



# Ampliación de Sistemas Operativos

---

Práctica 2: Parking



# Índice

## Introducción

- Técnicas de comunicación entre procesos

## MPI Message Passing Interface

- Funciones de gestión de entorno
- Funciones de comunicación
- Instalación MPI en Ubuntu
- Ejemplos

## Práctica: Parking

## Evaluación



# Introducción

# Introducción

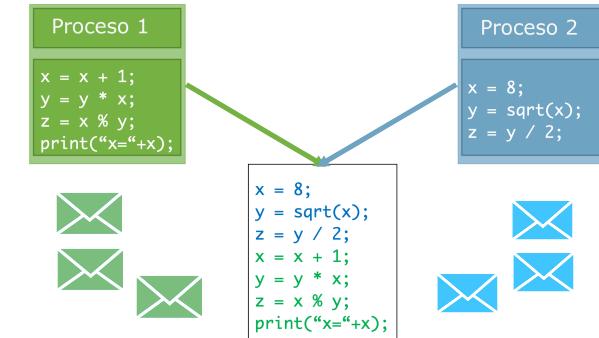
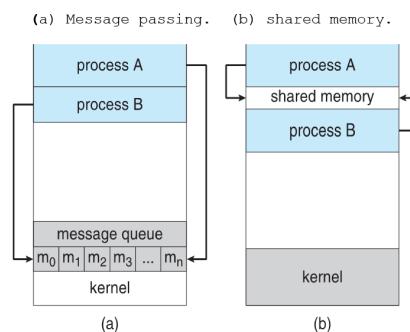
## 1. Técnicas comunicación entre procesos

### Modelo de Memoria compartida

- Los procesos pueden acceder a una memoria común
- Existen variables compartidas que varios procesos pueden leer y escribir

### Modelo de Paso de mensajes

- Los procesos se intercambian mensajes entre sí
- Un proceso envía mensaje y otro proceso lo recibe



# Introducción

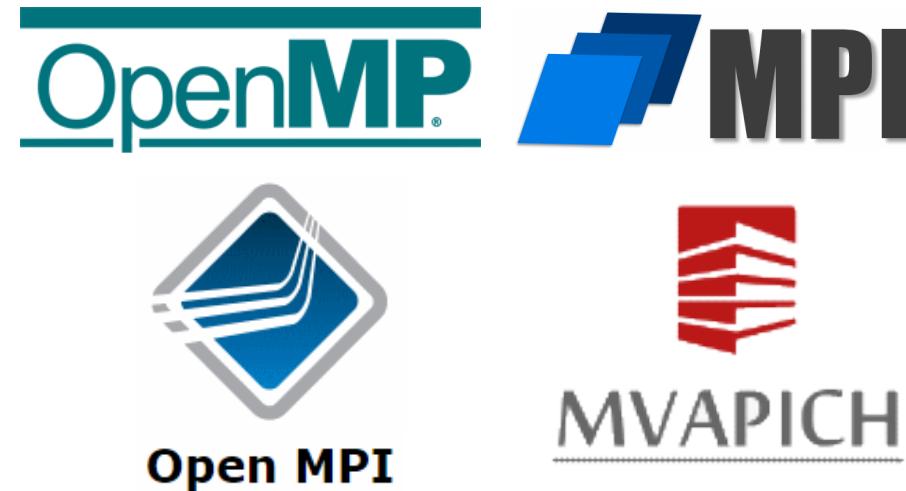
## 1. Técnicas de comunicación entre procesos

### APIs Memoria compartida

- Pthreads: POSIX Threads
- OpenMP:

### APIs Paso de mensajes

- MPI: Message Passing Interface
- PVM: Parallel Virtual Machine





# MPI

# Message Passing Interface

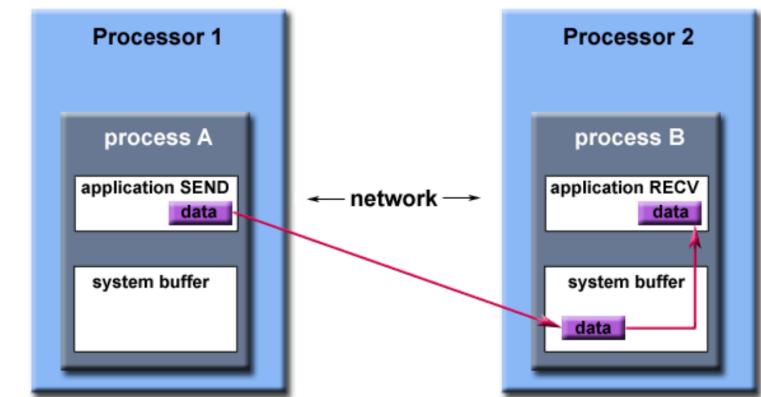
# MPI (Message Passing Interface)

- Qué es MPI?

- Modelo de comunicación de paso de mensajes mediante primitivas de envío y recepción de mensajes (Send y Receive)
- Estándar de facto en librerías de paso de mensajes
  - Especificación (para C, C++ y Fortran)
  - Hay muchas implementaciones diferentes.

- Para qué se utiliza?

- Comunicación de procesos a través de la red de comunicaciones incluida en la arquitectura

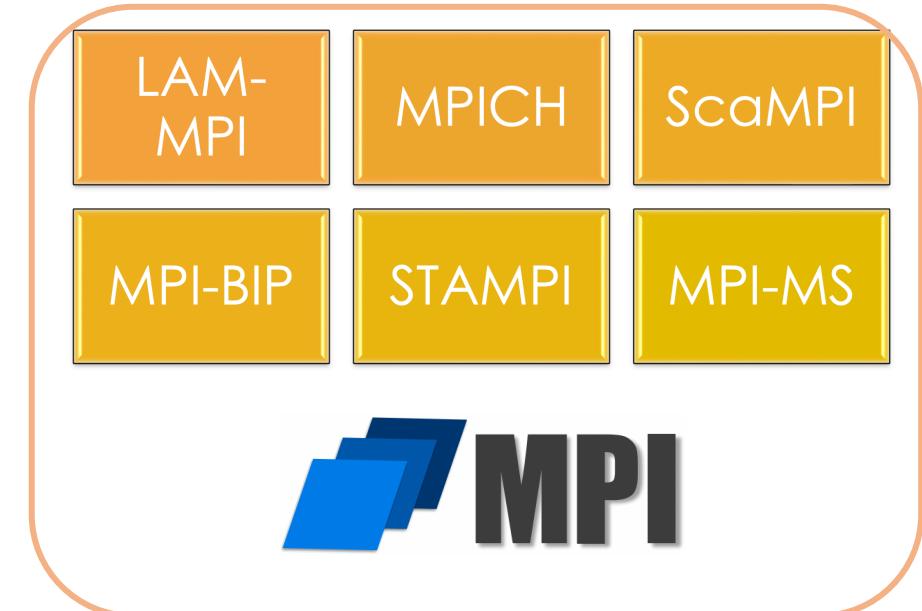


Path of a message buffered at the receiving process

# MPI (Message Passing Interface)

Estas librerías proporcionan distintos tipos de primitivas:

- Funciones para gestión del entorno MPI
- Funciones de comunicación punto a punto bloqueantes
- Funciones de comunicación punto a punto no bloqueantes
- Funciones de comunicación colectivas
- Funciones para gestión y definición de tipos de datos



# MPI (Message Passing Interface)

## Diferencia entre funciones bloqueantes y no bloqueantes:

- Las funciones bloqueantes no continúan con la comunicación hasta que el envío o recepción se haya completado.
- Las funciones no bloqueantes permiten solapar cómputo *con comunicaciones* y no esperan a que la primitiva de comunicaciones (función *Send* o *Receive*) se complete.

# MPI (Message Passing Interface)

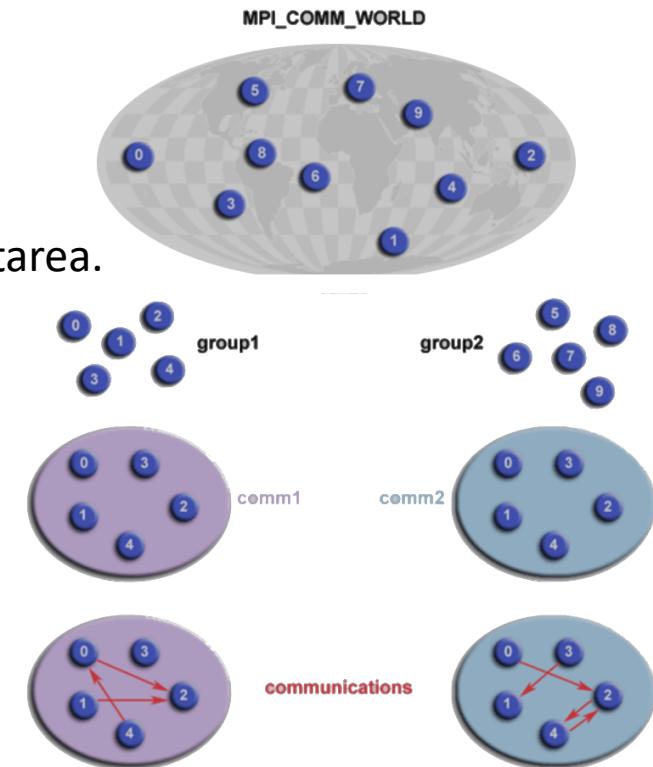
## Funciones para la gestión del entorno MPI.

- `MPI_Init`: Inicializa el entorno MPI.
- `MPI_Comm_size`: Devuelve el número de tareas MPI (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 (Message Passing Interface)

## Gestión tareas y comunicadores

- Cada tarea tiene un identificador único dentro del contexto de ejecución.
- MPI\_ANY\_SOURCE: parámetro comodín para recibir mensajes de cualquier tarea.
- Concepto de comunicador y grupos para definir que colecciones de procesos se pueden comunicar entre ellos.
- Todos los procesos MPI ejecutándose en el contexto son el comunicador MPI\_COMM\_WORLD.
- TAGs: entero no positivo para identificar únicamente los mensajes.
- MPI\_ANY\_TAG: parámetro comodín para recibir cualquier mensaje.



# MPI (Message Passing Interface)

## Funciones de comunicación punto a punto bloqueantes.

- MPI\_Send(buffer,count,type,dest,tag,comm)
- MPI\_Recv(buffer,count,type,source,tag,comm,status)

## Funciones de comunicación punto a punto no bloqueantes

- MPI\_Isend(buffer,count,type,dest,tag,comm,request)
- MPI\_Irecv(buffer,count,type,source,tag,comm,request)

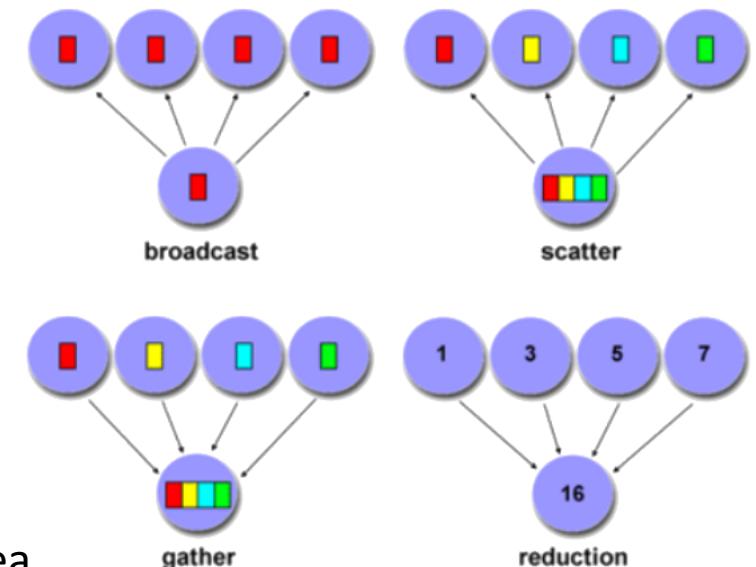
## Combinación habitual con funciones de tipo

- MPI\_Iprobe (comprobación no bloqueante del buffer)
- MPI\_Iprobe(source,tag,comm,flag,status)

# MPI (Message Passing Interface)

## Funciones de comunicación colectivas.

- MPI\_Barrier(comm) → Operación de sincronización
- MPI\_Bcast(&buffer,count,datatype,root,comm): proceso raíz → todos los procesos del grupo.
- MPI\_Scatter(&sendbuf,sendcnt,sendtype,&recvbuf, recvnt, recvtype,root,comm): mensajes únicos de origen a cada tarea del grupo
- MPI\_Gather(&sendbuf,sendcnt,sendtype,&recvbuf, recvcount,recvtype,root,comm): recopilar mensajes distintos de cada tarea de grupo para una única tarea destino.
- MPI\_reduce (&sendbuf,&recvbuf,count,datatype,op,root,comm): reducción a todas las tareas del grupo y coloca el resultado en una tarea



# Instalación MPI en Ubuntu

## Herramienta apt-get

- Práctica
  - sudo update
  - sudo apt-get install openmpi-bin: Ejecutor del programa en paralelo(mpirun).
  - sudo apt-get install xterm: debugger
- Cluster real/virtualizado
  - sudo apt-get install openmpi-bin openmpi-common openssh-client openssh-server libopenmpi1.3 libopenmpi-dbg libopenmpi-dev
    - **openmpi-bin**: Ejecutor del programa paralelo (mpirun).
    - **openssh-client, openssh-server**: Arquitectura segura de comunicación entre procesos.
    - **libopenmpi-dbg**: Generador de información de debug.
    - **libopenmpi-dev**: Herramientas necesarias para desarrollar el programas basados en MPI (mpicc).

# Instalación MPI en Ubuntu

## Paquetes

- Ir a la web: <http://www.open-mpi.org/software/ompi>
- Paso 1. Descomprimir el fichero: `tar -xvf openmpi-*`
- Paso 2. Ir dentro de la carpeta: `cd openmpi-*`
- Paso 3. Configurar el fichero de instalación: `./configure --prefix="/home/$USER/.openmpi"`
- Paso 4. Instalar:  
`make`  
`sudo make install`
- Paso 5. Incluir los paths lib y bin en "/home/<user>/.bashrc"  
`export PATH="$PATH:/home/$USER/.openmpi/bin"`  
`export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:/home/$USER/.openmpi/lib/"`

[https://lsi.ugr.es/jmantas/ppr/ayuda/datos/installaciones/Instalacion\\_OpenMPI.pdf](https://lsi.ugr.es/jmantas/ppr/ayuda/datos/installaciones/Instalacion_OpenMPI.pdf)

# Instalación MPI en Ubuntu

Resultado:

```
user@user:mpirun -version  
mpirun (Open MPI) 4.0.3
```

```
Report bugs to http://www.open-mpi.org/community/help/
```

# Ejemplos

## Hello World

```
#include <stdio.h>
#include "mpi.h"
int main(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 procesos que se están ejecutando\n", rank, nprocs); MPI_Finalize();
    return 0;
}
```

# Ejemplos

## Configurar entorno

- Práctica
  - Compilar: `mpicc HelloWorldMPI.c -o HelloWorldMPI.o`
  - Ejecutar:
    - Hostfile: número de slots (procesos) para ejecutar el programa
    - `localhost slots=4`
    - Ejecutar: `mpirun --hostfile filename -np 4 HelloWorldMPI.o`
    - Debugger: gdb en cada proceso
      - `mpirun -hostfile filename -np 4 xterm -e gdb ./HelloWorldMPI.o`

<https://www.open-mpi.org/faq/?category=debugging>

<https://www.open-mpi.org/faq/?category=running>

# Ejemplos

## Configurar entorno

- Cluster real/virtualizado

- Compilar: mpicc HelloWorldMPI.c -o HelloWorldMPI.o
  - Ejecutar:

- Hostfile: número de slots (procesos) para ejecutar el programa

- node 1

- node 2

- node 3...

- Ejecutar: mpirun --hostfile filename -np 4 HelloWorldMPI.o

- Debugger: gdb en cada proceso

- mpirun -hostfile filename -np 4 xterm -e gdb  
./HelloWorldMPI.o

<https://www.open-mpi.org/faq/?category=debugging>

<https://www.open-mpi.org/faq/?category=running>

# Ejemplos

## Message

```
#include <stdio.h>
#include <mpi.h>
int main(int argc, char* argv[]) {
    int me, nprocs, left, right, count,
        sigue; MPI_Status status;
    float inmsg;
    float outmsg;
    float sum;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &me);
    outmsg = me;
```

```
sigue=1;
right = (me+1)%nprocs;
left = (me-1+nprocs)%nprocs;
while (sigue) {
    MPI_Sendrecv(&outmsg, 1, MPI_FLOAT, left,
                0, &inmsg, 1, MPI_FLOAT, right, 0,
                MPI_COMM_WORLD, &status);
    sum += inmsg;
    outmsg = inmsg;
    if (inmsg == me)
        sigue=0;
}
printf("Soy el proceso %d y he sumado %1.2f \n", me, sum);
MPI_Finalize();
}
```

# Ejemplos

## Configurar entorno

- Práctica
  - Compilar: `mpicc MessageMPI.c -o MessageMPI.o`
  - Ejecutar:
    - Hostfile: número de slots (procesos) para ejecutar el programa  
`localhost slots=2`
    - Ejecutar: `mpirun --hostfile filename -np 2 MessageMPI.o`
    - Debugger: gdb en cada proceso  
`mpirun -hostfile filename -np 2 xterm -e gdb ./MessageMPI.o`

<https://www.open-mpi.org/faq/?category=debugging>

<https://www.open-mpi.org/faq/?category=running>

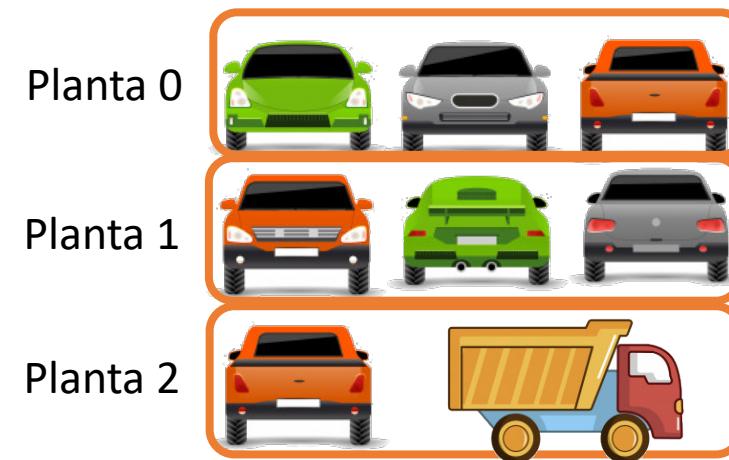


# Práctica 2: Parking

# Práctica 2: Parking

Objetivo general:

- Desarrollo de una aplicación concurrente
- Modelo paso de mensajes
- Arquitectura master/slave
- Slave: procesos que se encargan de hacer el trabajo.
- Master: encargado de gestionar los nodos slave.
- Compilar: mpicc:mpicc fuente.c -o ejecutable
- Fichero hostfile: configuración
  - modo cluster: node1  
                  node2  
                  node3
  - modo proceso: localhost slots=4



# Práctica 2: Parking

## Descripción concreta

- Objetivo principal: desarrollar un parking para vehículos y camiones
- Características:
  - Vehículos: ocupa 1 plaza
  - Camiones: ocupa 2 plazas
  - Plantas de garaje: plazas contiguas
  - Única salida: dispositivo de control
  - Dispositivo de control: asigna plaza
  - Tiempos de espera dentro del parking son aleatorios
  - Salida: vehículo/camión notifica dispositivo de control numero de plaza asignada  
libera plaza/plazas de aparcamiento
  - Entrada: vehículo/camión espera tiempo aleatorio para volver a entrar
  - Gestión indefinida

Planta 0:

0	1	2	3	...	N
---	---	---	---	-----	---

Planta 1:

0	1	2	3	...	N
---	---	---	---	-----	---

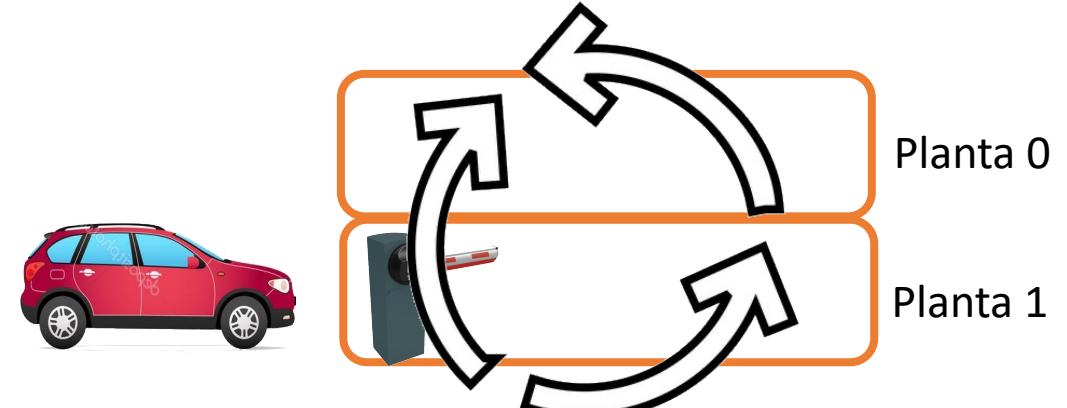
# Práctica 2: Parking

Programas a implementar

- Gestor del parking
  - Número de plazas (PLAZAS)
  - Número de plantas (PLANTAS)
- Vehículo
- Camión

Ejecución del algoritmo:

```
mpirun --hostfile hostfile -np 1 parking PLAZAS PLANTAS : -np  
10 coche : -np 3 camion
```



ENTRADA: Coche 1 aparcra en 0. Plazas libre: 5  
Parking: [1] [0] [0] [0] [0] [0]  
ENTRADA: Coche 2 aparcra en 1. Plazas libre: 4  
Parking: [1] [2] [0] [0] [0] [0]  
ENTRADA: Coche 3 aparcra en 2. Plazas libre: 3  
Parking: [1] [2] [3] [0] [0] [0]  
ENTRADA: Coche 4 aparcra en 3. Plazas libre: 2  
Parking: [1] [2] [3] [4] [0] [0]



# Evaluación

# Evaluación

- Práctica 2 (15%). Nota inferior a 5 puntos no hace media
- Grupos de tres personas máximo
- Evaluación en examen final

# Referencias

- Tutorial:

<https://computing.llnl.gov/tutorials/mpi/>

- Instalación:

[https://lsi.ugr.es/jmantas/ppr/ayuda/datos/installaciones/Instalacion\\_OpenMPI.pdf](https://lsi.ugr.es/jmantas/ppr/ayuda/datos/installaciones/Instalacion_OpenMPI.pdf)

# Preguntas



Miguel Ángel Rodríguez García  
[miguel.rodriguez@urjc.es](mailto:miguel.rodriguez@urjc.es)