#include "mpi.h"

#include <iostream>

using namespace std;

int main(int argc, char \*argv[])

{

int rank, size, contador\_par, contador\_impar;

MPI\_Status estado;

MPI\_Init(&argc, &argv); // Inicializamos la comunicación de los procesos

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size); // Obtenemos el número total de procesos

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank); // Obtenemos el valor de nuestro identificador

if(rank == 0) //El proceso 0 se encarga de enviar su identificador al siguiente par

{

if(size >= 3)

{

MPI\_Send(&rank,1,MPI\_INT ,rank+2 ,0 ,MPI\_COMM\_WORLD);

//Validamos que la cantidad de procesadores introducidos en la corrida sea mayor o igual a tres,pues

} //de no ser así,solo existirían procesadores que envían,al poder ser solo el procesador 0,1 o ambos

else

{

cout << "El numero de procesadores no puede ser menor o igual que 2" << endl;

}

}

else if(rank == 1) //El proceso 1 se encarga de enviar su identificador al siguiente impar

{

if(size >= 4)

{

MPI\_Send(&rank,1,MPI\_INT ,rank+2 ,0 ,MPI\_COMM\_WORLD);

}

}

else{ //Verificamos que solo reciban los procesadores que no sean 0 o 1

if(rank % 2 == 0) //Comprobamos que los procesadores que reciben sean pares

{

MPI\_Recv(&contador\_par // Referencia al vector donde se almacenara lo recibido

,1 // tamaño del vector a recibir

,MPI\_INT // Tipo de dato que recibe

,rank-2 // pid del proceso origen de la que se recibe

,0 // etiqueta

,MPI\_COMM\_WORLD // Comunicador por el que se recibe

,&estado); // estructura informativa del estado

cout<<"Soy el proceso "<<rank<<" y he recibido "<<contador\_par<<endl;

contador\_par+=2; //Incrementamos dos para pasar el siguiente proceso par a recibir

if(rank <= size-3) //Limitamos el envío de los procesadores pares hasta el último par existente

{

MPI\_Send(&contador\_par, 1 ,MPI\_INT ,rank+2 , 0 ,MPI\_COMM\_WORLD); //Enviamos al siguiente procesador par del proceso actual su identificador

}

}

else //Comprobamos que los procesadores que reciben sean impares

{

MPI\_Recv(&contador\_impar // Referencia al vector donde se almacenara lo recibido

,1 // tamaño del vector a recibir

,MPI\_INT // Tipo de dato que recibe

,rank-2 // pid del proceso origen de la que se recibe

,0 // etiqueta

,MPI\_COMM\_WORLD // Comunicador por el que se recibe

,&estado); // estructura informativa del estado

cout<<"Soy el proceso "<<rank<<" y he recibido "<<contador\_impar<<endl;

contador\_impar+=2; //Incrementamos dos para pasar el siguiente proceso impar a recibir

if(rank <= size-3) //Limitamos el envio de los procesadores impares hasta el ultimo impar existente

{

MPI\_Send(&contador\_impar, 1 ,MPI\_INT ,rank+2 , 0 ,MPI\_COMM\_WORLD); //Enviamos al siguiente procesador impar del proceso actual su identificador

}

}

}

// Terminamos la ejecucion de los procesos, despues de esto solo existira

// el proceso 0

// ïOjo! Esto no significa que los demas procesos no ejecuten el resto

// de codigo despues de "Finalize", es conveniente asegurarnos con una

// condicion si vamos a ejecutar mas codigo (Por ejemplo, con "if(rank==0)".

MPI\_Finalize();

return 0;

}