

Informe 02 - Algoritmos Paralelos

Lorena Xiomara Castillo Galdos

Abril 2017

Contents

1	Comunicación punto a punto	2
1.1	Modelos y modos de comunicación	2
1.2	Blocking	2
1.2.1	Basic	3
1.2.2	Buffered	3
1.2.3	Synchronous	3
1.2.4	Ready	3
1.3	Ejemplo	3
2	Nonblocking	3
2.1	Ejemplo	5

1 Comunicación punto a punto

Un buen número de funciones de MPI están dedicadas a la comunicación entre pares de procesos. Existen múltiples formas distintas de intercambiar un mensaje entre dos procesos, en función del modelo y el modo de comunicación elegido. [1].

1.1 Modelos y modos de comunicación

MPI define dos modelos de comunicación: *blocking* y *nonblocking*. El modelo de comunicación tiene que ver con el tiempo que un proceso pasa bloqueado tras llamar a una función de comunicación, sea ésta de envío o de recepción. Una función *blocking* mantiene a un proceso bloqueado hasta que la operación solicitada finalice, mientras que una *nonblocking* encarga al sistema la realización de una operación, recuperando el control inmediatamente. El proceso tiene que preocuparse, más adelante, de averiguar si la operación ha finalizado o no.

En el caso de la recepción cuando se tiene un mensaje nuevo en el buffer asignado, podemos dar como finalizada la operación, sin embargo, en la emisión se puede entender que ha finalizado cuando el mensaje ha sido recibido en el destino, o también se puede dar como terminada la operación en cuanto se hace una copia del mensaje en un buffer del sistema en el lado del emisor. MPI define un envío como finalizado cuando el emisor puede reutilizar sin problemas de causar interferencias, el buffer de emisión que tenía el mensaje.

En la tabla 1 se muestran las funciones de MPI para *blocking* y *nonblocking*.

Rutinas de MPI		
SEND	BLOCKING	NONBLOCKING
Basic	MPI_send	MPI_isend
Buffered	MPI_bsend	MPI_ibsend
Synchronous	MPI_ssend	MPI_issend
Ready	MPI_rsend	MPI_irsend
RECEIVE	BLOCKING	NONBLOCKING
Estándar	MPI_recv	MPI_irecv

Table 1: Funciones de MPI.

1.2 Blocking

MPI define 4 modos de envío: *basic*, *buffered*, *synchronous* y *ready*. Las funciones de envío y recepción básica de *blocking* son MPI_Send() y MPI_Receive() respectivamente (Figura 1).

```
int MPI_Send(void* buf, int count, MPI_Datatype datatype,
int dest, int tag, MPI_Comm comm);
int MPI_Recv(void* buf, int count, MPI_Datatype datatype,
int source, int tag, MPI_Comm comm, MPI_Status *status);
```

Figure 1: Funciones send y receive.

1.2.1 Basic

El modo de envío *basic* normalmente equivale a un envío con buffer para mensajes cortos y a un envío síncrono para mensajes largos. Se intenta así agilizar el envío de mensajes cortos a la vez que se procura no perder demasiado tiempo realizando copias de la información.

1.2.2 Buffered

Cuando se hace un envío con *buffer* se guarda inmediatamente en un *buffer* una copia del mensaje. La operación se da por completa en cuanto se ha efectuado esta copia. Si no hay espacio en el buffer, el envío fracasa.

1.2.3 Synchronous

Si se hace un envío síncrono, la operación se da por terminada sólo cuando el mensaje ha sido recibido en el destino. Este es el modo de comunicación habitual en los sistemas basados en Transputers. En función de la implementación elegida, puede exigir menos copias de la información conforme ésta circula del *buffer* del emisor al *buffer* del receptor.

1.2.4 Ready

Sólo se puede hacer si antes el otro extremo está preparado para una recepción inmediata. No hay copias adicionales del mensaje como en el caso del modo con buffer, y tampoco podemos confiar en bloquearnos hasta que el receptor esté preparado.

1.3 Ejemplo

A continuación en la Figura 2 se muestra los resultados del código de un programa en C para la ejecución de comunicación de procesos tipo *blocking*, no se muestra el código ya que es muy extenso.

2 Nonblocking

Una de los problemas que tiene el envío básico es que el programador no tiene control sobre cuánto tiempo va a tardar en completarse la operación. Puede que apenas tarde, si es que el sistema se limita a hacer una copia del mensaje

```
lorena@Lorena-pc: ~/Documentos/Paralelos/Tarea2
lorena@Lorena-pc:~/Documentos/Paralelos/Tarea2$ mpirun -np 2 ./block 2
Message size = 2 floats
Task 1 initialized
Message size = 2 floats
Task 0 initialized
Ready to send messages
Elapsed time for synchronous send = 10 uSec
Elapsed time for ready send = 2 uSec
Elapsed time for buffer allocation = 4 uSec
Elapsed time for buffered send = 2 uSec
Elapsed time for standard send = 1 uSec
lorena@Lorena-pc:~/Documentos/Paralelos/Tarea2$
```

Figure 2: Resultado en el cual se muestra el envío de datos con los diferentes métodos de envío blocking.

en un buffer, que saldrá más tarde hacia su destino; pero también puede que mantenga al proceso bloqueado un largo tiempo, esperando a que el receptor acepte el mensaje. Para evitar el riesgo de un bloqueo no deseado se puede solicitar explícitamente que la comunicación se complete copiando el mensaje en un buffer, que tendrá que asignar al efecto el propio proceso. Así, se elimina el riesgo de bloqueo mientras se espera a que el interlocutor esté preparado. La función correspondiente es `MPI_Bsend()`, que tiene los mismos argumentos que `MPI_Send()`

Para que se pueda usar el envío con buffer es necesario que el programador asigne un buffer de salida. Para ello hay reservar previamente una zona de memoria (de forma estática o con `malloc()`) y luego indicarle al sistema que la emplee como buffer. Esta última operación la hace `MPI_Buffer_attach()`. Ese buffer se puede recuperar usando `MPI_Buffer_detach()`.

Recordemos que una peculiaridad de los envíos con buffer es que fracasan en el caso de que el buffer no tenga suficiente espacio como para contener un mensaje, y el resultado es que el programa aborta. El programador puede decidir entonces que el buffer asignado es muy pequeño, y asignar uno más grande para una ejecución posterior, o bien cambiar el modo de comunicación a otro con menos requerimientos de buffer pero con más riesgo de bloqueo.

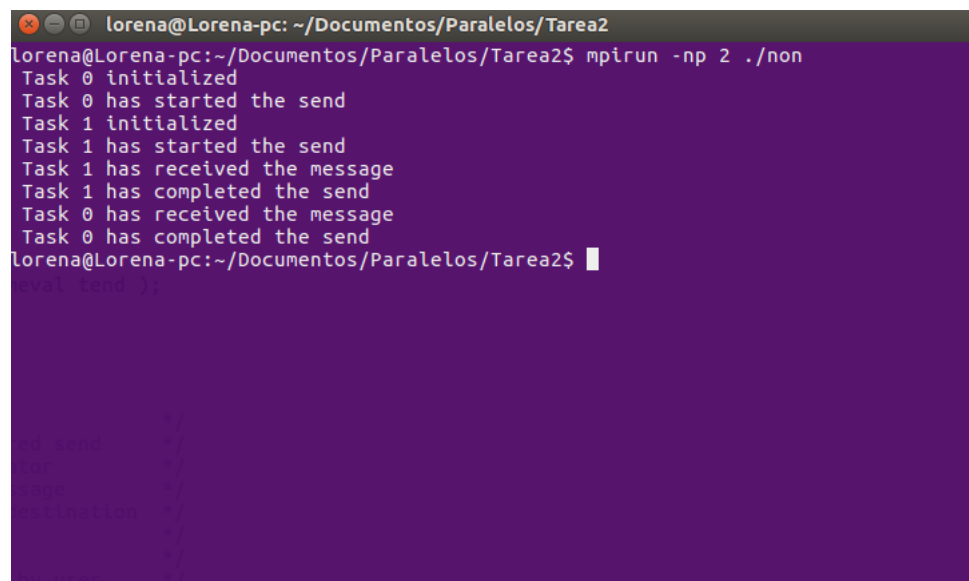
Las funciones no bloqueantes manejan un objeto *request* del tipo `MPI_Request`. Este objeto es una especie de "recibo" de la operación solicitada. Más adelante se podrá utilizar este recibo para saber si la operación ha terminado o no.

Cuando no interesa bloquearse, sino simplemente saber si la operación ha

terminado o no, podemos usar `MPI_Test()`. Esta función actualiza un flag que se le pasa como segundo parámetro. Si la función ha terminado, este flag toma el valor 1, y si no ha terminado pasa a ser 0.

2.1 Ejemplo

A continuación en la Figura 3 se muestra los resultados del código de un programa en C para la ejecución de comunicación de procesos tipo *nonblocking*, no se muestra el código ya que es muy extenso.

A terminal window with a dark background and light text. The title bar shows 'lorena@Lorena-pc: ~/Documentos/Paralelos/Tarea2'. The prompt is 'lorena@Lorena-pc:~/Documentos/Paralelos/Tarea2\$'. The command entered is 'mpirun -np 2 ./non'. The output shows a sequence of messages from two tasks: 'Task 0 initialized', 'Task 0 has started the send', 'Task 1 initialized', 'Task 1 has started the send', 'Task 1 has received the message', 'Task 1 has completed the send', 'Task 0 has received the message', and 'Task 0 has completed the send'. The prompt returns to 'lorena@Lorena-pc:~/Documentos/Paralelos/Tarea2\$'.

```
lorena@Lorena-pc:~/Documentos/Paralelos/Tarea2$ mpirun -np 2 ./non
Task 0 initialized
Task 0 has started the send
Task 1 initialized
Task 1 has started the send
Task 1 has received the message
Task 1 has completed the send
Task 0 has received the message
Task 0 has completed the send
lorena@Lorena-pc:~/Documentos/Paralelos/Tarea2$
```

Figure 3: Resultados programa nonblocking, mostrando que el proceso 0 y 1 han inicializado, enviado y recibido con éxito.

References

- [1] Peter Pacheco. *An introduction to parallel programming*. Elsevier, 2011.