UNIVERSIDAD DE BURGOS

Grado en Ingeniería Informática

Arquitecturas Paralelas

Curso 2024-2025

Practica 6

NUEVOS MODOS DE ENVÍO

AUTORES

Diego Urbaneja Portal

Hugo Gómez Martín

Nicolás Villanueva Ortega

ÍNDICE

Introducción

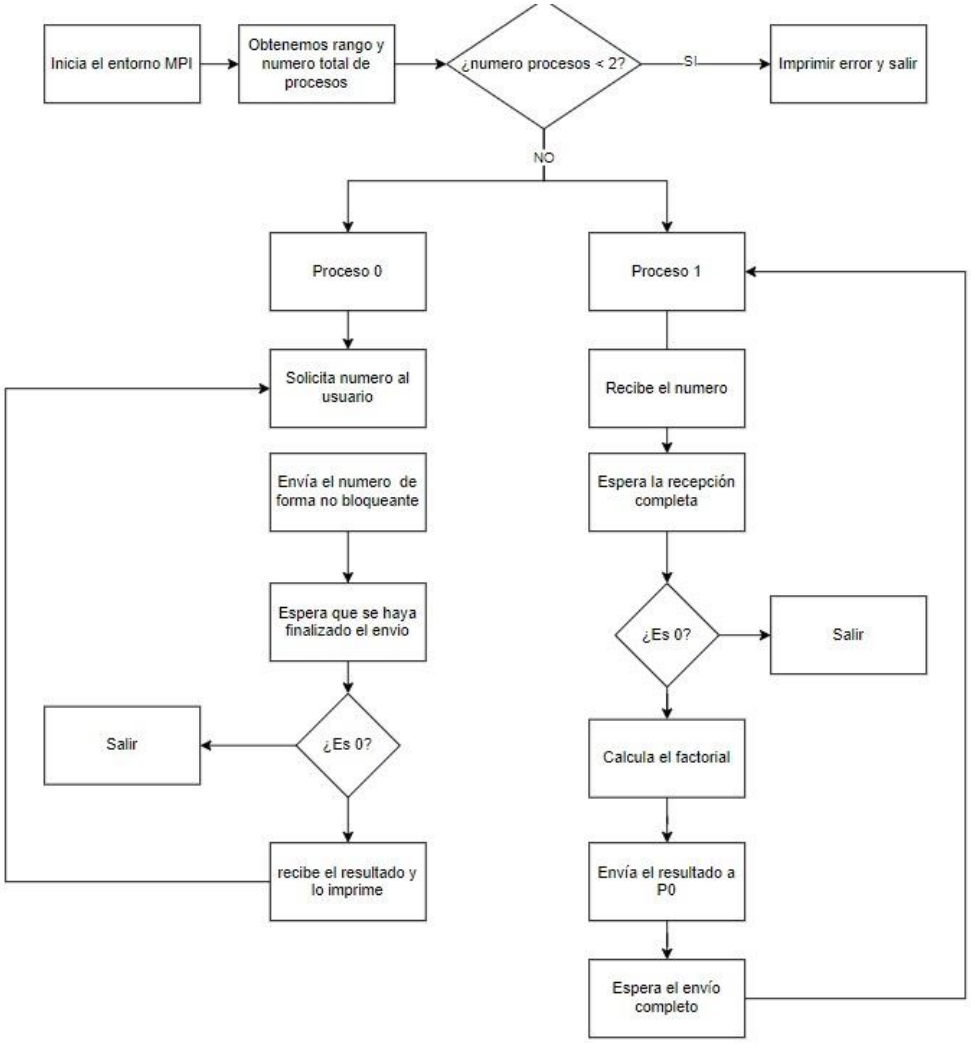
Esta práctica se centra en ampliar el conocimiento sobre los modos de envío en MPI, principalmente explorando los modos de envío no bloqueantes (**MPI_Isend** y **MPI_Irecv**), que permiten que el proceso continúe su ejecución sin esperar a que se complete la operación de comunicación. Esta práctica explica cómo iniciar una operación y luego finalizarla, con las funciones de espera (**MPI_Wait**) o prueba (**MPI_Test**), para verificar si la operación ha finalizado sin bloquear el proceso.

La implementación propuesta es la siguiente:

1. **Escenario**: Un proceso (0) recibe números del usuario y los envía al proceso 1, encargado de calcular el factorial.
2. **Condiciones de Salida**: El envío del dato 0 indica el fin de la operación.
3. **Gestión de Espera**: Para evitar que el usuario envíe datos antes de tiempo, el proceso muestra un mensaje de espera.

Recomendada la visualización del documento con un 140% - 150% de zoom.

Flujograma



El flujograma ilustra un programa en MPI diseñado para calcular el factorial de un número utilizando dos procesos (Proceso 0 y Proceso 1). La secuencia es la siguiente:

1. **Inicialización y Validación**: Se inicia el entorno MPI y se verifica si hay al menos dos procesos; de lo contrario, el programa muestra un error y termina.
2. **Proceso 0**:
    - Solicita al usuario un número y lo envía de forma no bloqueante a Proceso 1.
    - Espera la confirmación de envío y verifica si el número es 0 para terminar en ese caso.
    - Recibe el resultado del factorial calculado y lo imprime.
3. **Proceso 1**:
    - Recibe el número enviado por Proceso 0 y verifica si es 0 para finalizar en ese caso.
    - Calcula el factorial y lo envía de vuelta a Proceso 0, esperando la confirmación de envío antes de concluir.

Este flujo de trabajo en MPI permite una colaboración eficiente entre ambos procesos para el cálculo distribuido del factorial.

Código

```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;
    }
```

En este primer bloque de código se incluyen las librerías de MPI y las de manejo de entrada y salida en C.

También se incluye la función que nos servirá posteriormente para calcular el factorial en el proceso 1. Esta devuelve un long double tras introducirle un int.

A continuación, comienza el MAIN donde tenemos declaraciones de variables globales que emplearemos posteriormente, iniciamos el entorno MPI, obtenemos valores de rangos, y nombre de la máquina anfitriona.

Por último, realizamos la comprobación para ejecutar el programa con mínimo 2 procesos, que son los que necesitamos para el cálculo

En este segundo bloque vemos el código que se ejecuta en el **Proceso 0**. Al comienzo se declaran unas variables necesarias para el funcionamiento de las funciones empleadas.

```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 enviar
            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 solicita un número entero para calcular su factorial.

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**.

Si el usuario ingresa 0, termina el programa.

Mientras se esta realizando el cálculo se muestra un mensaje de espera.

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.

A continuación, apreciamos el código que se ejecuta en el **Proceso 1**. En teoría, este proceso es el que se encuentra alojado en una máquina de mayor potencia y el que se encarga de realizar los cálculos.

```
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
82              } while (!flag);  // Esperar hasta que l
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      }
```

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.

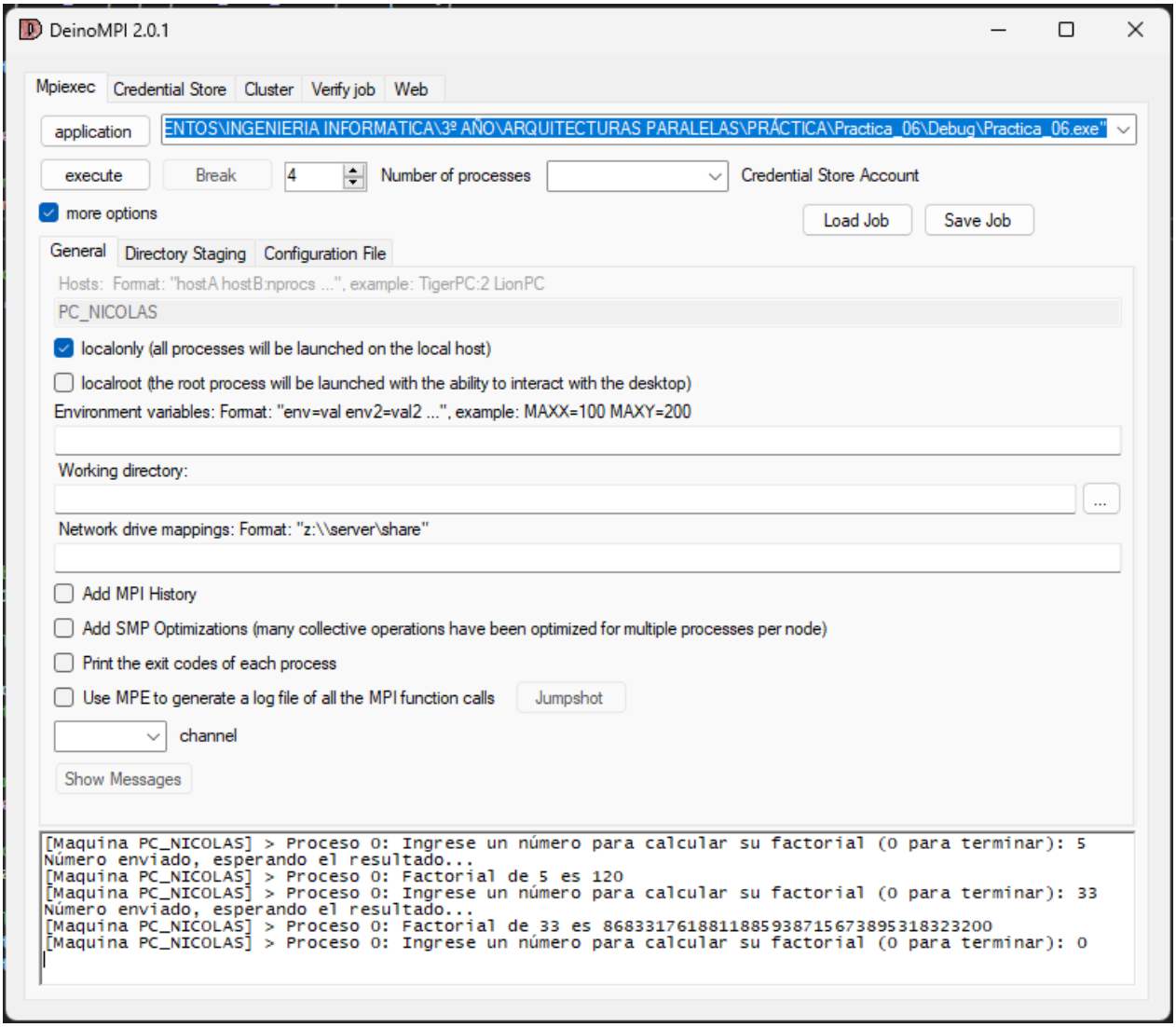
Al recibir el número, calcula el factorial con la función **factorial**()

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

Por último, finalizamos en entorno MPI y termina el programa.

```
96
97          // Finalizar el entorno MPI
98          MPI_Finalize();
99          return 0;
100     }
101
```

Salida por Pantalla



Como se puede ver, el factorial se calcula correctamente. No permite al usuario introducir otro número hasta que se halla terminado el cálculo del anterior, ya sean factoriales de números pequeños o grandes.

Al introducir el 0 el programa finaliza correctamente.

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.

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.

Bibliografía

Hemos obtenido la información para la realización de la práctica del **guion** proporcionado por el profesor.

Para la resolución de dudas con el código hemos empleado chat **GPT o1-mini**.