AMPLIACIÓN DE SISTEMAS OPERATIVOS

Práctica 2 – Programación paralela

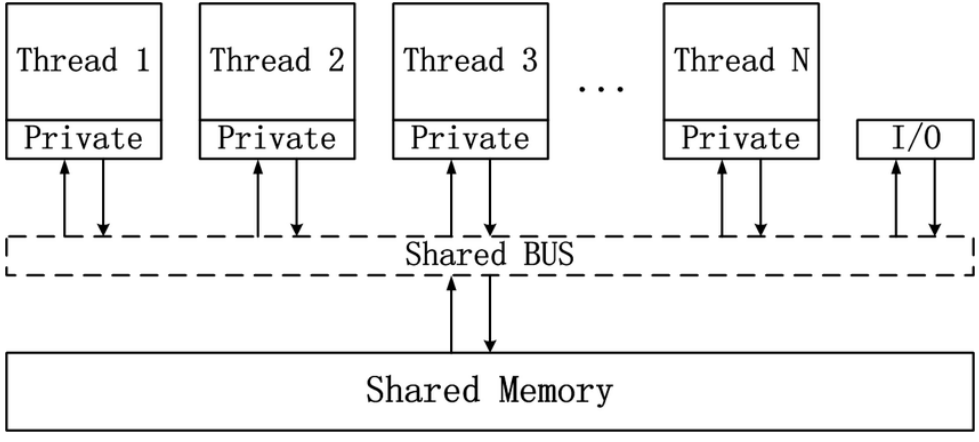
Prof. Luis Alberto Aranda

Universidad Rey Juan Carlos

COMUNICACIÓN ENTRE PROCESOS

- **Modelo de memoria compartida**: los procesos acceden a una memoria común en la que se comparten variables que varios procesos pueden leer/escribir

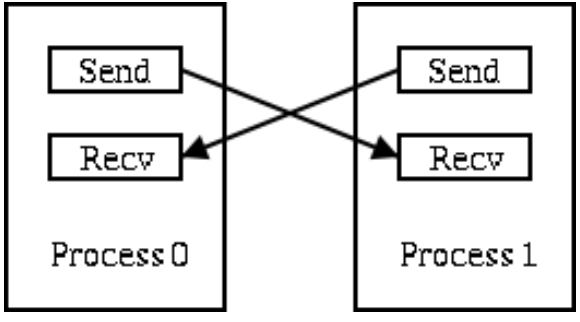
Ej. *POSIX Threads*



COMUNICACIÓN ENTRE PROCESOS

- **Modelo de paso de mensajes**: los procesos intercambian mensajes entre ellos. Un proceso envía y el otro recibe

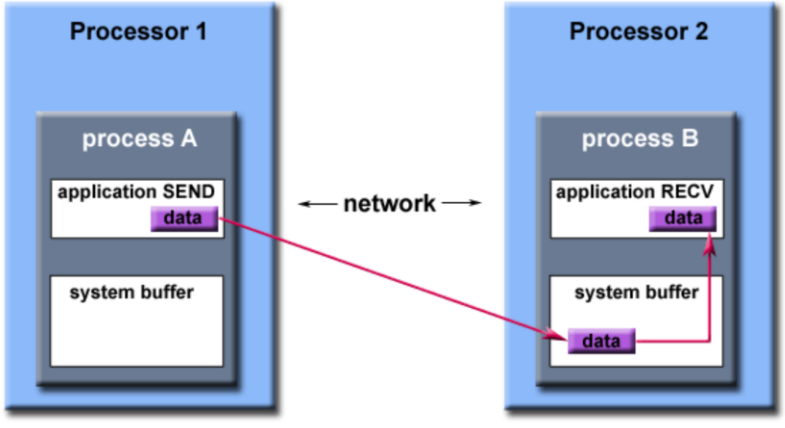
Ej. *MPI (Message Passing Interface)*



¡Cuidado con los interbloqueos!

MPI (MESSAGE PASSING INTERFACE)

- Modelo de comunicación de paso de mensajes mediante **primitivas de envío y recepción** de mensajes
- **Estándar** de facto en librerías de paso de mensajes
- Utilizado para comunicación de procesos a través de la red de comunicaciones de la arquitectura



Path of a message buffered at the receiving process

MPI: FUNCIONALIDADES

- Funciones para **gestión del entorno** MPI
- Funciones para gestión y **envío** de mensajes
- Funciones de **comunicación** punto a punto **bloqueantes**
- Funciones de **comunicación** punto a punto **no bloqueantes**
- Funciones de **comunicación colectivas**

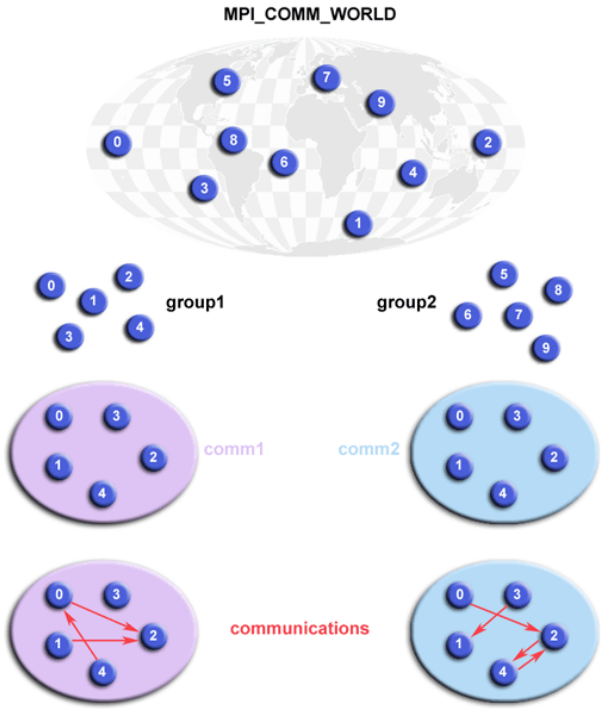
https://www.open-mpi.org/doc/v4.0/

MPI: GESTIÓN DEL ENTORNO

- **MPI_Init**: Inicializa el entorno MPI
- **MPI_Comm_size**: Devuelve el número de procesos que hay en el comunicador
- **MPI_Comm_rank**: Devuelve el rango (identificador numérico) del proceso dentro del comunicador
- **MPI_Finalize**: Finaliza el entorno MPI
- **MPI_Abort**: Termina todos los procesos asociados a un comunicador

MPI: GESTIÓN Y ENVÍO DE MENSAJES

- Cada proceso tiene un identificador único dentro del contexto de ejecución
- **MPI_COMM_WORLD** es el identificador del comunicador al que pertenecen todos los procesos de una ejecución
- El **comunicador** se divide en **grupos**, que son colecciones de procesos que se pueden comunicar entre ellos



MPI: GESTIÓN Y ENVÍO DE MENSAJES

- **Hay funciones bloqueantes:** la comunicación no continúa hasta que el envío o recepción se haya completado

  Ej. MPI_Send, MPI_Recv

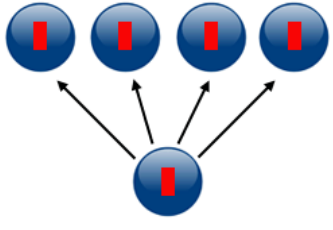
- **Hay funciones no bloqueantes:** permiten solapar cómputos con comunicaciones y no esperan a que se complete dicha comunicación

  Ej. MPI_Isend, MPI_Irecv

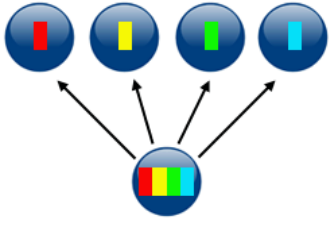
- La etiqueta **MPI_ANY_SOURCE** permite recibir mensajes de cualquier proceso
- La etiqueta **MPI_ANY_TAG** permite recibir cualquier tipo de mensaje

MPI: COMUNICACIÓN COLECTIVA

- **MPI_Barrier**: Bloquea un proceso hasta que todos los miembros del grupo lo hayan llamado
- **MPI_Bcast**: Envía un mensaje desde un proceso origen a todos los del grupo
- **MPI_Scatter**: Un proceso trocea un mensaje en partes iguales y lo envía al resto de procesos y a él mismo



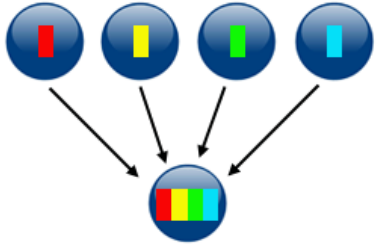
broadcast



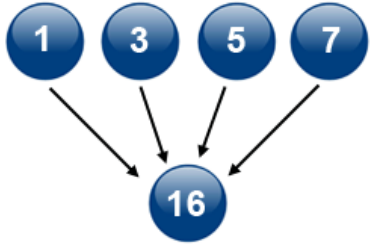
scatter

MPI: COMUNICACIÓN COLECTIVA

- **MPI_Gather**: Recoge datos de varios procesos en un único proceso raíz
- **MPI_Reduce**: Reduce todas las tareas de un grupo en un único proceso



gather



reduction

MPI: INSTALACIÓN

- Si se utiliza una máquina virtual es necesario **configurar el número de CPUs** ya que cada una correrá un proceso:

  En VirtualBox ir a: Configuración > Sistema > Procesador

- Es necesario tener instalado gcc/g++ y make

- Ejecutar el siguiente **comando rápido** en Ubuntu:

```
sudo apt-get install openmpi-bin openmpi-common openssh-client
openssh-server libopenmpi*
```

MPI: ESTRUCTURA DE UN PROGRAMA

Ficheros de **encabezado**:

- include “mpi.h”

Inicializar el entorno MPI:

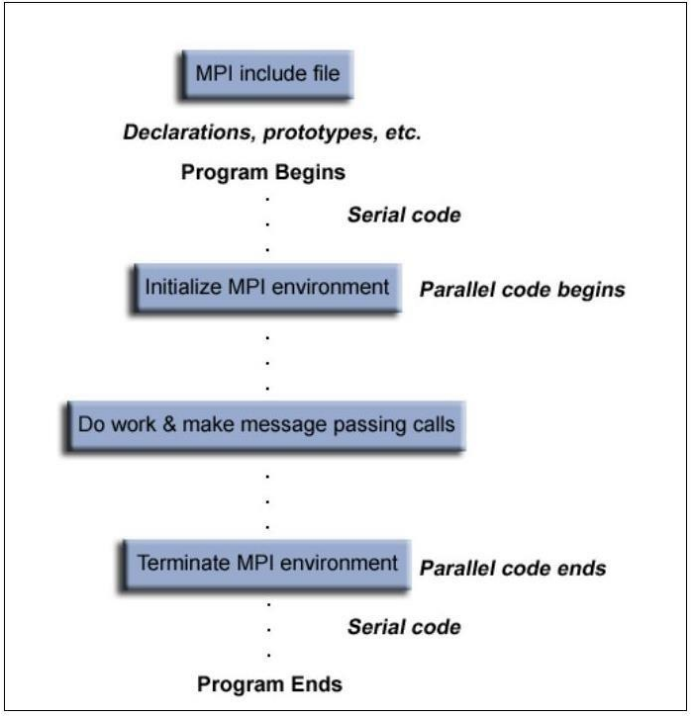
- MPI_Init

Parte del código con paso de mensajes

…

Cerrar el entorno MPI:

- MPI_Finalize



MPI: EJEMPLO 1 (en C)

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

int main(int argc, char **argv)
{
   int rank, nprocs;
   MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);

   printf("Hola mundo, soy el proceso %d de los %d que se están ejecutando\n", rank, nprocs);

   MPI_Finalize();
   return 0;
}
```

- **Compilar**:

```
mpicc holamundo.c –o holamundo
```

- **Ejecutar** utilizando 4 procesos:

```
mpirun –np 4 holamundo
```

MPI: EJEMPLO 2 (en C++)

```
#include "mpi.h"
#include <iostream>
using namespace std;

int main(int argc, char *argv[]) {
   int rank, size, contador;
   MPI_Status estado;
   MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD, &size);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);

   //Sólo el proceso 0
   if(rank == 0){
      MPI_Send(&rank //referencia al vector de elementos
            ,1 // tamaño del vector a enviar
            ,MPI_INT // Tipo de dato que envias
            ,rank+1 // pid del proceso destino
            ,0 //etiqueta
            ,MPI_COMM_WORLD); //Comunicador por el que se manda

   ...
```

```
...
   } else{
      MPI_Recv(&contador // Referencia al vector donde se almacenara lo recibido
            ,1 // tamaño del vector a recibir
            ,MPI_INT // Tipo de dato que recibe
            ,rank-1 // 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<<endl;
      contador++;
      //El último proceso no envía, sólo recibe
      if(rank != size-1)
         MPI_Send(&contador, 1 ,MPI_INT ,rank+1 , 0 ,MPI_COMM_WORLD);
   }
   MPI_Finalize();
   return 0;
}
```

- **Compilar**:

```
mpiCC ejemplo2.cpp –o ejemplo2
```

PRÁCTICA A REALIZAR: PARKING

- Desarrollar una aplicación concurrente de paso de mensajes con arquitectura maestro/esclavo

- **Simular la gestión de un parking**:
  - Automóviles (ocupan 1 plaza)
  - Camiones (ocupan 2 plazas)
  - Plazas contiguas
  - Una entrada y una salida
  - Se asigna plaza según la disponibilidad
  - Gestión infinita
  - Entrada: vehículo espera tiempo aleatorio para entrar. Sistema de control escoge la plaza
  - Salida: vehículo permanece tiempo aleatorio en el parking y notifica la plaza liberada

PRÁCTICA A REALIZAR: EJEMPLO

ENTRADA: Coche 1 aparca en 0. Plazas libres: 5
Parking: [1] [0] [0] [0] [0] [0]
ENTRADA: Coche 2 aparca en 1. Plazas libres: 4
Parking: [1] [2] [0] [0] [0] [0]
ENTRADA: Coche 3 aparca en 2. Plazas libres: 3
Parking: [1] [2] [3] [0] [0] [0]
ENTRADA: Coche 4 aparca en 3. Plazas libres: 2
Parking: [1] [2] [3] [4] [0] [0]
ENTRADA: Coche 5 aparca en 4. Plazas libres: 1
Parking: [1] [2] [3] [4] [5] [0]
SALIDA: Coche 2 saliendo. Plazas libres: 2
Parking: [1] [0] [3] [4] [5] [0]
SALIDA: Coche 3 saliendo. Plazas libres: 3
Parking: [1] [0] [0] [4] [5] [0]
ENTRADA: Coche 6 aparca en 2. Plazas libres: 2
Parking: [1] [0] [6] [4] [5] [0]

SALIDA: Coche 1 saliendo. Plazas libres: 3
Parking: [0] [0] [6] [4] [5] [0]
ENTRADA: Camión 101 aparca en 1. Plazas libres: 1
Parking: [101] [101] [6] [4] [5] [0]
SALIDA: Coche 4 saliendo. Plazas libres: 2
…

TAREAS A REALIZAR

1. **(4 puntos)** Gestión del parking para automóviles y una planta
2. **(5 puntos)** Gestión del parking también para camiones
3. **(1 punto)** Gestión del parking con varias plantas

IMPORTANTE gestionar correctamente los interbloqueos y casos de inanición

Realizar un documento PDF explicando los códigos y los resultados obtenidos

Entregar el PDF y los códigos por separado para su posterior evaluación