DISTRIBUTED AND CLOUD COMPUTING

LAB1: MPI INTRODUCTION AND SETUP

GEORGE DIAMANTOPOULOS

OVERVIEW

- During the lab:
  - Introduction to technical concepts in Distributed and Cloud Computing
  - Application of concepts and Q&A
- Programming language: C
- Blackboard site:
  - Distributed and Cloud Computing Fall 2024 : **bb.sustech.edu.cn**

DISTRIBUTED COMPUTING INTUITION

Doing an assignment alone VS doing an assignment in a group?

Can you think of one good thing and one bad thing for each?

DISTRIBUTED COMPUTING INTUITION

Alone:

+ Management is simple

- You have to do everything

Group:

+ Work is split amongst many people (ideally!)

- You have to communicate and assign tasks

DISTRIBUTED COMPUTING INTUITION

Same things apply to computers!

MPI – MESSAGE PASSING INTERFACE

- Core requirement for all parallel systems: **Communication**
  - **Shared Memory**
  - **Message Passing**
- **MPI:** a message passing **standard** to allow for distributed/parallel computation
  - Many implementations [open-mpi, mpich, ...] for various languages [C, C++, FORTRAN, ...]
- This course: **open-MPI** for **C**
- MPI is **extremely common** for High Performance Computing (HPC) applications
  - If something runs on supercomputer (and does not use GPUs) there's a good chance it uses MPI

MPI – MESSAGE PASSING INTERFACE

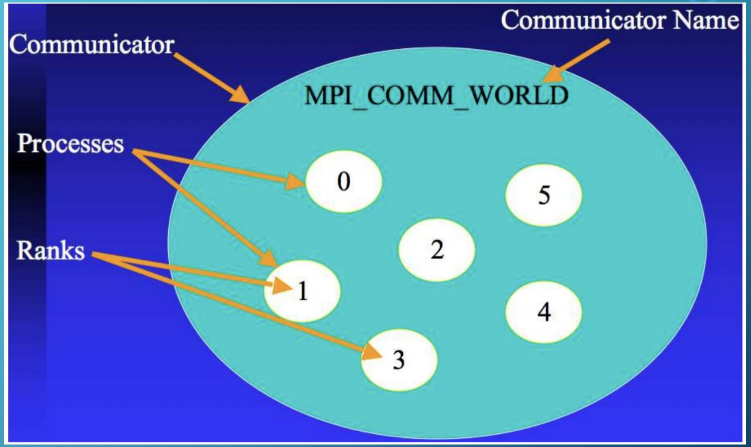
- MPI implements an interface for ***parallel process communication***
  - Abstracts the low-level details of processes communication
  - Allows the programmer to focus on the problem at hand (the parallel application)!
- You can think of **MPI** as **a postal service** and the processes as people sending letters
  - When you want to send a letter, you just drop it in the mailbox!

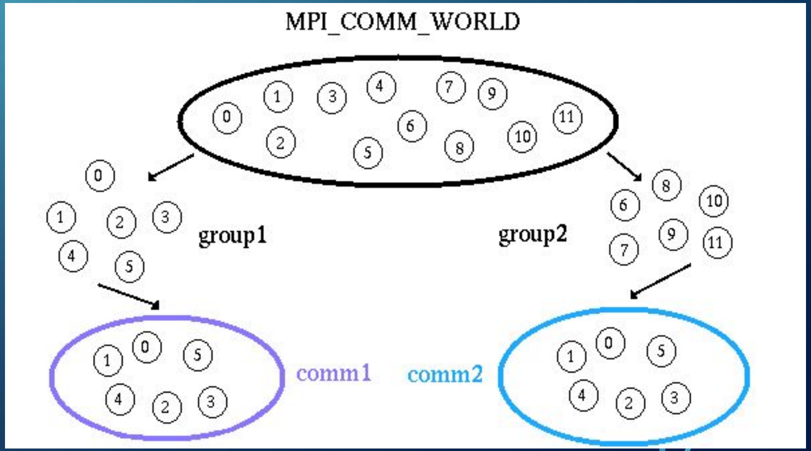
MPI - PROCESSES

- **MPI processes are managed by MPI and run concurrently (at the same time)**
  - On the CPU cores of a computer or server
  - On the cores of many CPUs each in a cluster of network computers
- **Each process:**
  - Runs an instance of the parallel application (e.g. large weather model)
  - Has its own local memory (does not share memory with any other process)
  - Can use the MPI interface to send messages to other processes

MPI - PROCESSES

- MPI **communicators - (neighbourhoods)**
  - Group processes
  - Assign them unique ID's called **ranks (house addresses)**
- "MPI_COMM_WORLD" is the default communicator
  - Contains all MPI processes generated for the application
- Costume communicators can be defined by the user
  - Can contain just a subset of the processes
  - Use full for managing the complex communication patterns





SETTING UP THE ENVIRONMENT FOR MPI

WINDOWS

- **Opt. 1: WSL2 (RECOMMENDED)**
  - Open the windows store and search 'Ubuntu' and install
  - Open ubuntu app and set up username and password when prompted
- **Opt. 2: Ubuntu Virtual Machine**
  - Download a VM hypervisor
    - VirtualBox is free and recommended
  - Download an ISO of the latest ubuntu LTS
  - There are plenty of guides online to help you

MAC OS

- **Opt. 1: Install locally (RECOMMENDED)**
  - Through homebrew or building from source
- **Opt. 2: Ubuntu Virtual Machine**
  - **not recommended**: macs natively support openMPI
  - Similar to Windows
  - Intel mac: VirtualBox
  - Apple silicone: VMware fusion

SETTING UP THE ENVIRONMENT FOR MPI

Ubuntu install (WLS or VM)

Run the following commands in the terminal:

```
sudo apt-get update
sudo apt install gcc
sudo apt-get install openmpi-bin openmpi-doc libopenmpi-dev
```

(`sudo apt install gcc`: C compiler: if not installed)

MAC OS Install

Get xcode-command line tools: `xcode-select --install`

Opt. 1: Homebrew (RECOMMENDED)

- Install homebrew packet manager: https://brew.sh/
- Run: `brew install open-mpi`

If download is slow or blocked set up the brew sustech mirror:
https://mirrors.sustech.edu.cn/help/homebrew.html#introduction

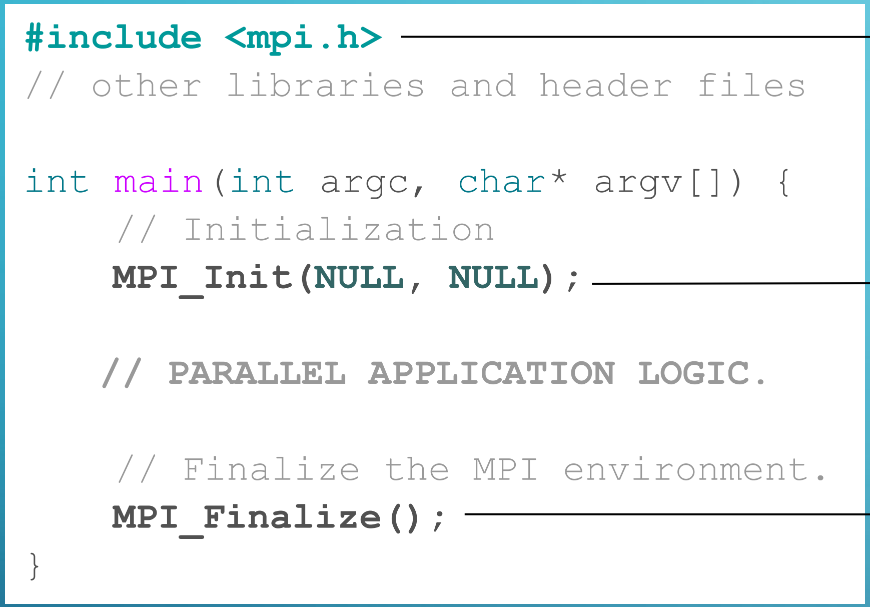
Opt. 2: Build openMPI (NOT RECOMMENDED)

- First answer at: https://www.open-mpi.org/faq/?category=building
- Download the zipped source code and unzip
- Go the unzipped directory and run the configuration
- Make all install

Code example:

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

int main(int argc, char* argv[]) {
    // Initialization
    MPI_Init(NULL, NULL);
    // Get the number of processes
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);
    // Get the rank of the process
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    // Get the name of the processor
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
    MPI_Get_processor_name(processor_name, &name_len);
    // Print off a hello world message
    printf("Hello world from processor %s, rank %d out of %d processors\n",
            processor_name, world_rank, world_size);
    // Finalize the MPI environment.
    MPI_Finalize();
}
```

MPI - BOILERPLATE CODE



```
#include <mpi.h>
// other libraries and header files

int main(int argc, char* argv[]) {
    // Initialization
    MPI_Init(NULL, NULL);

    // PARALLEL APPLICATION LOGIC.

    // Finalize the MPI environment.
    MPI_Finalize();
}
```

Imports the MPI library

Initialises the MPI environment

Finalise the MPI environment

THIS CODE NEVER CHANGES AND IS ALWAYS REQUIRED WHEN WORKING WITH MPI

MPI – USEFUL UTILITY FUNCTIONS

```c
// Get the number of processes
int world_size;
MPI_Comm_size(MPI_COMM_WORLD, &world_size);
```

Sets world_size equal equal to the total number of MPI processes

```c
// Get the rank of the process
int world_rank;
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
```

Sets world_rank equal equal to the rank of the current process

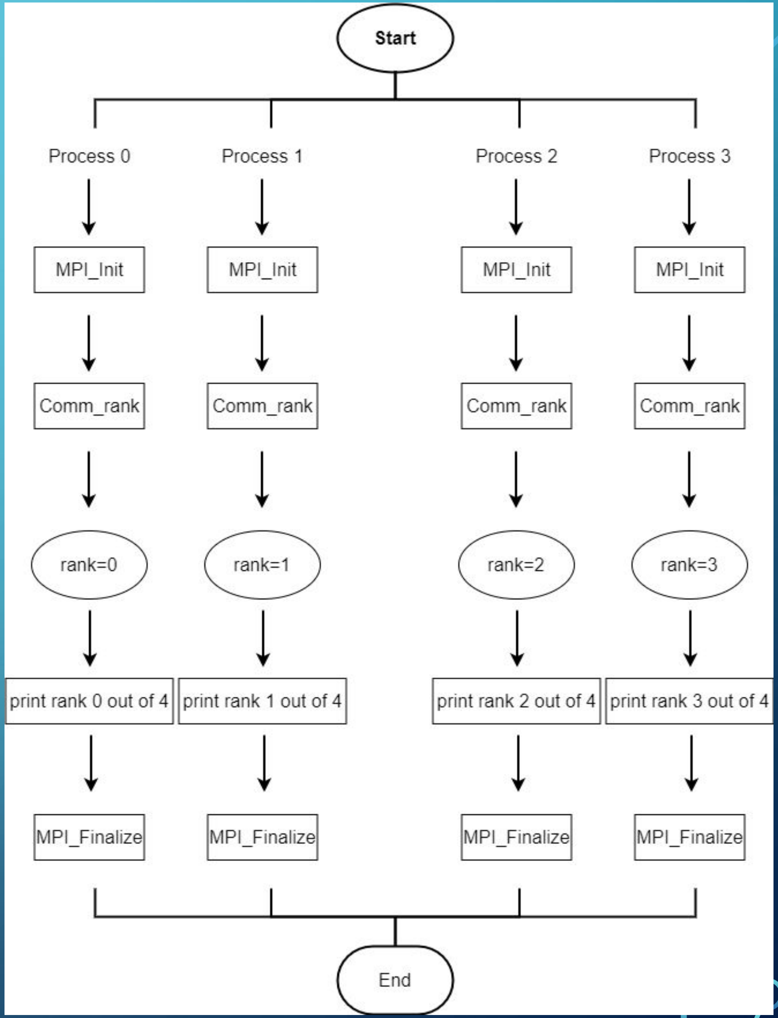
```c
// Get the name of the processor
char processor_name[MPI_MAX_PROCESSOR_NAME];
int name_len;
MPI_Get_processor_name(processor_name,
&name_len);
```

Retrieves the name of the processor (usually, the name you have given to the PC like GeorgesLaptop)

Code example:

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

int main(int argc, char* argv[]) {
    // Initialization
    MPI_Init(NULL, NULL);

    // Get the number of processes
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD,  &world_size);

    // Get the rank of the process
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD,  &world_rank);

    // Get the name of the processor
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
    MPI_Get_processor_name(processor_name,  &name_len);

    // Print off a hello world message
    printf("Hello world from processor %s, rank %d out of %d
         processors\n", processor_name, world_rank, world_size);

    // Finalize the MPI environment.
    MPI_Finalize();
}
```

Diagram of MPI execution



https://github.com/wesleykendall/mpitutorial/blob/gh-pages/tutorials/mpi-hello-world/code/mpi_hello_world.c

COMPILATION AND RUNNING

To compile, use mpicc, which is just a wrapper around gcc:

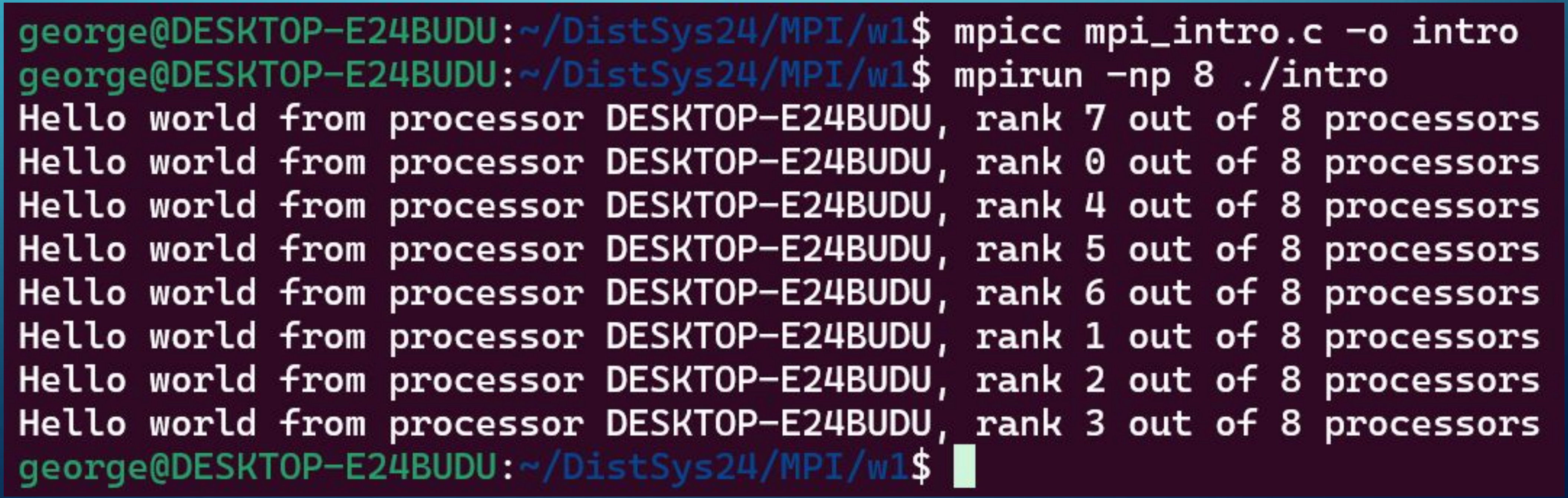
```
mpicc source.c -o executable_name
```

Then run the program use mpirun:

```
mpirun -np 4 ./executable_name
```

-np X: tells MPI how many processes to create

The result:

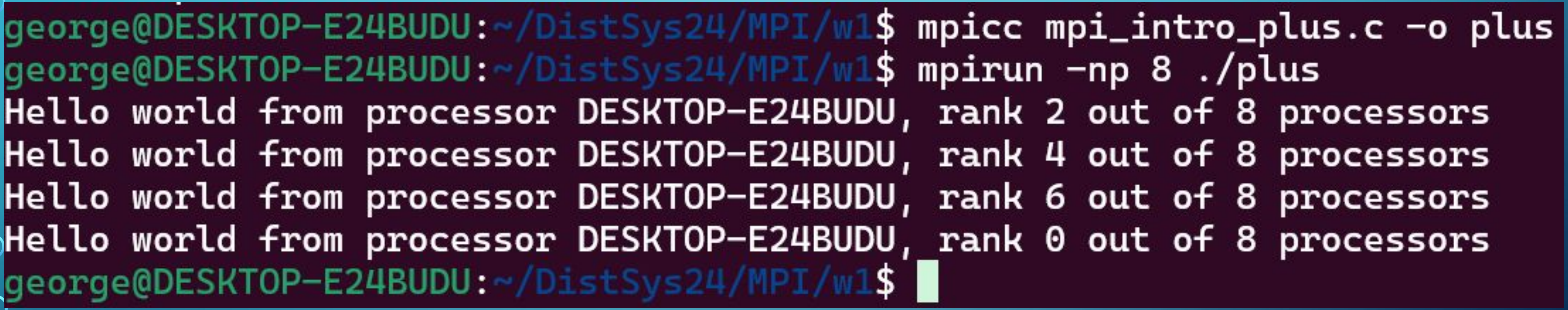


```
george@DESKTOP-E24BUDU:~/DistSys24/MPI/w1$ mpicc mpi_intro.c -o intro
george@DESKTOP-E24BUDU:~/DistSys24/MPI/w1$ mpirun -np 8 ./intro
Hello world from processor DESKTOP-E24BUDU, rank 7 out of 8 processors
Hello world from processor DESKTOP-E24BUDU, rank 0 out of 8 processors
Hello world from processor DESKTOP-E24BUDU, rank 4 out of 8 processors
Hello world from processor DESKTOP-E24BUDU, rank 5 out of 8 processors
Hello world from processor DESKTOP-E24BUDU, rank 6 out of 8 processors
Hello world from processor DESKTOP-E24BUDU, rank 1 out of 8 processors
Hello world from processor DESKTOP-E24BUDU, rank 2 out of 8 processors
Hello world from processor DESKTOP-E24BUDU, rank 3 out of 8 processors
george@DESKTOP-E24BUDU:~/DistSys24/MPI/w1$
```

MPI PROGRAMMING INTUITION

- Every process runs the same program! What's the point?
  - **Ranks!**
- Every process has a unique rank which we can use to assign it different tasks!

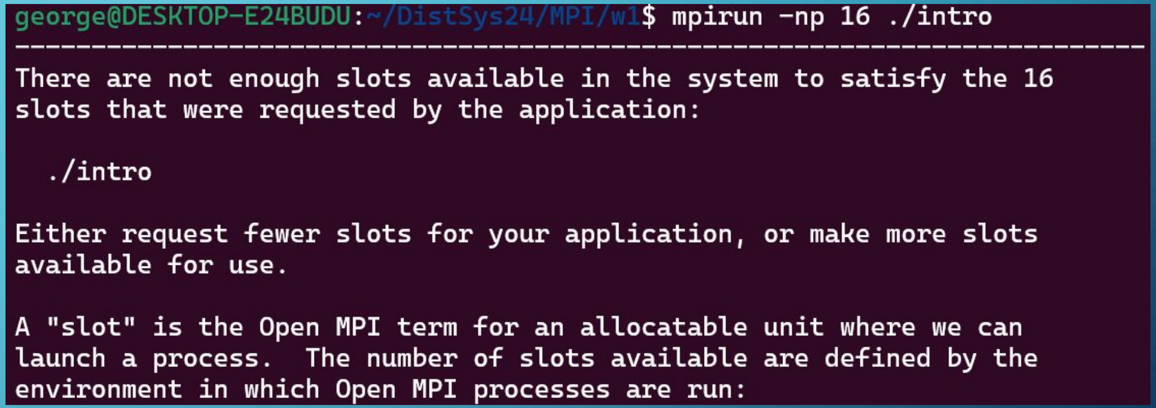
```c
#include <mpi.h>
#include <stdio.h>

int main(int argc, char* argv[]) {
    // Initialization
    MPI_Init(NULL, NULL);

    // Get the number of processes
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD,  &world_size);

    // Get the rank of the process
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD,  &world_rank);

    // Get the name of the processor
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
    MPI_Get_processor_name(processor_name,  &name_len);

    if(world_rank % 2 == 0){
     // Print a hello world message
     printf("Hello world from processor %s, rank %d out of %d
        processors\n", processor_name, world_rank, world_size);
    }

    // Finalize the MPI environment.
    MPI_Finalize();
}
```



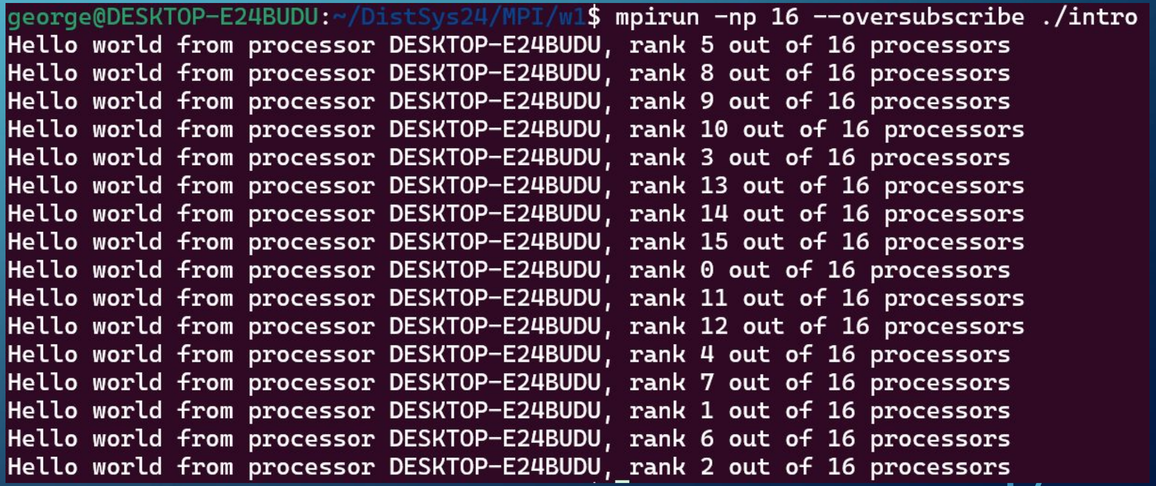
```
george@DESKTOP-E24BUDU:~/DistSys24/MPI/w1$ mpicc mpi_intro_plus.c -o plus
george@DESKTOP-E24BUDU:~/DistSys24/MPI/w1$ mpirun -np 8 ./plus
Hello world from processor DESKTOP-E24BUDU, rank 2 out of 8 processors
Hello world from processor DESKTOP-E24BUDU, rank 4 out of 8 processors
Hello world from processor DESKTOP-E24BUDU, rank 6 out of 8 processors
Hello world from processor DESKTOP-E24BUDU, rank 0 out of 8 processors
george@DESKTOP-E24BUDU:~/DistSys24/MPI/w1$ 
```

COMPILATION AND RUNNING (EXTRA)

- **Process to CPU core mapping**
  - MPI runs new processes in what are call **'slots'**
  - By default, MPI identifies the number of CPU cores and creates an equal number of **slots**
  - In this case, due to multithreading, there are 16 threads available for processing
  - Going over that amount is possible, but processes are not fully parallel (CPU task scheduling)



```
george@DESKTOP-E24BUDU:~/DistSys24/MPI/w1$ mpirun -np 16 ./intro
--------------------------------------------------------------------------
There are not enough slots available in the system to satisfy the 16
slots that were requested by the application:

  ./intro

Either request fewer slots for your application, or make more slots
available for use.

A "slot" is the Open MPI term for an allocatable unit where we can
launch a process.  The number of slots available are defined by the
environment in which Open MPI processes are run:
```



```
george@DESKTOP-E24BUDU:~/DistSys24/MPI/w1$ mpirun -np 16 --oversubscribe ./intro
Hello world from processor DESKTOP-E24BUDU, rank 5 out of 16 processors
Hello world from processor DESKTOP-E24BUDU, rank 8 out of 16 processors
Hello world from processor DESKTOP-E24BUDU, rank 9 out of 16 processors
Hello world from processor DESKTOP-E24BUDU, rank 10 out of 16 processors
Hello world from processor DESKTOP-E24BUDU, rank 3 out of 16 processors
Hello world from processor DESKTOP-E24BUDU, rank 13 out of 16 processors
Hello world from processor DESKTOP-E24BUDU, rank 14 out of 16 processors
Hello world from processor DESKTOP-E24BUDU, rank 15 out of 16 processors
Hello world from processor DESKTOP-E24BUDU, rank 0 out of 16 processors
Hello world from processor DESKTOP-E24BUDU, rank 11 out of 16 processors
Hello world from processor DESKTOP-E24BUDU, rank 12 out of 16 processors
Hello world from processor DESKTOP-E24BUDU, rank 4 out of 16 processors
Hello world from processor DESKTOP-E24BUDU, rank 7 out of 16 processors
Hello world from processor DESKTOP-E24BUDU, rank 1 out of 16 processors
Hello world from processor DESKTOP-E24BUDU, rank 6 out of 16 processors
Hello world from processor DESKTOP-E24BUDU, rank 2 out of 16 processors
```

CONFIGURATION OF MPIRUN

- **Official documentation available at:**
  - https://www.open-mpi.org/doc/v4.0/man1/mpirun.1.php

Structure of mpirun: `mpirun [MPI options] <program> [program arguments]`

KEY TAKEAWAY: Be careful to add MPI options **BEFORE** the program, otherwise, they will be treated as arguments for the program and will not take effect!