



# UNIVERSIDAD DEBURGOS

Grado en Ingeniería Informática  
**Arquitecturas Paralelas**  
Curso 2024-2025

## **Practica 8**

### **GESTIÓN DINÁMICA DE PROCESOS**

#### **AUTORES**

**Diego Urbaneja Portal**  
**Hugo Gómez Martín**  
**Nicolás Villanueva Ortega**

# ÍNDICE

<b>Introducción.....</b>	<b>3</b>
<b>Flujograma .....</b>	<b>4</b>
<b>Código.....</b>	<b>5</b>
<b>Salida por Pantalla.....</b>	<b>8</b>
<b>Cuestiones.....</b>	<b>9</b>
<b>Bibliografía .....</b>	<b>9</b>

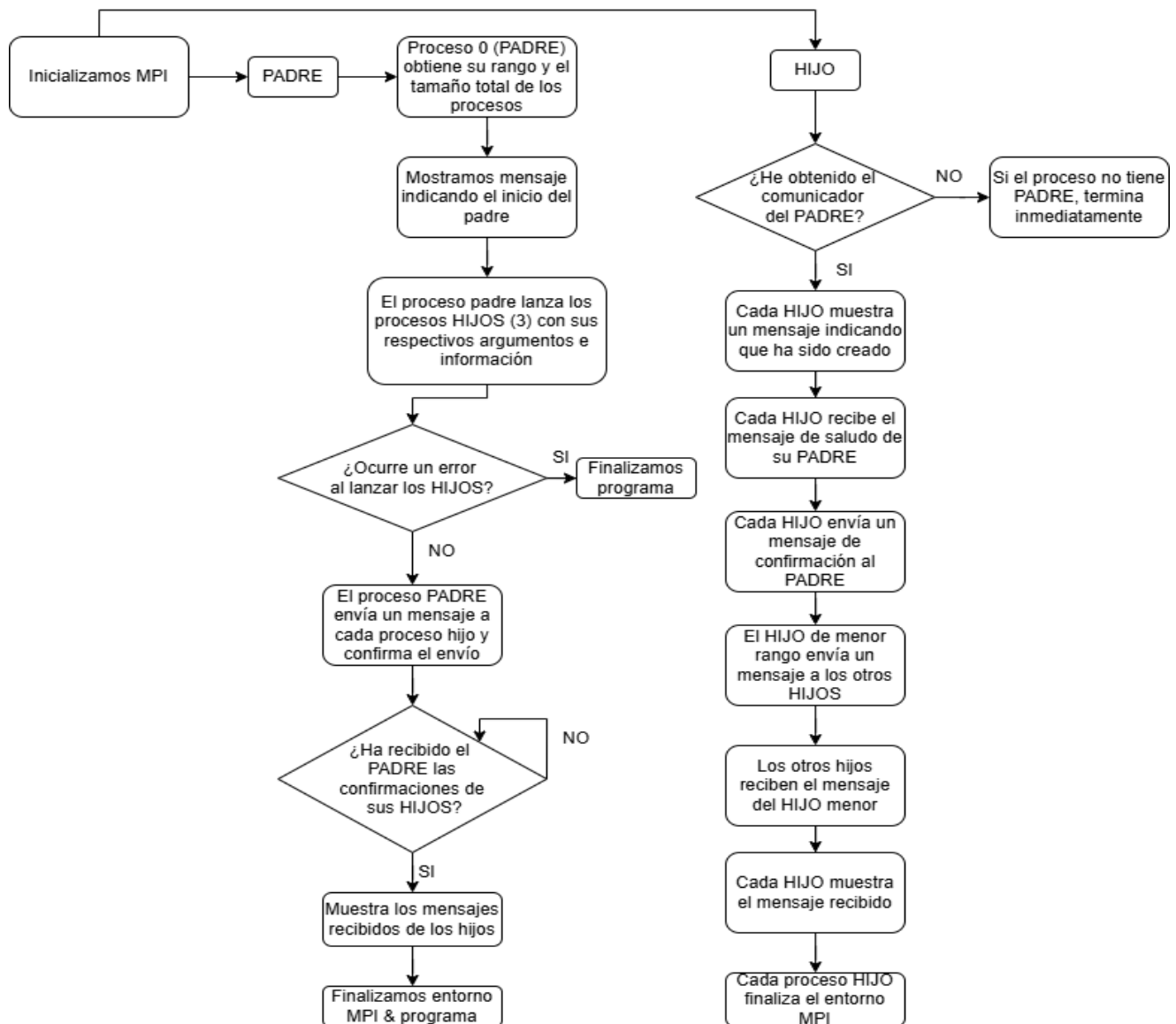
# **Introducción**

Esta práctica se centra en **explorar y aplicar técnicas de configuración dinámica de clústeres**, simulando un entorno de máquina virtual. El objetivo principal es **lograr que un proceso padre pueda iniciar procesos hijo durante la ejecución**, permitiendo el manejo dinámico de tareas y adaptándose a los recursos disponibles en el clúster.

En términos teóricos, **la práctica introduce los conceptos de gestión dinámica de procesos y de tolerancia a fallos**, destacando las limitaciones actuales de MPI en estos aspectos en comparación con sistemas como PVM, que ofrecen una mayor flexibilidad en el manejo de nodos y la resistencia a fallos

Recomendada la visualización del documento con un 120% - 130% de zoom.

# Flujograma



**Inicializamos el programa con el proceso 0** que en esta práctica será el padre. Este obtiene su rango y el tamaño total de procesos. El padre **muestra un mensaje indicando que se ha iniciado**. Este proceso **es el encargado de lanzar los respectivos hijos**, en nuestro caso hemos establecido estáticamente 3.

**Si ocurre algún error** al lanzar los hijos debido a que no hay una comunicación, **mostramos error y finalizamos** el programa, si no ocurre, el proceso padre envía un mensaje a cada proceso hijo y confirma el envío.

**Los procesos hijos verifican si tienen padre, si no lo tienen termina inmediatamente, si tienen padre cada hijo muestra un mensaje indicando que ha sido creado**, cada hijo recibe el mensaje de saludo de su padre y envían un mensaje de confirmación.

**Además, antes de finalizar el hijo menor envía un mensaje a sus hijos mayores y estos lo muestran**, después finalizan completamente los procesos hijos. Una vez el padre ha recibido las confirmaciones de sus hijos, muestra los mensajes recibidos y finaliza el entorno MPI y el programa.

# Código

Para esta práctica se necesitan 2 programas diferentes, uno que hace referencia al padre que es el encargado de lanzar y crear los hijos, y otro que son el código de los propios hijos en sí.

Nosotros hemos colocado ambos programas en la misma solución, pero se encuentran en archivos .cpp distintos, de esta manera se compilarán los dos a la vez a la hora de ejecutarlos.

## Proceso Padre

```
// parent.cpp
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define num_hijos 3

int main(int argc, char* argv[])
{
    int mirango, tamano;
    int longitud;
    char nombre[32];
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &mirango);
    MPI_Comm_size(MPI_COMM_WORLD, &tamano);
    MPI_Get_processor_name(nombre, &longitud);

    if (mirango == 0) { // Solo el proceso con rango 0 actúa como padre
        printf("PADRE> Proceso %d de %d: Iniciando proceso padre.\n", mirango, tamano);
        fflush(stdout);

        // Argumentos para los hijos (pueden ser MPI_ARGV_NULL si no se necesitan)
        char** argumentos_hijos = MPI_ARGV_NULL;

        // Información sobre dónde y cómo lanzar los hijos (MPI_INFO_NULL por defecto)
        MPI_Info info;
        MPI_Info_create(&info);

        MPI_Comm intercom;
        int errcodes[num_hijos];

        // Definir el comando como un arreglo de char no constante
        char child_command[] = "C:\\Users\\urban\\DIEGO\\UBU 4º\\1ºSemestre\\Arquitecturas Paralelas\\PRACTICAS\\P8\\x64\\Debug\\HIJO.exe"; // Ruta absoluta

        // Lanzar procesos hijos
        int resultado = MPI_Comm_spawn(child_command, argumentos_hijos, num_hijos, info, 0, MPI_COMM_WORLD, &intercom, errcodes);

        if (resultado != MPI_SUCCESS) {
            fprintf(stderr, "Error al lanzar los procesos hijos.\n");
            MPI_Abort(MPI_COMM_WORLD, resultado);
        }

        printf("PADRE> Proceso %d: Se han lanzado %d procesos hijos.\n", mirango, num_hijos);
        fflush(stdout);

        // Enviar mensajes de saludo a los hijos
        char saludo[] = "Hola desde el proceso padre!";
        for (int i = 0; i < num_hijos; i++) {
            MPI_Send(saludo, strlen(saludo) + 1, MPI_CHAR, i, 0, intercom);
            printf("PADRE> Proceso %d: Enviado saludo al hijo %d.\n", mirango, i);
            fflush(stdout);
        }

        // Recibir mensajes de confirmación de los hijos
        for (int i = 0; i < num_hijos; i++) {
            char mensaje_recibido[100];
            MPI_Recv(mensaje_recibido, 100, MPI_CHAR, MPI_ANY_SOURCE, MPI_ANY_TAG, intercom, MPI_STATUS_IGNORE);
            printf("PADRE> Proceso %d: Recibido mensaje de hijo: %s\n", mirango, mensaje_recibido);
            fflush(stdout);
        }

        MPI_Info_free(&info);
    }
    else {
        // Otros procesos (si los hay) pueden realizar otras tareas
        printf("PADRE> Proceso %d de %d: No es el proceso padre.\n", mirango, tamano);
        fflush(stdout);
    }

    MPI_Finalize();
    return 0;
}
```

El proceso padre comienza inicializando el entorno MPI y determinando su rango dentro del comunicador global MPI\_COMM\_WORLD. **Solo el proceso con rango 0 actúa como padre**, mientras que cualquier otro proceso puede realizar tareas adicionales o simplemente anunciar que no es el padre.

Una vez identificado como padre, el proceso **imprime un mensaje de inicio y procede a lanzar tres procesos hijos utilizando la función MPI\_Comm\_spawn**. Esta función toma como parámetros la ruta al ejecutable del hijo (HIJO.exe), los argumentos a pasar (en este caso, ninguno), el número de hijos a lanzar, y la información adicional de lanzamiento. **Si la función MPI\_Comm\_spawn falla, el padre imprime un mensaje de error y aborta la ejecución**.

Tras lanzar los hijos, el padre envía un mensaje de saludo a cada uno de ellos a través del comunicador intercomunicador intercom. Utiliza un bucle para enviar el saludo "¡Hola desde el proceso padre!" a cada hijo individualmente. Después de enviar los saludos, el padre espera recibir confirmaciones de cada hijo. **Utiliza otro bucle para recibir mensajes de confirmación**, asegurándose de que ha recibido un mensaje de cada uno de los hijos lanzados.

Finalmente, **el padre libera el objeto MPI\_Info utilizado para el lanzamiento y finaliza el entorno MPI**.

### Proceso Hijo

```
// child.cpp
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char* argv[])
{
    int mirango, tamano;
    int longitud;
    char nombre[32];

    MPI_Init(&argc, &argv);

    MPI_Comm parent;
    MPI_Comm_get_parent(&parent);

    MPI_Comm_rank(MPI_COMM_WORLD, &mirango);
    MPI_Comm_size(MPI_COMM_WORLD, &tamano);
    MPI_Get_processor_name(nombre, &longitud);

    if (parent == MPI_COMM_NULL) {
        printf("HIJO> Proceso %d: No tengo padre. Finalizando.\n", mirango);
        fflush(stdout);
        MPI_Finalize();
        return 0;
    }

    // Enviar mensaje de saludo al ser creado
    printf("HIJO> Proceso %d: Hijo creado. ¡Hola!\n", mirango);
    fflush(stdout);

    // Recibir mensaje del padre
    char mensaje_padre[100];
    MPI_Recv(mensaje_padre, 100, MPI_CHAR, 0, MPI_ANY_TAG, parent, MPI_STATUS_IGNORE);
    printf("HIJO> Proceso %d: Recibido del padre: %s\n", mirango, mensaje_padre);
    fflush(stdout);
}
```

```

// Enviar mensaje de confirmación al padre
char respuesta_padre[] = "Mensaje recibido por el hijo.";
MPI_Send(respuesta_padre, strlen(respuesta_padre) + 1, MPI_CHAR, 0, 0, parent);
printf("HIJO> Proceso %d: Enviado confirmación al padre.\n", mirango);
fflush(stdout);

// Comunicación entre hijos usando MPI_COMM_WORLD
if (mirango == 0) {
    // El hijo de menor rango envía mensajes a los demás hijos
    char mensaje_hijos[] = "Mensaje del hijo de menor rango.";
    for (int i = 1; i < tamano; i++) {
        MPI_Send(mensaje_hijos, strlen(mensaje_hijos) + 1, MPI_CHAR, i, 1, MPI_COMM_WORLD);
        printf("HIJO> Proceso %d: Enviado mensaje a hijo %d.\n", mirango, i);
        fflush(stdout);
    }
} else {
    // Los hijos que no son el de menor rango reciben mensajes del hijo 0
    char mensaje_hijo[100];
    MPI_Recv(mensaje_hijo, 100, MPI_CHAR, 0, 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    printf("HIJO> Proceso %d: Recibido del hijo de menor rango: %s\n", mirango, mensaje_hijo);
    fflush(stdout);
}

MPI_Finalize();
return 0;
}

```

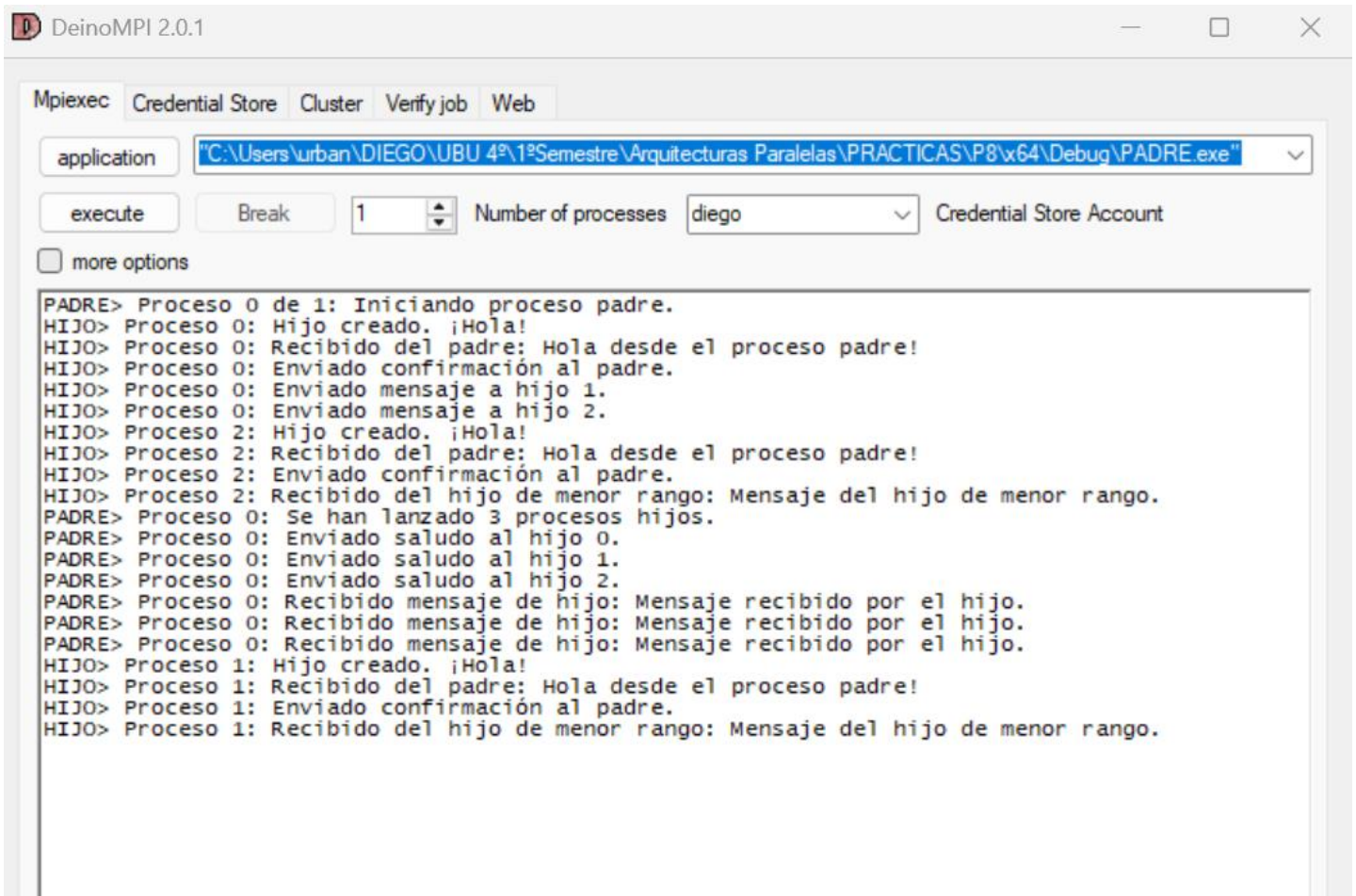
Cada proceso hijo, al iniciarse, también inicializa el entorno MPI y obtiene el comunicador con su padre mediante `MPI_Comm_get_parent`. Si el proceso hijo no tiene un padre (es decir, si `MPI_Comm_get_parent` devuelve `MPI_COMM_NULL`), imprime un mensaje y finaliza.

Una vez verificado que tiene un padre, **el hijo anuncia su creación y espera recibir un mensaje de saludo del padre**. Tras recibir el saludo, **el hijo envía una confirmación** de recepción de vuelta al padre, asegurando así que el padre puede verificar la recepción del saludo.

A continuación, se establece una comunicación entre los hijos utilizando `MPI_COMM_WORLD`, que en el contexto de los hijos incluye a todos los procesos hijos. **El hijo de menor rango** (`mirango == 0`) **envía un mensaje** a cada uno de los otros hijos indicando "Mensaje del hijo de menor rango." Por otro lado, **los hijos que no son de menor rango esperan** a recibir este mensaje y lo imprimen en pantalla.

Finalmente, cada hijo finaliza el entorno MPI.

# Salida por Pantalla



```
PADRE> Proceso 0 de 1: Iniciando proceso padre.  
HIJO> Proceso 0: Hijo creado. ¡Hola!  
HIJO> Proceso 0: Recibido del padre: Hola desde el proceso padre!  
HIJO> Proceso 0: Enviado confirmación al padre.  
HIJO> Proceso 0: Enviado mensaje a hijo 1.  
HIJO> Proceso 0: Enviado mensaje a hijo 2.  
HIJO> Proceso 2: Hijo creado. ¡Hola!  
HIJO> Proceso 2: Recibido del padre: Hola desde el proceso padre!  
HIJO> Proceso 2: Enviado confirmación al padre.  
HIJO> Proceso 2: Recibido del hijo de menor rango: Mensaje del hijo de menor rango.  
PADRE> Proceso 0: Se han lanzado 3 procesos hijos.  
PADRE> Proceso 0: Enviado saludo al hijo 0.  
PADRE> Proceso 0: Enviado saludo al hijo 1.  
PADRE> Proceso 0: Enviado saludo al hijo 2.  
PADRE> Proceso 0: Recibido mensaje de hijo: Mensaje recibido por el hijo.  
PADRE> Proceso 0: Recibido mensaje de hijo: Mensaje recibido por el hijo.  
PADRE> Proceso 0: Recibido mensaje de hijo: Mensaje recibido por el hijo.  
HIJO> Proceso 1: Hijo creado. ¡Hola!  
HIJO> Proceso 1: Recibido del padre: Hola desde el proceso padre!  
HIJO> Proceso 1: Enviado confirmación al padre.  
HIJO> Proceso 1: Recibido del hijo de menor rango: Mensaje del hijo de menor rango.
```

En la ejecución del programa MPI, el proceso padre inicia correctamente y lanza tres procesos hijos. Cada hijo, al crearse, recibe un mensaje de saludo del padre y envía una confirmación de vuelta, lo que asegura la correcta recepción del mensaje inicial. Además, el hijo con rango 0, identificado como el de menor rango, se comunica internamente enviando mensajes a los otros hijos (hijo 1 y hijo 2).

Estos hijos adicionales reciben y muestran los mensajes enviados por el hijo de menor rango, demostrando una comunicación efectiva tanto entre el padre y los hijos como entre los propios hijos. El flujo de mensajes se completa sin bloqueos ni errores, indicando que la implementación de la gestión dinámica de procesos y la comunicación en MPI funciona conforme a los requisitos de la práctica.



# Cuestiones

## ¿Es posible que un hijo tenga varios padres?

**No**, en el contexto de MPI y la función `MPI_Comm_spawn`, cada proceso hijo es lanzado por un único proceso padre y establece comunicación con él a través de un intercomunicador específico. La estructura de MPI no admite que un hijo pertenezca simultáneamente a varios padres.

## ¿Podría ser lanzado un hijo por diferentes padres de forma alternativa?

Alternativamente, sí sería posible en una estructura más compleja donde diferentes procesos padres lanzan hijos independientes para tareas distintas, pero esto **requeriría una coordinación explícita de los padres**, y no se trataría del mismo "hijo" en sentido técnico, sino de procesos distintos que podrían comunicarse indirectamente.

## ¿Puede un hijo lanzar otros hijos suyos?

**Sí**, un proceso hijo en MPI puede actuar como un "padre" al lanzar nuevos procesos usando `MPI_Comm_spawn`, creando así una jerarquía de procesos donde los hijos pueden generar subprocesos para tareas específicas.

## Reflexión sobre las posibilidades de estas herramientas.

La gestión dinámica de procesos en MPI abre la puerta a sistemas más flexibles y escalables, donde los recursos del clúster pueden adaptarse a la carga de trabajo en tiempo real. Además, este modelo **facilita la distribución de tareas y mejora la tolerancia a fallos** en sistemas distribuidos, acercando MPI al paradigma de máquinas virtuales, aunque con ciertas limitaciones en la gestión de fallos y reconfiguración de nodos

# 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**.