# Tecnologías de la Información. Redes, Comunicaciones y Arquitecturas Distribuidas.

## AÑO 2010

## Trabajo Práctico N° 3

## Programación de Flooding con MPI

**Profesor:**

- MSc. Pablo Pessolani

**Integrantes:**

- Lorena Diorio
lorenadiorio@gmail.com

- Ariel Rossanigo
arielrossanigo@gmail.com

- Román Zenobi
rozenobi@hotmail.com

# Desarrollo del Práctico

## *MPI Flood (mpiflood.c)*

```
#include <stdio.h>
#include <string.h>
#include "mpi.h"
//incluimos funciones comunes
#include "utiles_ping.c"

int main(int argc, char* argv[]){
  int rank, size;
  int repeticiones = 5;
  int tamanio=100;
  //Esta funcion se encarga de completar los parametros repeticiones y tamaño
  parsear_parametros_cliente(argc, argv, &repeticiones, &tamanio);

  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  //buscamos el nombre del host para no afectar las mediciones más adelante
  char * hostname = nombre_host();

  printf("Rank: %d Cant. servidores: %d Nombre servidor: %s\n", rank, size, hostname);

  //preparamos el mensaje con el tamaño ingresado por el usuario
  char * mensaje;
  int largo_mensaje;
  mensaje = generar_paquete(tamanio);
  largo_mensaje = tamanio;

  int r;
  for (r = 0 ; r < repeticiones; r++)
  {
    if (rank == 0)
    {
      iniciar_temporizador();
      //Enviamos el mensaje a todos los hosts del anillo
      MPI_Bcast((void *)mensaje, largo_mensaje, MPI_CHAR, rank, MPI_COMM_WORLD);
      printf("Se envio el mensaje %d desde el host %s y rank: %d\n",r, hostname, rank);
      //Esperamos la respuesta de cada host
      int i;
      for (i=1;i<size;i++)
      {
        MPI_Bcast(mensaje, largo_mensaje, MPI_CHAR, i, MPI_COMM_WORLD);
        printf("%s - Se recibio el mensaje desde el rank: %d\n", hostname, i);
      }
      //Imprimimos RTT de esta repeticion
      printf("RTT flooding %d: %d uSeg\n", r+1,finalizar_temporizador());
    }
    else
    {
        //Esperamos mensaje desde el proceso 0
        MPI_Bcast(mensaje, largo_mensaje, MPI_CHAR, 0, MPI_COMM_WORLD);
        printf("%s - Se recibio notificacion desde el rank: %d\n", hostname, 0);
        //Respondemos con un broadcast a todos los hosts del grupo
        MPI_Bcast( (void *) mensaje, largo_mensaje, MPI_CHAR, rank, MPI_COMM_WORLD);
        printf("Se envio el mensaje %d desde el host %s y rank: %d\n",r, hostname, rank);
    }
  }
  if (rank==0)
  {
    //Imprimimos los resultados finales
    imprimir_resultados();
  }

  MPI_Finalize();
  return 0;
}
```

## ***utiles_ping.c***

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include <time.h>
#include <unistd.h>

//Devuelve el nombre del host
char * nombre_host()
{
  size_t len = 126;
  char * hostname = malloc(128);
  gethostname (hostname, len);
  return hostname;
}

//completa repeticiones y tamanio de acuerdo a los parametros de entrada
int parsear_parametros_cliente(int argc, char* argv[], int *repeticiones, int *tamanio)
{
  int c;
  while ((c= getopt(argc, argv, "r::s::")) != -1)
  {
    switch(c)
    {
      case 'r':
      {
        *repeticiones = atoi(optarg);
        if (*repeticiones < 1 || *repeticiones > 101)
        {
          printf ("El nro de repeticiones debe estar comprendido entre 1 y 101\n");
          return -1;
        }
        break;
      }
      case 's':
      {
        *tamanio = atoi(optarg);
        if (*tamanio <1 || *tamanio > 10001)
        {
          printf ("El tamano debe estar comprendido entre 1 y 10001\n");
          return -1;
        }
        break;
      }
    }
  }
  return 0;
}

//Genera un string con los digitos del 0 al 9, de tamanio caracteres
char* generar_paquete(int tamanio)
{
  char* res = malloc((tamanio+1)*sizeof(char));
  int i;
  for (i = 0; i<tamanio; i++)
  {
    res[i] = (char)(i%10 + 48);
  }
  res[tamanio]='\0';
  return res;
}

//Variables utilizadas en los temporizadores
struct timeval start;
struct timeval stop;
struct timezone tz;
```

```
//Calcula la diferencia entre dos valores de tiempo y la almacena en out
void tvsub( out, in )
struct timeval *out, *in;
{
  if( (out->tv_usec -= in->tv_usec) < 0 )
  {
    out->tv_sec--;
    out->tv_usec += 1000000;
  }
  out->tv_sec -= in->tv_sec;
}

struct ResultadosTemporizador
{
  int Minimo;
  int Maximo;
  int Total;
  int Cantidad;
  float Promedio;
};

//variable para llevar los resultados acumulados del todas las mediciones
struct ResultadosTemporizador res;

//inicializamos res
void iniciar_resultados_temporizador()
{
  res.Minimo = 0;
  res.Maximo = -1;
  res.Cantidad= 0;
  res.Total = 0;
}

//Iniciamos una medicion, colocamos en start la hora del dia
void iniciar_temporizador()
{
  gettimeofday( &start, &tz );
}

//finalizamos una medicion, calculamos la diferencia entre la hora del dia y start
//actualizamos los valores acumulados
int finalizar_temporizador()
{
  int tiempo;
  gettimeofday( &stop, &tz );
  tvsub( &stop, &start );
  tiempo =  stop.tv_sec * 1000000 + stop.tv_usec;
  if (res.Minimo > tiempo || res.Cantidad==0) res.Minimo = tiempo;
  if (res.Maximo < tiempo || res.Cantidad==0) res.Maximo = tiempo;
  res.Total += tiempo;
  res.Cantidad++;
  return tiempo;
}

//Obtenemos los resultados acumulados de todas las mediciones
struct ResultadosTemporizador obtener_resultados_temporizador()
{
    if (res.Cantidad!= 0)
        res.Promedio = (float)res.Total/res.Cantidad;
    else
        res.Promedio = 0;
    return res;
}
```

```
//Imprimimos los resultados acumulados
void imprimir_resultados()
{
  struct ResultadosTemporizador r= obtener_resultados_temporizador();
  printf("=========================================================\n");
  printf("RTT min: %d uSeg, max: %d uSeg, prom: %.2f uSeg\n", r.Minimo, r.Maximo,
r.Promedio);
  printf("=========================================================\n");
}

char * str_hora_actual()
{
  struct tm *ptr;
  time_t lt;
  //obtenemos tiempo actual
  lt = time(NULL);
  //obtenemos tiempo local
  ptr = localtime(&lt);
  //obtenemos un string formateado con el tiempo local
  char * res= malloc(9*sizeof(char));
  strftime(res, 9, "%H:%M:%S", ptr);
  return res;
}
```

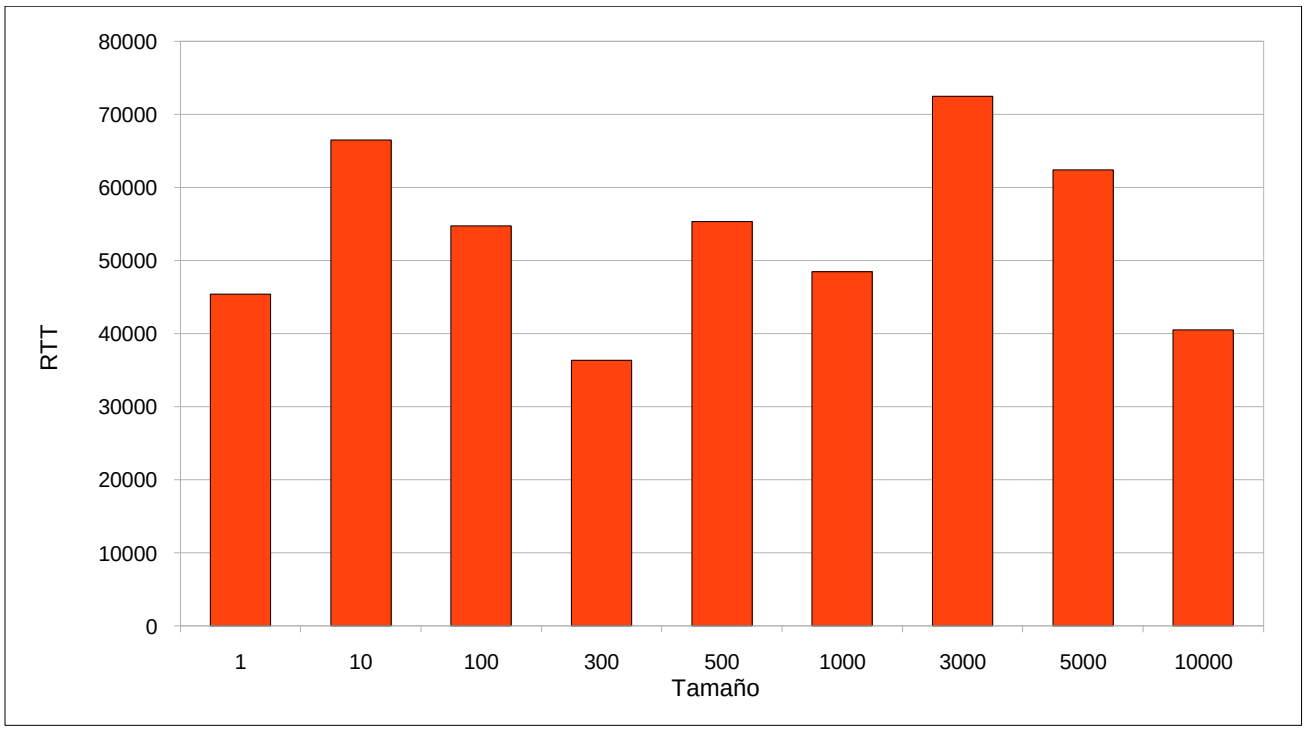
## ***Makefile***

```
TP3:
	mpicc -o ../obj/mpiflood ../src/mpiflood.c
clean:
	rm ../obj/*

copy:
	if test -d /home/sod/2010/tp3/Zenobi-DIorio-Rossanigo;
		then echo "el path ya existe";
		else mkdir /home/sod/2010/tp3/Zenobi-DIorio-Rossanigo;
	fi
	cp -R -f -u ../* /home/sod/2010/tp3/Zenobi-DIorio-Rossanigo
```

## ***Mediciones***

| Tamaño | RTT Promedio (uSeg) |
|---|---|
| 1 | 45400 |
| 10 | 66485 |
| 100 | 54738 |
| 300 | 36402 |
| 500 | 55335 |
| 1000 | 48468 |
| 3000 | 72482 |
| 5000 | 62417 |
| 10000 | 40558 |




- ***Las pruebas se realizaron en 3 hosts.***