

MPI: Punt a punt

- El programa hello-world.c implementa la funcionalitat vista en el 1r exemple de la classe d'avui (cada procés de l'aplicació imprimeix "Hello world!"). Modifiqueu aquest programa per tal de que cada procés indiqui el seu identificador i el nombre total de processos de l'aplicació en el seu missatge ("Hello world from X of Y!!")

1. Quina modificació heu hagut de fer?

S'ha afegit aquestes línies de codi per a obtenir el rang (identificador dl procés) i la mida (total de processos) de la aplicació:

```
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
```

```
MPI_Comm_size(MPI_COMM_WORLD, &size);
```

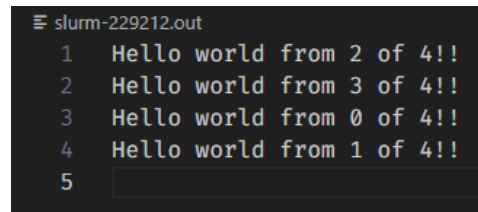
Per a poder imprimir-ho de la següent manera:

```
printf("Hello world from %d of %d!!\n", rank, size);
```

2. Podem extreure alguna conclusió sobre l'ordre d'execució dels processos a partir de l'ordre en que apareixen els missatges?

Observem que l'ordre en el que rebem els missatges de tornada són aleatoris

a priori, com observem en la imatge de sortida. Cal remarcar que l'ordre que veiem en la imatge no és el mateix que l'ordre en el qual els nodes reben la instrucció, i que la escriptura del missatge pot arribar en un ordre qualsevol, ja que intervenen molts factors en l'execució del programa, per cada procés.



```
slurm-229212.out
1 Hello world from 2 of 4!!
2 Hello world from 3 of 4!!
3 Hello world from 0 of 4!!
4 Hello world from 1 of 4!!
5
```

- Escriviu un programa que implementi la següent funcionalitat entre dos processos fent servir comunicació punt a punt bloquejant:

1. El procés 0 envia un missatge amb el nombre enter 12345 al procés 1 i el procés 1 imprimeix per pantalla el número rebut ("Soc el procés 1 i he rebut 12345")

(Programa inclòs al .zip)

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

int main(int argc, char *argv[]) {
    int rank;
    int number;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    if (rank == 0) { // Procés 0, envia
        number = 12345;
        MPI_Send(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
        printf("Soc el procés 0 i he enviat %d.\n", number);
    } else if (rank == 1) { // Procés 1, rep
        MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
        printf("Soc el procés 1 i he rebut %d.\n", number);
    }

    MPI_Finalize();
    return 0;
}
```

- Repetiu el mateix programa, però ara feu servir crides no bloquejants.

(Programa inclòs al .zip)

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

int main(int argc, char *argv[]) {
    int rank;
    int number;
    MPI_Status status;
    MPI_Request send_request, recv_request;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    if (rank == 0) { // Procés 0, envia
        number = 12345;
        MPI_Isend(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD, &send_request);
        printf("Soc el procés 0 i he enviat %d.\n", number);
    } else if (rank == 1) { // Procés 1, rep
        MPI_Irecv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &recv_request);
        printf("Soc el procés 1 i estic esperant per rebre missatge.\n");
        MPI_Wait(&recv_request, &status);
        printf("Soc el procés 1 i he rebut %d.\n", number);
    }

    MPI_Finalize();
    return 0;
}
```