**1. ¿Qué es un proceso de una computadora?**

Nos referimos al conjunto de instrucciones que ejecutará el [microprocesador](https://www.ecured.cu/Microprocesador) mientras lee un programa determinado. Esto también implica a la memoria reservada y a sus contenidos, el estado de ejecución en determinado momento, y la información que permite al sistema operativo planificar.

**2. Comunicación punto a punto entre dos procesos**

El paso de mensajes entre dos procesos se basa en dos operaciones ENVIAR Y RECIBIR. En caso del uso del lenguaje de programación C/C++, cuando un proceso requiere enviar un mensaje a otro proceso, se usa la función MPI\_Send(…), y cuando se desea recibir se usa MPI\_Receive(…)

#include<iostream>

#include”mpi.h”

using namespace std;

int main(int arg, char \*args[]){

int rank,size;

MPI\_Init(&arg,&args);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

int number;

if (rank == 0) {

number = -1;

MPI\_Send(&number, 1, MPI\_INT, 1, 0, MPI\_COMM\_WORLD);

} else if (rank == 1) {

MPI\_Receive(&number, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

printf("Proceso 1 received number %d from process 0\n",

number);

}

MPI\_Finalize();

}

3. ¿Qué es una memoria Ram (principal), Cache y Virtual? E indicar ¿Cómo funciona?

Memoria Ram: Random Access Memory o memoria de acceso aleatorio, es utilizada para procesar toda la información del sistema, en donde se ejecuta la mayor parte del software: el Sistema Operativo, el software de aplicación y otros programas similares.

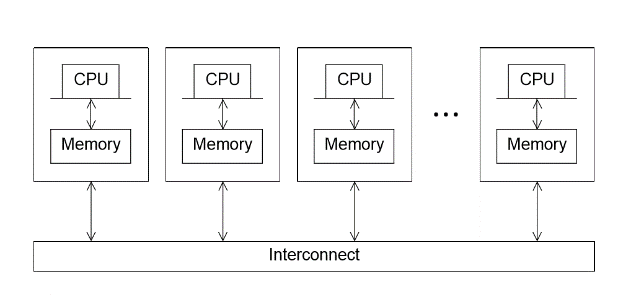
Memoria Cache: (Memoria Auxiliar) Memoria de Acceso rápido a los recursos de CPU, permite el almacenamiento momentáneo de los datos recién procesados en el bufer, es más rápida que la RAM pero más pequeña que la misma. Beneficia al microprocesador.

Memoria Virtual: Simulación de la Memoria RAM que posee una máquina pero en mayor tamaño, es la gestión de la memoria que se encarga que el sistema operativo realiza para acceder a una mayor cantidad de memoria de la físicamente disponible

4. ¿En qué consiste la **programación en Memoria Distribuida** y la **programación en Memoria Compartida?**

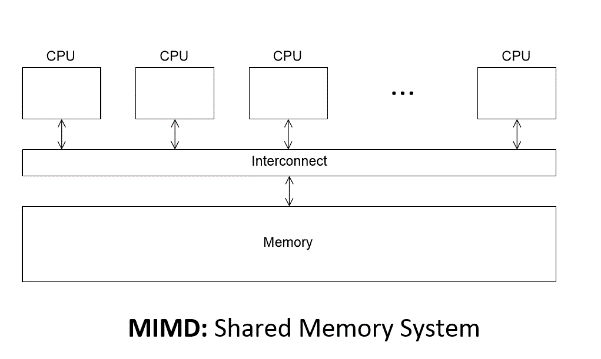
**Memoria Distribuida**

Cada procesador tiene su memoria Local, permitiendo su acceso de datos más rápido, pero pueden transferir o intercambiar datos con un intercomunicador. Representan la creación hibrida de dos tipos de computación paralelos. Es una abstracción utilizada para compartir datos entre computadores que no comparten memoria física



**Memoria Compartida**

Cuando acceden a un mismo espacio de direcciones. Todos los procesadores están conectados a un intercomunicador para acceder a la memoria permitiendo así el intercambio de datos entre dos programas ejecutándose al mismo tiempo.



5.Describa en 3 líneas como máximo e indicar los parámetros de los siguientes comandos del MPI:

MPI\_Send(

void\* data,//dirección del arreglo a enviar

int count, //tamaño del arreglo a enviar

MPI\_Datatype datatype, //tipo de dato a enviar

int destination, //pid del proceso que recibira

int tag, //etiqueta del mensaje

MPI\_Comm communicator) //comunicador

MPI\_Recv(

void\* data,//dirección del arreglo a recibir

int count, //tamaño del arreglo a recibir

MPI\_Datatype datatype, //tipo de dato

int source, //pid del proceso que envia

int tag, //etiqueta del mensaje

MPI\_Comm communicator, //comunicador

MPI\_Status\* status) // estructura estado del envio

MPI\_Reduce(

void\* data, //dirección del buffer de envio

void resultado, //dirección del buffer de recepcion

int count, //numero de elementos que se enviaran

MPI\_Datatype datatype, //tipo de dato

MPI\_OP operador, //operación de reduccion

int root, //proceso raíz o receptor

MPI\_Comm communicator) //comunicador

MPI\_Allreduce(

void\* send\_data,//dirección del buffer de envio

void\* recv\_data, //dirección del buffer de recepcion

int count, //numero de elementos que se enviaran

MPI\_Datatype datatype, //tipo de dato

MPI\_Op op, //operación de reduccion

MPI\_Comm communicator) //comunicador

**6.**

#include <math.h>

#include <iostream>

using namespace std;

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

{

int n,rank,size, count=0, elemento\_buscado, resultado;

int B[];

MPI\_Init(&argc, &argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

if (rank == 0) {

cout<<"Introduce la cantidad de elementos (n): ";

cin>>n;

cout<<"Elemento buscado es: ";

cin>>elemento\_buscado;

int A[n] = llenarVector(n);

MPI\_Bcast(&A,

n,

MPI\_INT,

0,

MPI\_COMM\_WORLD);

}else{

MPI\_Receive(&B,

n,

MPI\_INT,

0,

MPI\_COMM\_WORLD);

for(int i=rank\*n/size;i<(rank+1)\*n/size;i++){

if(B[i]==elemento\_buscado){

count++;

}

}

MPI\_Reduce(

&count,

&resultado,

1,

MPI\_INT,

MPI\_SUM,

0,

MPI\_COMM\_WORLD);

}

if(rank==0){

cout<<"El numero de veces que sale "<<elemento\_buscado<<" es: "<<resultado<<endl;

}

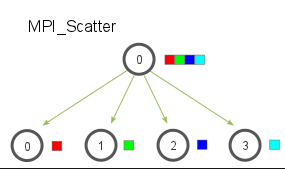
MPI\_Finalize();

return 0;

}

**8. a.**

Es una rutina colectiva que es muy similar a MPI\_Bcast (si no está familiarizado con estos términos, lea la lección anterior). MPI\_Scatter implica un proceso raíz designado que envía datos a todos los procesos en un comunicador. La diferencia principal entre MPI\_Bcast y MPI\_Scatter es pequeña pero importante. MPI\_Bcast envía la misma pieza de datos a todos los procesos, mientras que MPI\_Scatter envía fragmentos de una matriz a diferentes procesos.



**b.**

MPI\_Gather es el inverso de MPI\_Scatter. En lugar de distribuir elementos de un proceso a muchos procesos, MPI\_Gather toma elementos de muchos procesos y los reúne en un solo proceso. Esta rutina es muy útil para muchos algoritmos paralelos, como la clasificación y búsqueda paralelas.

