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

Cuando un usuario ejecuta un programa, el sistema operativo crea un proceso, una instancia de un programa de computadora que se está ejecutando. Un proceso consta de varias entidades:

. El programa ejecutable en lenguaje máquina.

. Un bloque de memoria, que incluirá el código ejecutable, una pila de llamadas que mantiene

seguimiento de funciones activas, un montón y algunas otras ubicaciones de memoria.

. Descriptores de los recursos que el sistema operativo ha asignado al proceso:

por ejemplo, descriptores de archivos.

. Información de seguridad, por ejemplo, información que especifica qué hardware y

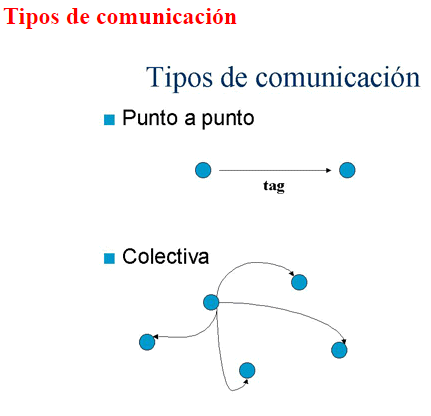
recursos de software a los que puede acceder el proceso.

. Información sobre el estado del proceso, como si el proceso está listo para

ejecutar o está esperando algún recurso, el contenido de los registros e información

sobre la memoria del proceso.

1. **Comunicación punto a punto entre procesos**



**Send y Receive Standard Bloqueantes.**

* **Mpi\_send ( IN start\_of\_buffer, IN number\_of\_items, IN datatype, IN destination\_rank, IN tag, IN communicator, OUT error\_cod )**
* **Mpi\_recv ( OUT start\_of\_buffer, IN max\_number\_of\_items, IN datatype, IN source\_rank, IN tag, IN communicator, OUT return\_status, OUT error\_cod ) source\_rank puede valer mpi\_any\_source y tag puede valer mpi\_any\_tag.**

1. **¿Qué es memoria RAM, Caché y Virtual?**

Los cachés hacen posible que la CPU acceda rápidamente a las instrucciones y datos que están en memoria principal. Sin embargo, si ejecutamos un programa muy grande o un programa que accede conjuntos de datos muy grandes, es posible que todas las instrucciones y datos no quepan en la memoria principal. Esta es especialmente cierta con los sistemas operativos multitarea; para cambiar entre programas y crean la ilusión de que varios programas se ejecutan simultáneamente, las instrucciones y los datos que se utilizarán durante la próxima porción de tiempo deben estar en main memoria. Por lo tanto, en un sistema multitarea, incluso si la memoria principal es muy grande, muchos los programas en ejecución deben compartir la memoria principal disponible. Además, este intercambio debe hacerse de tal manera que los datos e instrucciones de cada programa estén protegidos de corrupción por otros programas.

La memoria virtual se desarrolló para que la memoria principal pueda funcionar como caché para almacenamiento secundario Explota el principio de localidad espacial y temporal por mantener en la memoria principal solo las partes activas de los muchos programas en ejecución; aquellos Las partes que están inactivas se guardan en un bloque de almacenamiento secundario llamado espacio de intercambio . Me gusta Cachés de CPU, la memoria virtual opera en bloques de datos e instrucciones. Estas

Los bloques se denominan comúnmente páginas , y dado que el acceso al almacenamiento secundario puede ser importante. dragas de miles de veces más lento que el acceso a la memoria principal, las páginas son relativamente grande: la mayoría de los sistemas tienen un tamaño de página fijo que actualmente oscila entre 4 y 16 kilobytes

1. **Programación en memoria distribuida vs programación en memoria compartida.**

Los recién llegados a la computación paralela a veces se preguntan por qué no todos los sistemas MIMD memoria compartida, ya que la mayoría de los programadores encuentran el concepto de coordinación implícita El trabajo de los procesadores a través de estructuras de datos compartidos es más atractivo que enviando mensajes explícitamente. Hay varios problemas, algunos de los cuales discutiremos cuando hablamos de software para memoria distribuida y compartida. Sin embargo, el principal- El problema de hardware es el costo de escalar la interconexión. A medida que agregamos procesadores a un autobús, la posibilidad de que haya conflictos sobre el acceso al autobús aumenta dramáticamente lógicamente, los buses son adecuados para sistemas con solo unos pocos procesadores. Barras transversales grandes son muy caros, por lo que también es inusual encontrar sistemas con una gran interconexión de barras cruzadas nects. Por otro lado, las interconexiones de memoria distribuida como el hipercubo y la malla toroidal son relativamente económicos, y los sistemas de memoria distribuida Con miles de procesadores que utilizan estas y otras interconexiones se han construido. Por lo tanto, los sistemas de memoria distribuida a menudo son más adecuados para problemas que requieren vastas cantidades de datos o cálculo.

**COMANDOS DEL MPI**

**MPI\_SEND**

Realiza el envío de un mensaje de un proceso fuente a otro destino.

**Sintaxis**

**En C++**

#include <mpi.h>

void Comm::Send(const void\* buf, int count, const Datatype& datatype,

int dest, int tag) const

**Parámetros**

**De entrada**

|  |  |
| --- | --- |
| buf | Dirección inicial del buffer de envío. Esto significa que requiere un puntero. Si solo se pretende enviar un elemento, se puede enviar el puntero a este (&elemento). |
| count | Numero de elementos a enviar (Debe ser un entero no negativo). |
| datatype | Tipo de dato de cada elemento que se va a enviar. Acepta constantes definidas por [MPI](http://lsi.ugr.es/jmantas/pdp/ayuda/mpi_ayuda.php) (Véase el apartado de constantes). Por ejemplo [MPI\_INT](http://lsi.ugr.es/jmantas/pdp/ayuda/mpi_ayuda.php?ayuda=mpi_const_types). |
| dest | Rango del proceso destino. (Véase [MPI\_Comm\_rank](http://lsi.ugr.es/jmantas/pdp/ayuda/mpi_ayuda.php?ayuda=MPI_Comm_rank)) |
| tag | Entero que representa la etiqueta del mensaje. El significado de la etiqueta queda en manos del usuario, durante el proceso de envío no es modificado. |
| comm | Comunicador utilizado para la comunicación. |

[**MPI\_Recv**](http://lsi.ugr.es/jmantas/pdp/ayuda/mpi_ayuda.php?ayuda=MPI_Recv)

Rutina de recibimiento de un mensaje desde un proceso.

**Sintaxis**

**En C++**

#include <mpi.h>

void Comm::Recv(void\* buf, int count, const Datatype& datatype,

int source, int tag, Status& status) const

void Comm::Recv(void\* buf, int count, const Datatype& datatype,

int source, int tag) const

**Parámetros**

**De entrada**

|  |  |
| --- | --- |
| count | Entero que indica el número máximo de elementos que se espera recibir en el buffer de entrada. |
| datatype | Tipo de dato de cada elemento que se va a recibir. Acepta constantes definidas por [MPI](http://lsi.ugr.es/jmantas/pdp/ayuda/mpi_ayuda.php) (Véase el apartado de constantes). Por ejemplo [MPI\_INT](http://lsi.ugr.es/jmantas/pdp/ayuda/mpi_ayuda.php?ayuda=mpi_const_types). |
| source | Rango del proceso de origen esperado, solo se recogen mensajes cuyo origen sea el especificado. Se acepta el valor [MPI\_ANY\_SOURCE](http://lsi.ugr.es/jmantas/pdp/ayuda/mpi_ayuda.php?ayuda=mpi_const_types) , el cual recoge de cualquier proceso origen. (Véase también [MPI\_Comm\_rank](http://lsi.ugr.es/jmantas/pdp/ayuda/mpi_ayuda.php?ayuda=MPI_Comm_rank)) |
| tag | Entero que representa la etiqueta del mensaje. Solo se recogerá un mensaje con la etiqueta especificada. Se acepta el valor [MPI\_ANY\_TAG](http://lsi.ugr.es/jmantas/pdp/ayuda/mpi_ayuda.php?ayuda=mpi_const_types) , que recoge con cualquier etiqueta. El significado de la etiqueta queda en manos del usuario, durante el proceso de envío no es modificado. |
| comm | Comunicador utilizado para la comunicación. Solo se recogerán mensajes que han sido enviados por el comunicador seleccionado. |

**De salida**

|  |  |
| --- | --- |
| buf | Buffer de entrada en el que se guarda el contenido del mensaje enviado. Recibe un puntero al comienzo del buffer. |
| status | Objeto de tipo [MPI\_Status](http://lsi.ugr.es/jmantas/pdp/ayuda/mpi_ayuda.php?ayuda=mpi_datatypes), contiene datos relevantes sobre el mensaje (como son el origen (MPI\_SOURCE) , la etiqueta (MPI\_TAG)y el tamaño (size)). |

**MPI-Reduce**

Las funciones de reducción (tales como [MPI\_Reduce](http://lsi.ugr.es/jmantas/pdp/ayuda/mpi_ayuda.php?ayuda=MPI_Reduce), [MPI\_Allreduce](http://lsi.ugr.es/jmantas/pdp/ayuda/mpi_ayuda.php?ayuda=MPI_Allreduce), [MPI\_Scan](http://lsi.ugr.es/jmantas/pdp/ayuda/mpi_ayuda.php?ayuda=MPI_Scan)...) realizan una operación de reducción global (como puede ser calcular el máximo, la suma, hacer un AND lógico, etc) sobre cada uno de los miembros del grupo. La operación de reducción puede ser tanto una predefinida de la lista de operaciones como una operación definida por el usuario.  
Se puede notar que únicamente se especifica una vez el número de datos y el tipo de datos, esto es porque el número de datos es el mismo tanto en la emisión como en la recepción, al igual que el tipo. Si el número de datos es mayor que uno, la operación de reducción se realiza uno a uno cada elemento con su posición correspondiente (es decir, el elemento 1 de envío se reduce con cada elemento en la posición 1 de cada sendbuf).

## En C++

#include <mpi.h>

void [MPI](http://lsi.ugr.es/jmantas/pdp/ayuda/mpi_ayuda.php)::Intracomm::Reduce(const void\* sendbuf, void\* recvbuf,

int count, const [MPI](http://lsi.ugr.es/jmantas/pdp/ayuda/mpi_ayuda.php)::Datatype& datatype, const [MPI](http://lsi.ugr.es/jmantas/pdp/ayuda/mpi_ayuda.php)::Op& op,

int root) const

**MPI\_Allreduce**

Igual que [MPI\_Reduce](http://lsi.ugr.es/jmantas/pdp/ayuda/mpi_ayuda.php?ayuda=MPI_Reduce), salvo que el resultado se redistribuye entre todos los procesos.  
Las funciones de reducción (tales como [MPI\_Reduce](http://lsi.ugr.es/jmantas/pdp/ayuda/mpi_ayuda.php?ayuda=MPI_Reduce), [MPI\_Allreduce](http://lsi.ugr.es/jmantas/pdp/ayuda/mpi_ayuda.php?ayuda=MPI_Allreduce), [MPI\_Scan](http://lsi.ugr.es/jmantas/pdp/ayuda/mpi_ayuda.php?ayuda=MPI_Scan)...) realizan una operación de reducción global (como puede ser calcular el máximo, la suma, hacer un AND lógico, etc) sobre cada uno de los miembros del grupo. La operación de reducción puede ser tanto una predefinida de la lista de operaciones como una operación definida por el usuario.  
Se puede notar que únicamente se especifica una vez el número de datos y el tipo de datos, esto es porque el número de datos es el mismo tanto en la emisión como en la recepción, al igual que el tipo. Si el número de datos es mayor que uno, la operación de reducción se realiza uno a uno cada elemento con su posición correspondiente (es decir, el elemento 1 de envío se reduce con cada elemento en la posición 1 de cada sendbuf).  
Lista de operaciones disponibles:

## En C++

#include <mpi.h>

void [MPI](http://lsi.ugr.es/jmantas/pdp/ayuda/mpi_ayuda.php)::Comm::Allreduce(const void\* sendbuf, void\* recvbuf,

int count, const [MPI](http://lsi.ugr.es/jmantas/pdp/ayuda/mpi_ayuda.php)::Datatype& datatype, const

[MPI](http://lsi.ugr.es/jmantas/pdp/ayuda/mpi_ayuda.php)::Op& op) const=0

**8 Movimiento colectivo de datos**

