PRÁCTICA 8: GESTIÓN DINÁMICA DE PROCESOS

Arquitecturas paralelas, Ingeniería informática, UBU.

Sergio Buil Laliena, Sandro Martín Rubio, Rocío Esteban Valverde

2 diciembre 2023

ÍNDICE

Objetivos

Ensayar las técnicas de configuración dinámica del cluster como aproximación al concepto de máquina virtual.

Introducción

El código proporcionado es un ejemplo de programación paralela utilizando la biblioteca MPI (Message Passing Interface). MPI es ampliamente utilizado en la programación paralela para crear aplicaciones que se ejecutan en sistemas distribuidos o en múltiples procesadores en un clúster. El objetivo de este código es crear un código “padre” que arranque un determinado número de procesos “hijo”. El “padre” se lanzará desde MPI y el código del “hijo” se lanzará desde el código del “padre”. Se mandarán mensajes de saludo para verificar la conexión y creación de los “hijos”. El “hijo” de menor rango además mandará un saludo a todos sus “hermanos”.

Código

Programa padre

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

#define N 3  // Numero de hijos

int main(int argc, char* argv[])
{
	//nuevos intercomunicadores
	MPI_Comm intercom, intracom;

	MPI_Init(&argc, &argv);

	int tamano;
	char saludo_padre[] = "Hola, soy tu padre";
	char ruta_hijo[] = "PracticaARPA8hijo.exe";
	int array_of_errcodes[4];

	printf("\nSoy el padre");
	fflush(stdout);

	MPI_Comm_spawn(ruta_hijo,   //nombre del programa hijo
			MPI_ARGV_NULL, //argumentos en linea de comandos
			3, // numero de copias que lanzar (del programa mpi)
			MPI_INFO_NULL,
			0,                              //root, rango del porceso pradre
			MPI_COMM_SELF,
			&intercom, //devolviendo un nuevo intercomunicador
			array_of_errcodes);
```

```
    printf("\nHola, soy el padre, he lanzado los hijos \n");
    fflush(stdout);

    MPI_Intercomm_merge(intercom, 0, &intracom);

            //for (int i = 0; i < tamano; i++) {
            //          MPI_Send(saludo_padre, 20, MPI_CHAR, i, 0, MPI_COMM_WORLD);
            //}
    MPI_Bcast(saludo_padre, 20, MPI_CHAR, 0, intracom);

    printf("\nPadre: saludo enviado\n");
    fflush(stdout);

    MPI_Comm_disconnect(&intracom);
    MPI_Finalize();
            return 0;
}
```

Programa hijo

```
#include <mpi.h>
#include <stdio.h>
#include <string.h>
int main(int argc, char* argv[])
{
            int tamano, mirango;
            char saludo_pa[20];
            char saludo_hijo[] = "Hola hermano, soy el hijo menor";
            char saludo_her[33];

            MPI_Comm commPadre, intracom;
            MPI_Init(&argc, &argv);

    MPI_Comm_get_parent(&commPadre);
            if (commPadre == MPI_COMM_NULL) {
       printf("No se pudo obtener el comunicador del padre.\n");
       fflush(stdout);
       fprintf(stderr, "Error: El proceso hijo no fue lanzado por un padre.\n");
       MPI_Abort(MPI_COMM_WORLD, 1);
            }
    MPI_Intercomm_merge(commPadre, 1, &intracom);
    MPI_Comm_rank(MPI_COMM_WORLD, &mirango);
    MPI_Comm_size(MPI_COMM_WORLD, &tamano);

    printf("Hola, soy el hijo %d y he sido creado por mi padre\n", mirango);
    fflush(stdout);

            //MPI_Recv(saludo_pa, 20, MPI_CHAR, 0, 0, intracom, MPI_STATUS_IGNORE);
    MPI_Bcast(&saludo_pa, 20, MPI_CHAR, 0, intracom);
```

```
    printf("Soy el hijo %d y he recibido el saludo: %s\n", mirango, saludo_pa);
    fflush(stdout);

    fflush(stdout);
            if (mirango == 0) // primer proceso hijo
            {
        //MPI_Bcast(&saludo_hijo, sizeof(saludo_hijo), MPI_CHAR, 0, intracom);
        for (int i = 1; i < tamano; i++) {  // Asegúrate de enviar a todos los hermanos en el nuevo comunicador
            MPI_Send(saludo_hijo, strlen(saludo_hijo) + 1, MPI_CHAR, i, 1, intracom);
        }
            }
            else
            {
        MPI_Recv(saludo_her, strlen(saludo_hijo)+1, MPI_CHAR, 0, 0, intracom, MPI_STATUS_IGNORE);
        printf("Soy el proceso %d y he recibido: %s\n", mirango, saludo_her);
        fflush(stdout);

            }
    MPI_Comm_disconnect(&intracom);
    MPI_Finalize();
            return 0;
}
```

Descripción del Código

En cuanto al código del padre:

1. Inicialización de MPI

Se incluyen las bibliotecas necesarias para programar con MPI. Se define el número de hijos que lanzará el “padre”. Se inicializa el MPI.

2. Declaración de comunicadores e inicialización de variables:

Se declaran dos comunicadores MPI: intercom para la comunicación entre “padres” e “hijos” e intracom para la fusión de comunicadores. Se inicializan las variables del saludo del “padre”, la ruta del programa “hijo” y un array de códigos de error.

3. Creación de procesos hijos:

Se lanza el spawn de los “hijos” usando la ruta del programa “hijo”. Además se muestra en pantalla un mensaje del “padre” anunciando cuántos “hijos” ha lanzado.

4. Fusión de comunicadores:

Se fusiona el comunicador intercom para formar un nuevo comunicador intracom. Además se manda el mensaje del padre a todos los hijos mediante MPI_Bcast.

5. Finalización de MPI

Se desconecta el comunicador intracom, se finaliza MPI al salir del bucle y se devuelve 0 para indicar una terminación exitosa del programa.

En cuanto al código del hijo:

1. Inicialización de MPI:

Se incluyen las bibliotecas necesarias y se inicializa el entorno MPI.

2.Declaración de variables:

Se declaran variables, incluyendo los saludos del hijo y del hermano.

3.Obtención y verificación del comunicador del padre:

Se obtiene mediante MPI_Comm_get_parent, y se verifica si no es nulo el comunicador del padre.

4.Fusión de comunicadores y obtención de rangos y tamaños:

Se fusiona el comunicador del padre para formar un nuevo comunicador intracom. Luego se obtienen tanto el rango como el tamaño del comunicador MPI_COMM_WORLD

5.Recepción del saludo del padre y mensaje de confirmación:

Se recibe el mensaje del padre mediante el comando MPI_Recv, luego se imprime un mensaje por pantalla de que el hijo en concreto ha recibido el mensaje del padre.

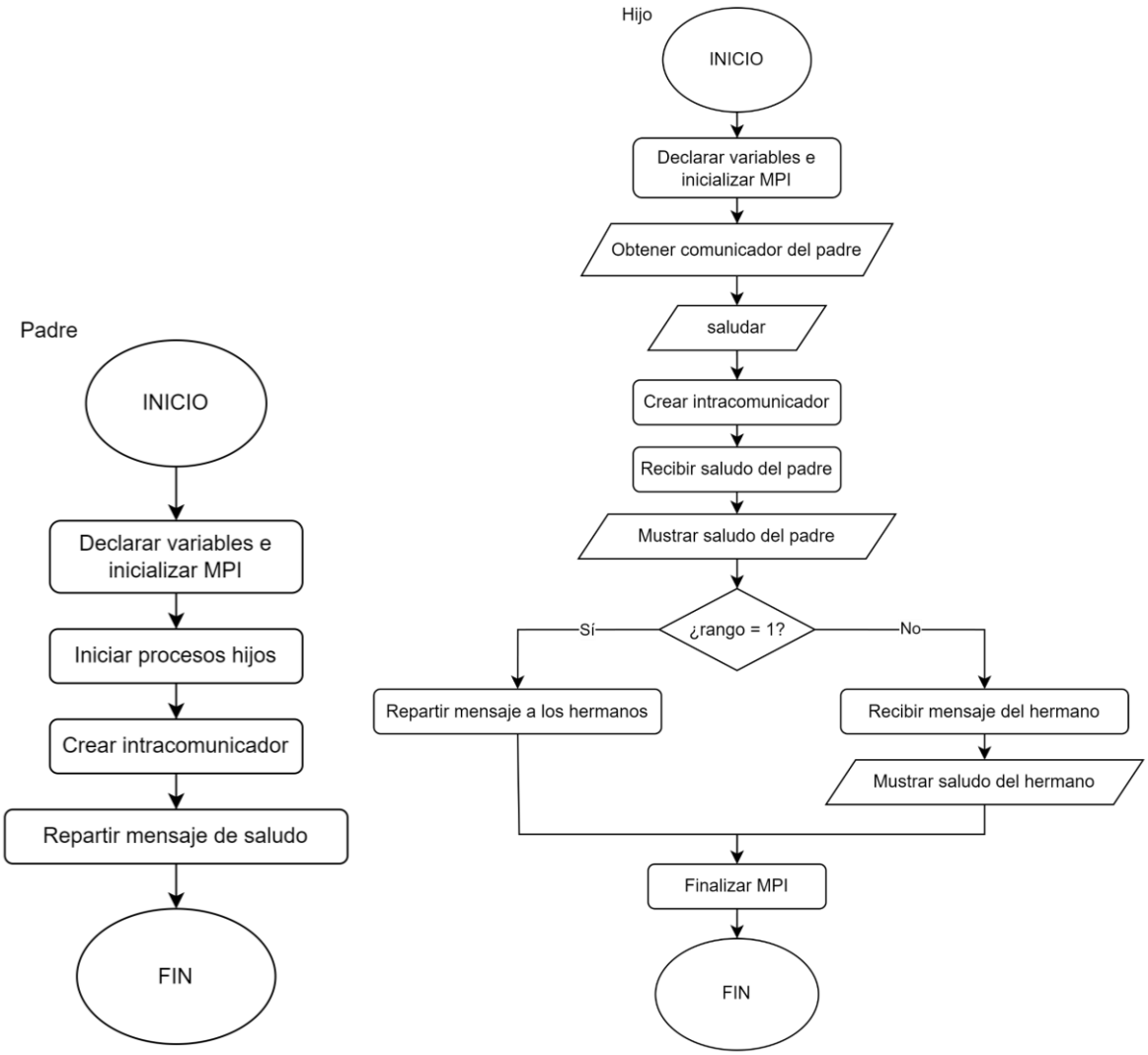
6. Condiciones para devolver el saludo:

Si el rango del proceso es 1 significa que es el de menor rango por tanto es el hijo menor y se lanza un broadcast del saludo del hijo menor a todos sus hermanos. Si no lo es se recibe el mensaje de un hermano a través del comunicador intracom y lo imprime por pantalla.

5. Finalización de MPI

Se desconecta el comunicador intracom, se finaliza MPI al salir del bucle y se devuelve 0 para indicar una terminación exitosa del programa.

Diagrama de flujo



Cuestiones

Según se ha visto que funciona la gestión dinámica de procesos un padre puede lanzar un número determinado de procesos hijos. ¿Sería posible que un hijo tuviera varios padres?

En el modelo estándar de MPI, cada proceso hijo es lanzado por un único proceso padre. La relación entre procesos en MPI es similar a un árbol de procesos donde hay un nodo raíz (el proceso padre inicial) y nodos hoja (procesos hijos). Esta estructura no admite que un proceso hijo tenga múltiples procesos padres en el sentido tradicional de gestión de procesos.

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

Sí, es posible lanzar un hijo por diferentes padres de forma alternativa en MPI aunque es algo complejo ya que cada proceso padre puede tener su propio espacio de direcciones y contexto de ejecución, lo que puede generar desafíos en la comunicación y coordinación entre los procesos padres y el proceso hijo. Además, se deben gestionar adecuadamente los recursos compartidos y garantizar la sincronización para evitar problemas de concurrencia.

¿Puede un hijo lanzar otros hijos suyos?

Sí, un proceso hijo puede lanzar otros procesos hijos. Esto permite la creación de estructuras jerárquicas y complejas de procesos dentro de un programa. Sin embargo, es crucial gestionar cuidadosamente la comunicación y sincronización entre estos procesos para evitar posibles problemas.

Tratar de realizar una reflexión sobre las posibilidades que se abren con estas herramientas

La capacidad de lanzar procesos dinámicamente en MPI abre una gama de posibilidades para la computación paralela. Se amplía enormemente la flexibilidad en la computación paralela, permitiendo a las aplicaciones adaptarse a las cargas de trabajo y recuperarse de fallos de manera más efectiva, facilitando un paralelismo más versátil y una mejor escalabilidad.

Conclusión

Estos códigos dan ejemplo de cómo los conceptos de comunicadores MPI y la reserva de memoria dinámica se aplican en el contexto de MPI, contribuyendo esto a una mejor comprensión de los fundamentos sobre programación paralela.