**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](mailto:lorenadiorio@gmail.com)

* Ariel Rossanigo

[arielrossanigo@gmail.com](mailto:arielrossanigo@gmail.com)

* Román Zenobi

[rozenobi@hotmail.com](mailto: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***](http://code.google.com/p/pracmaestriaemisi/source/browse/trunk/Sistemas%20distribuidos/guia1/src/utiles_ping.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***](http://code.google.com/p/pracmaestriaemisi/source/browse/trunk/Sistemas%20distribuidos/guia1/src/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.***