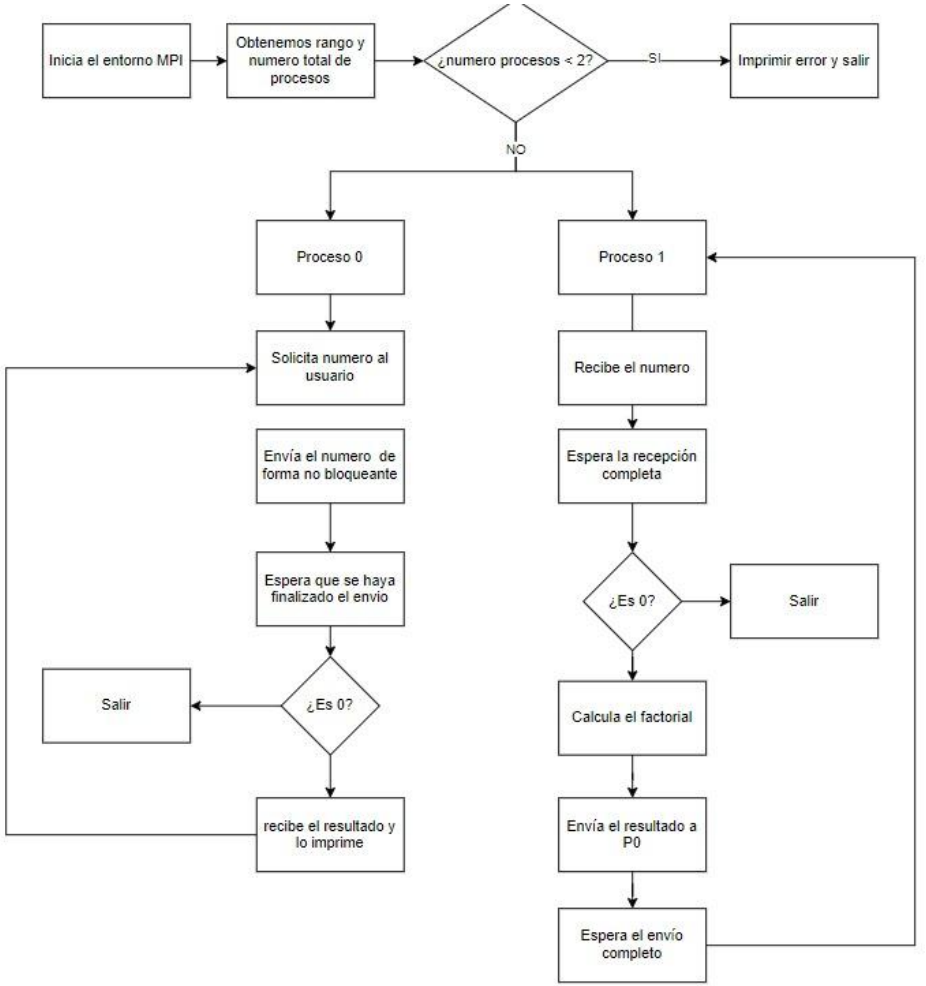
Práctica 6: Nuevos modos de envío

DIEGO URBANEJA
HUGO GÓMEZ
NICOLÁS VILLANUEVA

ÍNDICE

- FLUJOGRAMA
- CÓDIGO DEL PROGRAMA
- EJECUCIÓN Y SALIDA POR PANTALLA
- CUESTIONES PLANTEADAS

FLUJOGRAMA


Inicia el entorno MPI
Obtenemos rango y numero total de procesos
¿numero procesos < 2?
SI
Imprimir error y salir
NO
Proceso 0
Solicita numero al usuario
Envía el numero de forma no bloqueante
Espera que se haya finalizado el envio
¿Es 0?
Salir
recibe el resultado y lo imprime
Proceso 1
Recibe el numero
Espera la recepción completa
¿Es 0?
Salir
Calcula el factorial
Envía el resultado a P0
Espera el envío completo

CÓDIGO DEL PROGRAMA

```c
#include <mpi.h>
#include <stdio.h>

// Funcion para calcular el factorial de un numero
long double factorial(int n) {
    long double result = 1;
    for (int i = 1; i <= n; ++i) {
        result *= i;
    }
    return result;
}

int main(int argc, char* argv[]) {
    int rank, size;
    int length;
    char name[32];
    int flag;

    // Inicializar el entorno MPI
    MPI_Init(&argc, &argv);
    // Obtener el rango del proceso
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    // Obtener el numero total de procesos
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    // Obtener el nombre del procesador
    MPI_Get_processor_name(name, &length);

    // Asegurarse de que haya al menos 2 procesos
    if (size < 2) {
        if (rank == 0) {
            printf("[Maquina % s] > Proceso %d: Se necesitan al menos 2 procesos para esta tarea.\n",name, rank);
            fflush(stdout);

        }
        MPI_Finalize();
        return 0;
    }
```

```c
    if (rank == 0) { // Proceso 0: Maneja la entrada del usuario y envia los datos al proceso 1
        int num;
        MPI_Request request;
        MPI_Status status;
        long double result;

        while (1) {
            printf("[Maquina % s] > Proceso %d: Ingrese un número para calcular su factorial (0 para terminar): ",name, rank);
            fflush(stdout);
            scanf_s("%d", &num);

            // Enviar el número de forma no bloqueante
            MPI_Isend(&num, 1, MPI_INT, 1, 0, MPI_COMM_WORLD, &request);

            // Esperar hasta que el envío esté completo antes de env
            MPI_Wait(&request, &status);

            // Si el usuario ingresa 0, terminar el programa
            if (num == 0) break;

            //Imprimimos el mensaje de espera
            printf("Número enviado, esperando el resultado...\n");
            fflush(stdout);

            //Cuando recibamos el cálculo del factorial dese el proceso 1 mostramos el resultado
            MPI_Recv(&result, 1, MPI_LONG_DOUBLE, 1, 0, MPI_COMM_WORLD, &status);
            printf("[Maquina %s] > Proceso 0: Factorial de %d es %.0Lf\n", name, num, result);
            fflush(stdout);
```

Se usa **MPI_Isend** para enviar el número al proceso 1 de forma no bloqueante y espera a que el envío se complete con **MPI_Wait**.

Tras enviar el número, espera el resultado del proceso 1 mediante **MPI_Recv**, el cual recibe el factorial calculado y se muestra el resultado en pantalla.

CÓDIGO DEL PROGRAMA

```
69      else if (rank == 1) { // Proceso 1: Recibe los datos y calcula el factorial
70          int num;
71          long double result;
72          MPI_Request request;
73          MPI_Status status;
74          while (1) {
75              // Recepción no bloqueante
76              MPI_Irecv(&num, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &request);
77
78              // Comprobar si la recepción está completa
79              do {
80                  MPI_Test(&request, &flag, &status);
81                  // Realiza otras operaciones aquí si fuera necesario mientras espera
82              } while (!flag);  // Esperar hasta que la recepción se complete
83
84              // Si el numero recibido es 0, terminar el programa
85              if (num == 0) break;
86
87              //Cáluclo del factorial
88              result = factorial(num);
89
90              //Enviar datos al proceso 0
91              MPI_Isend(&result, 1, MPI_LONG_DOUBLE, 0, 0, MPI_COMM_WORLD, &request);
92              MPI_Wait(&request, &status);
93
94          }
95      }
96
97      // Finalizar el entorno MPI
98      MPI_Finalize();
99      return 0;
100 }
101
```

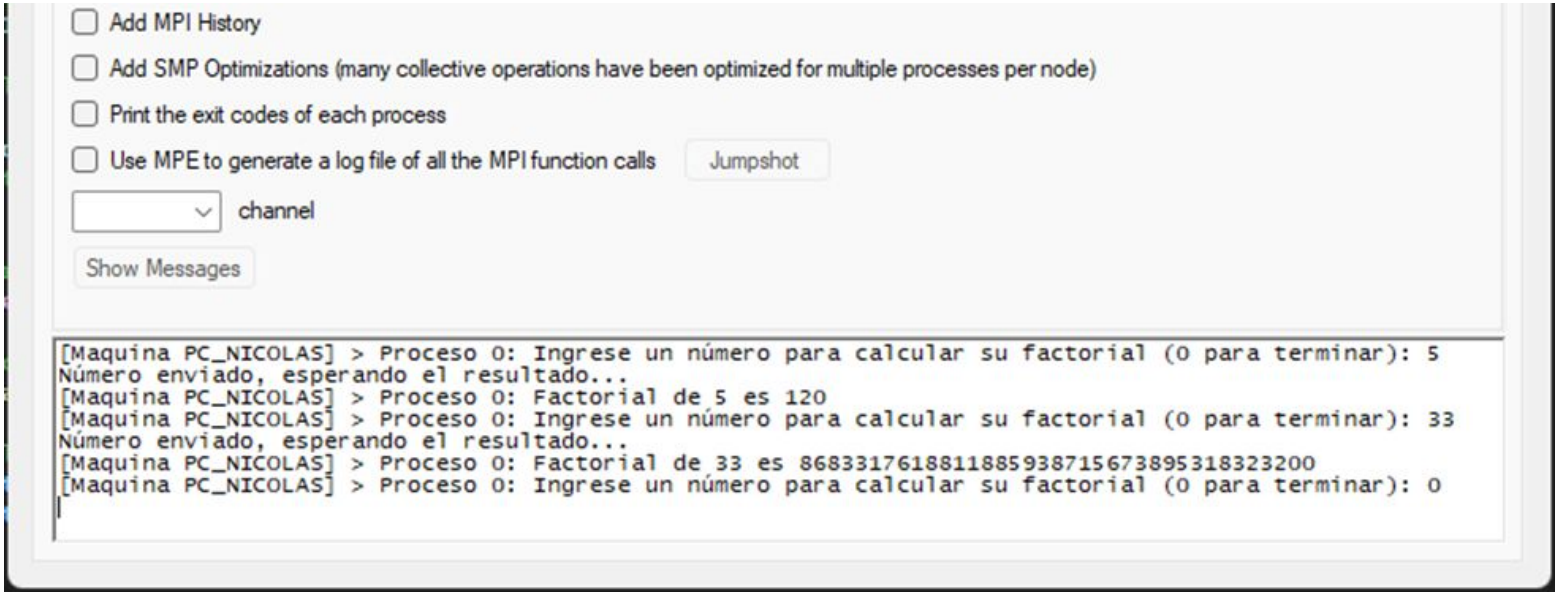
Utiliza **MPI_Irecv** para recibir el número de forma no bloqueante.

Con **MPI_Test**, verifica repetidamente si la recepción ha terminado, permitiendo realizar otras tareas en el mismo bucle si fuese necesario.

Usa **MPI_Isend** para enviar el resultado de vuelta al proceso maestro y espera a que se complete con **MPI_Wait**.

Se finaliza el entorno MPI y termina el programa.

EJECUCIÓN Y SALIDA POR PANTALLA



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?**

  Utilizando operaciones de envío y recepción no bloqueantes como **MPI_Isend** y **MPI_Irecv**, los procesos pueden iniciar comunicaciones y continuar ejecutando otras tareas mientras se completa la transferencia de datos. Además, mediante el uso de funciones de finalización como **MPI_Wait** o **MPI_Test**, se garantiza que los datos se han recibido completamente antes de utilizarlos. Esto asegura que los procesos trabajen siempre con datos actualizados, evitando el uso de información obsoleta.

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

  En nuestra práctica con solo dos procesos y una comunicación bien definida (Proceso 0 envía y luego recibe, Proceso 1 recibe y luego envía), es poco probable que ocurra un abrazo mortal. Sin embargo, si se expandiera a más procesos con comunicaciones circulares sin una correcta sincronización, podría surgir un deadlock.

CUESTIONES

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

  El abrazo mortal ocurre cuando dos o más procesos quedan bloqueados esperando mutuamente que el otro complete una operación. En modos de envío bloqueantes (**`MPI_Send`**, **`MPI_Recv`**), es más fácil incurrir en deadlocks si no se coordina adecuadamente el orden de envíos y recepciones. Las operaciones no bloqueantes (**`MPI_Isend`**, **`MPI_Irecv`**) reducen este riesgo al permitir que los procesos continúen ejecutándose mientras las comunicaciones se completan, siempre que se gestionen correctamente las finalizaciones con **`MPI_Wait`** o **`MPI_Test`**. En esta práctica, el uso de operaciones no bloqueantes ayuda a evitar abrazos mortales al permitir una comunicación más flexible y eficiente entre los procesos.