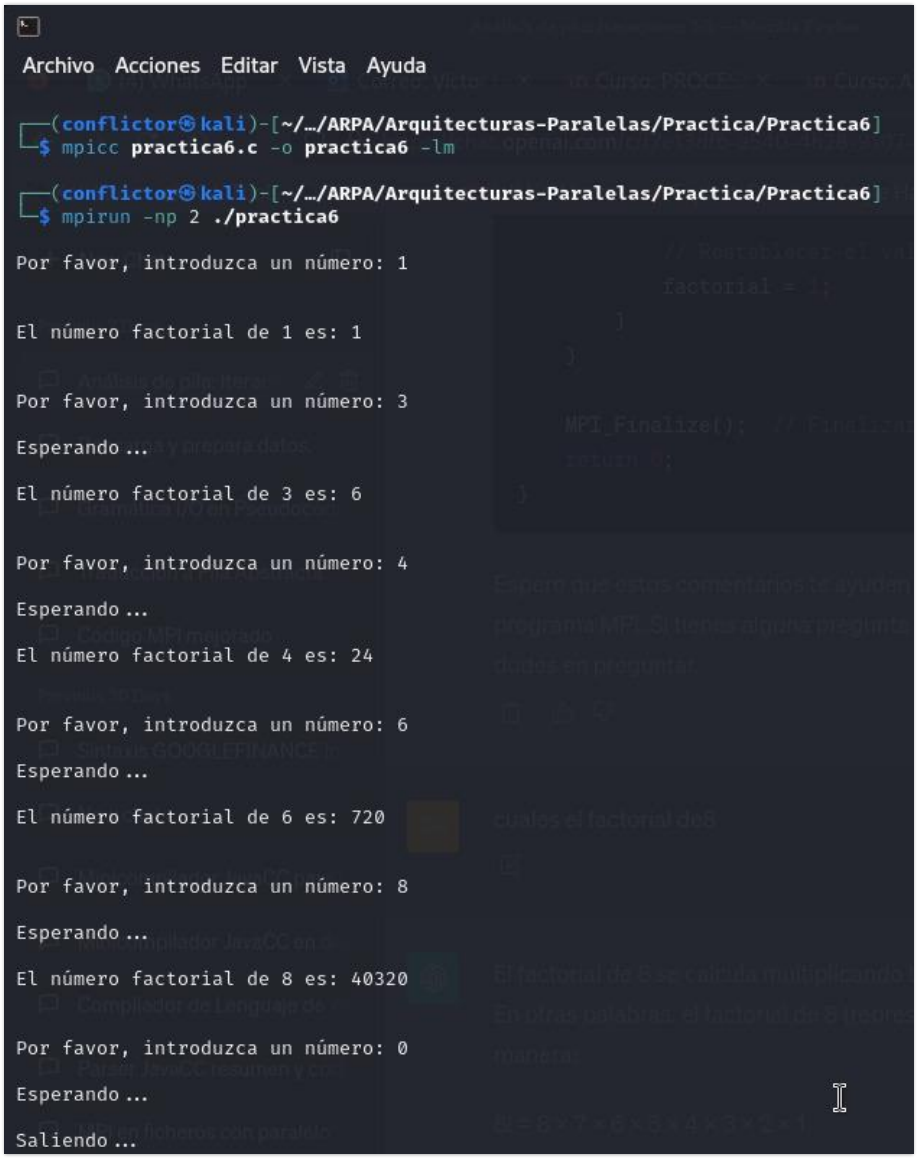
CODIGO FUENTE:

```c
#include <mpi.h>
#include <stdio.h>
int main(int argc, char* argv[]) {
    int mirango;                // Variable para almacenar el rango del proceso
    int numero = 1;             // Número ingresado por el usuario
    int factorial = 1;          // Resultado del cálculo del factorial
    int flag = 0;               // Bandera para verificar si la operación asincrónica ha terminado
    MPI_Status estado;          // Estructura para el estado de la comunicación
    MPI_Request referencia;     // Solicitud de comunicación asincrónica

    MPI_Init(&argc, &argv);      // Iniciar MPI
    MPI_Comm_rank(MPI_COMM_WORLD, &mirango);  // Obtener el rango del proceso

    // Bucle principal: continuar hasta que el usuario ingrese 0
    while (numero != 0) {
        if (mirango == 0) {  // Proceso con rango 0
            // Impresión por pantalla
            printf("\nPor favor, introduzca un número: ");
            fflush(stdout);
            scanf("%d", &numero);  // Leer un número del usuario

            if (numero == 0) {
                printf("\n\nSaliendo...");  // Mensaje de salida si el número es 0
                fflush(stdout);
            }

            // Envío y recepción de datos a otro proceso
            MPI_Isend(&numero, 1, MPI_INT, 1, 0, MPI_COMM_WORLD, &referencia);  // Recibir número del proceso con rango 0
            // Mensaje de espera hasta que la operación termine
            while (flag == 0) {
                MPI_Test(&referencia, &flag, &estado);
            }

            MPI_Irecv(&factorial, 1, MPI_INT, 1, 0, MPI_COMM_WORLD, &referencia);  // Iniciar la recepción asincrónica

            // Comprobación de la finalización de la recepción asincrónica
            MPI_Test(&referencia, &flag, &estado);  // Verificar si la operación ha terminado

            // Mensaje de espera hasta que la operación termine
            while (flag == 0) {
                printf("\nEsperando...");
                fflush(stdout);
                MPI_Test(&referencia, &flag, &estado);
            }


            // Salida de datos
            printf("\n\nEl número factorial de %d es: %d\n\n", numero, factorial);
            fflush(stdout);

        }

        if (mirango == 1) {  // Proceso con rango 1
            // Recepción del número enviado por el proceso 0
            MPI_Recv(&numero, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &estado);

            // Cálculo del factorial
            for (int i = 1; i <= numero; i++) {
                factorial *= i;
            }

            // Envío del resultado del factorial al proceso 0
```

```
63.             MPI_Send(&factorial, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
64.
65.             // Restablecer el valor de factorial para futuros cálculos
66.             factorial = 1;
67.         }
68.     }
69.
70.     MPI_Finalize();  // Finalizar MPI
71.     return 0;
72. }
73.
```

CAPTURAS EJECUCUION:

EJECUCION CON 2 PROCESOS:



RESUMEN CODIGO:

Se incluyen las bibliotecas MPI y stdio, se definen algunas variables, como mirango para el rango del proceso, numero para el número ingresado, factorial para el resultado del cálculo y flag para verificar la finalización de las operaciones asincrónicas. Además, se crean estructuras para el estado de la comunicación y solicitudes de comunicación asincrónica.

Se inicializa MPI y se obtiene el rango del proceso actual.

Se inicia un bucle que solicita al usuario un número hasta que se ingresa 0. El proceso con rango 0 (proceso principal) realiza las siguientes acciones:

Solicita al usuario un número.

Envía el número de manera asincrónica al proceso con rango 1.

Espera a que la operación de envío asincrónico termine utilizando MPI_Test.

Inicia una operación asincrónica para recibir el resultado del factorial del proceso 1.

El proceso con rango 1 recibe el número enviado por el proceso 0, calcula el factorial y envía el resultado de manera asincrónica al proceso 0.

Ambos procesos verifican la finalización de las operaciones asincrónicas utilizando MPI_Test y esperan hasta que la operación termine.

El proceso con rango 0 imprime el resultado del cálculo del factorial.

El programa finaliza MPI y devuelve 0.

En resumen, este programa utiliza MPI para realizar cálculos asincrónicos del factorial de un número entre dos procesos, mostrando cómo se pueden enviar y recibir datos de manera asincrónica en un entorno de cómputo paralelo.

OBSERVACIONES :

Cuando el programa se ejecuta solo llega a resolver hasta el número 12, cuando se intenta el número 13 se imprime de forma incorrecta. Esto es debido a que MPI está limitado, de todas formas, el número factorial está declarado como un integer y también podría limitarlo, sin embargo, si se cambia por un long pasa lo mismo.

No hace falta calcular el tiempo de ejecución en paralelo en este caso porque el cálculo del factorial se realiza secuencialmente dentro de cada proceso. La comunicación asincrónica entre los procesos permite que continúen trabajando en otras tareas mientras esperan los resultados de los demás. Sin embargo, dado que el cálculo del factorial en sí mismo no se ha paralelizado ni se ha distribuido entre los procesos, no habría una mejora significativa en el tiempo de ejecución si se midiera el tiempo de manera paralela. La medición del tiempo de ejecución generalmente se utiliza para evaluar el rendimiento de operaciones paralelas o distribuidas, y en este caso, no hay una concurrencia real en el cálculo del factorial en sí.

CUESTIONES :

• ¿De qué manera se puede aprovechar la potencia de las instrucciones vistas en esta práctica para evitar que los procesos trabajen en ocasiones con datos obsoletos?

- Para poder aprovechar mejor la potencia de las instrucciones deberíamos utilizar bloqueos que hagan que hasta que no se envíen todos los datos, no continúe el proceso para que de esta forma no llegase a poder sobrescribirse dichos datos y se llegasen a quedar obsoletos. Al hacer esto creamos un retraso que hace que tarde más tiempo, pero nos aseguramos que los datos se escriban de forma correcta

• ¿Se podría producir una situación de abrazo mortal por estar todos los procesos en curso bloqueados en espera de que se complete una petición?

- Sí, debido a que puede ocasionarse que, un proceso este enviando un dato otro proceso, y a su vez, este esté enviando otro dato a este otro proceso, por lo que los dos estarían esperando a que se reciba su dato, pero esto nunca ocurrirá porque en ningún momento ninguno de los dos recibirá ninguno.

• Realizar una reflexión sobre el concepto de abrazo mortal indicando cómo afecta a los diferentes modos de envío que se conocen.

1. MPI_Send y MPI_Recv:
    1. En el caso de MPI_Send y MPI_Recv, un abrazo mortal puede ocurrir si un proceso espera recibir datos (MPI_Recv) antes de que otro proceso haya enviado los datos correspondientes (MPI_Send). Si ambos procesos esperan recibir antes de enviar, se bloquearán indefinidamente, ya que ninguno de ellos enviará datos hasta que los reciba.
    2. También puede ocurrir un abrazo mortal si la cantidad de datos enviados (MPI_Send) no coincide con la cantidad de datos que se espera recibir (MPI_Recv). Si un proceso envía más o menos datos de los que se esperan, los procesos pueden bloquearse.
2. MPI_Isend y MPI_Irecv:
    1. Con MPI_Isend y MPI_Irecv, el riesgo de un abrazo mortal se mantiene presente si los procesos no coordinan adecuadamente sus operaciones. La

principal ventaja de MPI_Isend y MPI_Irecv es que permiten operaciones asincrónicas, lo que puede complicar aún más la sincronización.