



Código Fuente:

- Proceso de distribución
- Demonio de carga



- Proceso de distribución

```
/*
 * Proceso distrib.c
 * Proceso encargado de realizar las labores de distribucion
 *
 * Version: 1.0          Fecha: 6 - 9 - 2000
 * Autor: Diego Chaparro Gonzalez
 *
 */
*****/

#include "../include/bdd.h"
#include <errno.h>

#define WORKTAG          1
#define DIETAG           2

// Longitud de las lineas que indican el path de las imagenes
#define LONGLINEA        68

// Numero de firmas total a procesar por el nodo actual
long total_firmas;

// Parámetros para el inicializar el MPI
MPI_Comm MPI_COMM_DIST;
int sizeDIST, rankDIST, size, rank;

/*
 * -----
 * FUNCION Numero_Tareas_Medio:
 * -----
 * Devuelve el numero de tareas medio en el ultimo minuto.
 * Este dato se obtiene del fichero /proc/loadavg
 * -----
 */
*****/
float Numero_Tareas_Medio (void) {

    FILE *fd;
    char cadnum1[10], cadnum2[10], cadnum3[10], cadnum[10];
    float numero;

    fd = fopen ("/proc/loadavg", "r");

    fscanf (fd, "%s %s %s %s", cadnum1, cadnum2, cadnum3, cadnum);

    numero = atof (cadnum1);

    fclose(fd);

    return (numero);
}

/*
 * -----
 * FUNCION Mi_Velocidad:
 * -----
 * Devuelve la velocidad inicial de cada nodo almacenada en el
 * fichero veloc
 * -----
 */
*****/
float Mi_Velocidad (void) {

    FILE *fd;
    char cadnum[10];
    float numero;

    fd = fopen ("/var/tmp/distribuidoDiego/exe/veloc", "r");

    fscanf (fd, "%s", cadnum);

    numero = atof (cadnum);

    return (numero);
}

```



```

/*****
 *                               FUNCION Numero_Firmas:                               *
 * -----
 * Recibe como parámetro un fichero y devuelve el número de                     *
 * líneas del mismo                                                           *
 *****/
long Numero_Firmas (char fichero[100]) {
    long num;
    char comando[200]= "wc -l >/var/tmp/distribuidoDiego/exe/numlin ";
    char cadaux[10];
    int cod;

    FILE *fd;

    strcat (comando, fichero);

    cod = system (comando);
    if (cod==127 || cod==-1)
        printf ("ERROR: Al ejecutar system");
    fd = fopen ("/var/tmp/distribuidoDiego/exe/numlin", "r");

    fscanf (fd, "%s", cadaux);

    num = atol (cadaux);
    fclose (fd);

    return (num);
}

/*****
 *                               FUNCION Inicializar_MPI:                               *
 * -----
 * Inicializa MPI, crea el comunicador, halla el rango y el tamaño,... *
 *****/
void Inicializar_MPI (void) {

    int color;

    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank (MPI_COMM_WORLD, &rank);

    // Crea el comunicador MPI_COMM_DIST
    color=1;
    MPI_Comm_split (MPI_COMM_WORLD, color, rank, &MPI_COMM_DIST);
    MPI_Comm_rank (MPI_COMM_DIST, &rankDIST);
    MPI_Comm_size (MPI_COMM_DIST, &sizeDIST);
}

/*****
 *                               PROCEDIMIENTO Pedir_Carga                               *
 * -----
 * Cuando el slave ha terminado, este procedimiento pide la información *
 * sobre el nodo adecuado para realizar la transferencia, y realiza los *
 * pasos necesarios para llevarla a cabo.                                     *
 *****/
int Pedir_Carga (void) {
    int nodotrans; // Indica el número de nodo con el que se realizará la
    transferencia
    MPI_Status status;
    int buffer;
    long bufferlong;
    long nfirmas; // nº de firmas a recibir
    char *firmas; // todas las firmas recibidas
    int fd;
    long tamano;
    int iteracion;
    FILE *fd2;
    int salir, test;
    int velocidad;

```



```
buffer=0;
bufferlong = 0;

iteracion = 0;
nfirmas = 0;

velocidad = (int) (Mi_Velocidad() / Numero_Tareas_Medio());

// Permanece en el siguiente bucle hasta que obtiene un nodo adecuado para realizar una
// transferencia, o hasta que comprueba que ninguno de los nodos posee carga para ser
// transferida

while (nfirmas<1 && iteracion<sizeDIST) {

    // Envia peticion al demonio de carga de numero de nodo para transferencia
    MPI_Send (&iteracion, 1, MPI_LONG, (rank + sizeDIST), 1, MPI_COMM_WORLD);

    // Recibe el nodo para la transferencia
    MPI_Recv (&nodotrans, 1, MPI_INT, (rank + sizeDIST), 4, MPI_COMM_WORLD, &status);

    if (nodotrans!=rankDIST)
        if (nodotrans==-1) // No hay nodos disponibles para transferencia
            iteracion = sizeDIST;
        else{
            // Envia al nodo seleccionado peticion de carga
            MPI_Send (&velocidad, 1, MPI_INT, nodotrans +(sizeDIST+1), 1,
MPI_COMM_WORLD);

            salir = 0;
            while (!salir) {
                // Comprueba si recibe la respuesta del nodo
                MPI_Iprobe (nodotrans, 2, MPI_COMM_DIST, &test, &status);
                if (test)
                    salir = 1;
                else {
                    // Si recibe peticion de otro nodo le responde que no
                    // Esto se hace para evitar inter bloqueos
                    MPI_Iprobe (MPI_ANY_SOURCE, 1, MPI_COMM_WORLD, &test,
&status);

                    if (test) {
                        MPI_Recv (&buffer, 1, MPI_INT,
status.MPI_SOURCE, status.MPI_TAG, MPI_COMM_WORLD, &status);
                        bufferlong = 0;
                        MPI_Send (&bufferlong, 1, MPI_LONG,
status.MPI_SOURCE-sizeDIST-1, 2, MPI_COMM_DIST);
                    }
                }
            }

            // Recibo el numero de firmas que puede pasarme el nodo
            seleccionado
            MPI_Recv (&nfirmas, 1, MPI_LONG, nodotrans, 2, MPI_COMM_DIST,
&status);

            if (nfirmas>0) iteracion = 1;
        }
        iteracion ++;
    }

    // Si he recorrido toda la lista o no hay nodos disponibles, el numero de firmas es 0
    if (nodotrans==-1 || iteracion>=sizeDIST)
        nfirmas = 0;

    iteracion = 0;
    // Le aviso al demonio de carga de que finaliza el proceso de eleccion de nodo
    MPI_Send (&iteracion, 1, MPI_LONG, rank+sizeDIST, 1, MPI_COMM_WORLD);

    // Si otro nodo tiene firmas para pasarme
    if (nfirmas>0) {
        // Recibo el tamaño del paquete que voy a recibir
        MPI_Recv (&tamano, 1, MPI_LONG, nodotrans, 5, MPI_COMM_DIST, &status);

        // Reservo memoria para el paquete que voy a recibir
        firmas = (char*) malloc (tamano);
    }
}
```



```
// Recibo las firmas para procesar
MPI_Recv (firmas, tamaño, MPI_CHAR, nodotrans, 3, MPI_COMM_DIST, &status);

// Guardo las firmas en el fichero para el solver
fd = open ("/var/tmp/distribuidoDiego/exe/firmas.en2", O_TRUNC | O_CREAT |
O_WRONLY);
write (fd, firmas, tamaño);
close(fd);

// Calculo el nuevo numero de firmas, y lo meto en el fichero totalfirmas
total_firmas = nfirmas;
fd2 = fopen ("/var/tmp/distribuidoDiego/exe/totalfirmas", "w");
fprintf (fd2, "%ld", total_firmas);
fclose (fd2);
fd2 = fopen ("/var/tmp/distribuidoDiego/exe/firmasreali", "w");
bufferlong = 0;
fprintf (fd2, "%ld", bufferlong);
fclose (fd2);

// Mando mensaje al slave para que vuelva a ejecutar
MPI_Send (&buffer, 1, MPI_INT, (rank - sizeDIST), 0, MPI_COMM_WORLD);
// Mando mensaje al demonio de carga con el nuevo numero de firmas
MPI_Send (&nfirmas, 1, MPI_LONG, (rank + sizeDIST), 4, MPI_COMM_WORLD);
return (0);
} else
return (1);
}

/*****
*          PROCEDIMIENTO Mandar_Carga          *
* ----- *
* Procedimiento que se ejecuta cuando se recibe una petición de *
* carga. Se calcula el porcentaje realizado, y si es posible, se *
* manda el trabajo al nodo destino *
*****/
void Mandar_Carga (int nododest, int veloc) {

long firmasreali; // Número de firmas realizadas hasta el momento
double porcreali; // Porcentaje realizado hasta el momento
long nfirmas;
char *firmas;
char comando[200], cadaux[15];
int fd;
int cod;
long leidos;
FILE *fd2;
FILE *fd3;
int velocidad;
long mifirmasordenar, susfirmasordenar;

// Calculo el rango del nodo que solicita carga en MPI_COMM_DIST
nododest = nododest - (sizeDIST) -1;

// Obtengo el numero de firmas procesadas hasta el momento
fd3 = fopen ("/var/tmp/distribuidoDiego/exe/firmasreali", "r");
fscanf (fd3, "%ld", &firmasreali);
fclose (fd3);

// Hallo la velocidad relativa del nodo actual
velocidad = (int) (Mi_Velocidad() / Numero_Tareas_Medio());

// Calculo el porcentaje realizado en el nodo actual
porcreali = ((float)firmasreali / (float)total_firmas)*100;

// Calculo el numero de firmas que puedo pasar al nodo
nfirmas = (int) (((float)veloc / (float)(veloc+velocidad))*(total_firmas-firmasreali));

// Calculo el tiempo de ordenacion
mifirmasordenar = (int) ( ((float) (total_firmas-nfirmas))/10000) * (float)velocidad;
susfirmasordenar = (int)((float)(nfirmas+mifirmasordenar)/10000)*veloc;

// Obtengo las firmas despues de procesar el tiempo de ordenacion
if (nfirmas>2000)
nfirmas = nfirmas+mifirmasordenar-susfirmasordenar;
```



```
if ((total_firmas-nfirmas)<firmasreali)
    nfirmas = total_firmas-firmasreali;

// Si el numero de firmas es mayor de un umbral
if (nfirmas>2000) {

    // Mando al nodo origen el numero de firmas a transferir
    MPI_Send (&nfirmas, 1, MPI_LONG, nododest, 2, MPI_COMM_DIST);

    // Reservo espacio para guardar las firmas a mandar
    firmas = (char*) malloc ( nfirmas * LONGLINEA );

    // Meto en un fichero las firmas a transferir
    strcpy (comando, "sed </var/tmp/distribuidoDiego/exe/firmas.en2
>/var/tmp/distribuidoDiego/exe/firmas.en2tmp -n ' ";
    sprintf (cadaux, "%ld", (total_firmas-nfirmas));
    strcat (comando, cadaux);
    strcpy (cadaux, ",");
    strcat (comando, cadaux);
    sprintf (cadaux, "%ld", total_firmas);
    strcat (comando, cadaux);
    strcpy (cadaux, "p");
    strcat (comando, cadaux);
    cod = system (comando);
    total_firmas= total_firmas-nfirmas;
    if (cod==127 || cod==--1)
        printf ("ERROR: Al ejecutar system (tail)");

    // Abro el fichero y extraigo las firmas a transferir
    if ((fd = open ("/var/tmp/distribuidoDiego/exe/firmas.en2tmp", O_RDONLY))==--1)
        perror ("En el open");
    if ((leidos = read (fd, firmas, nfirmas * LONGLINEA))==--1)
        perror ("En el read: ");

    // Mando el tamaño del paquete que se va a mandar
    MPI_Send (&leidos, 1, MPI_LONG, nododest, 5, MPI_COMM_DIST);

    // Mando las firmas a transferir
    MPI_Send (firmas, leidos, MPI_CHAR, nododest, 3, MPI_COMM_DIST);

    // Informo al demonio de carga de la transferencia de firmas
    MPI_Send (&nfirmas, 1, MPI_LONG, (rank + sizeDIST), 2, MPI_COMM_WORLD);

    fd2 = fopen ("/var/tmp/distribuidoDiego/exe/totalfirmas", "w");
    fprintf (fd2, "%ld", total_firmas);
    fclose (fd2);
} else {
    // Mando al nodo origen que no hay firmas para transferir
    nfirmas = 0;
    MPI_Send (&nfirmas, 1, MPI_INT, nododest, 2, MPI_COMM_DIST);
}
}

/*****
*                               PROGRAM PRINCIPAL                               *
*****/

int main (int argc, char *argv[])
{
    int buffer, buffer2;
    int terminar;
    MPI_Status status;

    MPI_Init (&argc, &argv);
    Inicializar_MPI ();

    total_firmas = Numero_Firmas ("/var/tmp/distribuidoDiego/exe/firmas.en2");
    terminar = 0;

    // Permanece en el bucle hasta que no haya nodos para transferir
```



```
while (!terminar) {  
  
    // Espero hasta que recibo un mensaje  
    MPI_Recv (&buffer, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD,  
&status);  
  
    switch (status.MPI_TAG) {  
        case 0: // SLAVE ha terminado y tengo que pedir carga  
            terminar = Pedir_Carga ();  
            break;  
        case 1: // recibo petición de carga  
            Mandar_Carga (status.MPI_SOURCE, buffer);  
            break;  
        default:  
            printf ("%dDISTRIB: RECIBO TAG INVALIDO\n", rankDIST);  
            exit (1);  
    }  
}  
  
//Cuando el slave ha terminado pero no hay nodos para darme carga  
  
// Mando mensaje al slave para que termine  
MPI_Send (&buffer2, 1, MPI_INT, ( rank - sizedIST ), 1, MPI_COMM_WORLD);  
  
// Mando mensaje a carga para decirle que no hay nodo para transferencia y espere a  
terminar  
MPI_Send (&buffer, 1, MPI_INT, ( rank + sizedIST ), 3, MPI_COMM_WORLD);  
  
// Permanece en el bucle hasta que recibe mensaje del master para terminar  
terminar = 0;  
buffer = 0;  
while (!terminar) {  
  
    MPI_Recv (&buffer, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD,  
&status);  
  
    if (status.MPI_SOURCE==0) // Recibo del master y termino  
        terminar = 1;  
    else if (status.MPI_TAG==1) { // Recibo petición de carga y respondo que no hay  
firmas  
        buffer = 0;  
        MPI_Send (&buffer, 1, MPI_INT, (status.MPI_SOURCE)-sizedIST-1, 2,  
MPI_COMM_DIST);  
    }  
}  
  
MPI_Finalize ();  
exit (0);  
}
```

- Proceso de distribución



```
/******  
*                               Proceso carga.c                               *  
*                               *                                           *  
*   Proceso encargado de la informacion del sistema. Obtiene la *  
*   información local y la de los demas nodos *  
*   *  
*   Version: 1.0                               Fecha: 6 - 9 - 2000 *  
*   Autor: Diego Chaparro Gonzalez *  
*   *  
*****/  
  
#include "../include/bdd.h"  
  
#define WORKTAG          1  
#define DIETAG          2  
    // Periodo de tiempo para actualizar información sobre noods  
#define TIEMPOINF      10  
  
// Numero total de firmas que tiene el nodo actual para procesar  
long total_firmas;  
  
// Parámetros para el inicializar el MPI  
MPI_Comm MPI_COMM_CARGA;  
int sizeCARGA, rankCARGA, size, rank;  
  
// Para guardar en la tabla, los nodos y sus velocidades dinámicas  
struct reg {  
    int nodo;  
    float velocini; // velocidad inicial del nodo  
    float tareas; // N° de tareas medio  
    float porcentaje; // Porcentaje realizado por el nodo  
    time_t tultact; // Para saber cuando se actualizó por última vez  
    int tttotal; // Para saber cuanto tiempo ha transcurrido  
    long nfirmas; // Para saber nº firmas, puede cambiar  
};  
struct reg tabla[30];  
struct reg tablaaux[30];  
  
/******  
*                               FUNCION Numero_Tareas:                       *  
*   ----- *  
*   Manda la velocidad inicial local a los demás, y recibe la de los *  
*   otros. Después manda el número de tareas, y recibe la de los demás. *  
*   Después introduce los datos correctos en la tabla. *  
*   ----- *  
*****/  
int Numero_Tareas (void) {  
  
    FILE *fd;  
    char cadnum1[10], cadnum2[10], cadnum3[10], cadnum[10];  
    char *aux;  
    int numero;  
  
    fd = fopen ("/proc/loadavg", "r");  
  
    fscanf (fd, "%s %s %s %s", cadnum1, cadnum2, cadnum3, cadnum);  
  
    aux = strchr (cadnum, '/');  
    *aux='\0';  
  
    numero = atoi (cadnum);  
  
    fclose(fd);  
  
    return (numero);  
}  
  
/******  
*                               FUNCIÓN Mi_Velocidad:                       *  
*   ----- *  
*   Devuelve la velocidad inicial de cada nodo almacenada en el fichero veloc *  
*   ----- *  
*****/  
float Mi_Velocidad (void) {
```



```
FILE *fd;
char cadnum[10];
float numero;

fd = fopen ("/var/tmp/distribuidoDiego/exe/veloc", "r");

fscanf (fd, "%s", cadnum);

numero = atof (cadnum);

return (numero);
}

/*****
*          PROCEDIMIENTO Inicializar_Tabla:
* -----
*   Manda la velocidad inicial local a los demás, y recibe la de los
*   otros. Después manda el número de tareas, y recibe la de los demás.
*   Después introduce los datos correctos en la tabla.
*****/
void Inicializar_Tabla (time_t tult) {

    int env_ntareas, rec_ntareas;
    float env_veloc, rec_veloc;
    int i;
    MPI_Status status;

    // Mandar nº de tareas y velocidad inicial local a todos los nodos
    env_ntareas = Numero_Tareas();
    env_veloc = Mi_Velocidad();
    for (i=0; i<sizeCARGA; i++){
        if (i != rankCARGA ) {
            MPI_Send (&env_ntareas, 1, MPI_INT, i, 0, MPI_COMM_CARGA);
            MPI_Send (&env_veloc, 1, MPI_FLOAT, i, 1, MPI_COMM_CARGA);
        }
    }

    // Recibir nº de tareas y velocidad inicial de los demás
    for (i=0; i<sizeCARGA; i++) {
        if (i!= rankCARGA) {
            MPI_Recv (&rec_ntareas, 1, MPI_INT, i, 0, MPI_COMM_CARGA, &status);
            MPI_Recv (&rec_veloc, 1, MPI_FLOAT, i, 1, MPI_COMM_CARGA, &status);

            tabla[i].nodo = i;
            tabla[i].velocini = rec_veloc;
            tabla[i].tareas = rec_ntareas;
            tabla[i].porcentaje = 0;
            tabla[i].tultact = tult;
            tabla[i].ttotal = 1;
            tabla[i].nfirmas = total_firmas;
        }
    }

    // Introduzco en la tabla los valores del nodo local
    tabla[rankCARGA].nodo = rankCARGA;
    tabla[rankCARGA].velocini = env_veloc;
    tabla[rankCARGA].tareas = env_ntareas;
    tabla[rankCARGA].porcentaje = 0;
    tabla[rankCARGA].tultact = tult;
    tabla[rankCARGA].ttotal = 1;
    tabla[rankCARGA].nfirmas = total_firmas;
}

/*****
*          FUNCION Numero_Firmas:
* -----
*   Recibe como parámetro un fichero y devuelve el número de
*   líneas del mismo
*****/
```



```
long Numero_Firmas (char fichero[100]) {
    long num;
    char comando[200]= "wc -l >/var/tmp/distribuidoDiego/exe/numlin ";
    char cadaux[10];
    int cod;

    FILE *fd;

    strcat (comando, fichero);

    cod = system (comando);
    if (cod==127 || cod==-1)
        printf ("ERROR: Al ejecutar system");
    fd = fopen ("/var/tmp/distribuidoDiego/exe/numlin", "r");
    fscanf (fd, "%s", cadaux);

    num = atol (cadaux);
    fclose (fd);

    return (num);
}

/*****
 *                               FUNCIÓN Inicializar_MPI:                               *
 * -----*
 *   Inicializa MPI, crea el comunicador, halla el rango y el tamaño, ... *
 *****/
void Inicializar_MPI (int argc, char **argv) {

    int color;

    MPI_Init (&argc, &argv);

    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank (MPI_COMM_WORLD, &rank);

    // Creo el grupo para los demonios de carga
    color=2;
    MPI_Comm_split (MPI_COMM_WORLD, color, rank, &MPI_COMM_CARGA);
    MPI_Comm_rank (MPI_COMM_CARGA, &rankCARGA);
    MPI_Comm_size (MPI_COMM_CARGA, &sizeCARGA);
}

/*****
 *                               FUNCIÓN Enviar_Tareas:                               *
 * -----*
 *   Manda el número de tareas local a los demás nodos *
 *****/
void Enviar_Tareas () {
    int env_ntareas, i;

    // Mandar nº de tareas local
    env_ntareas = Numero_Tareas();
    for (i=0; i<sizeCARGA; i++){
        if (i != rankCARGA )
            MPI_Send (&env_ntareas, 1, MPI_INT, i, 0, MPI_COMM_CARGA);
    }
}

/*****
 *                               PROCEDIMIENTO Actualizar_Tabla;                               *
 * -----*
 *   Actualiza los datos de un nodo en la tabla *
 *****/
void Actualizar_Tabla (int nodo, int ntar, time_t tult) {
    int indice;
```



```
int dift;

indice = 0;
while (indice<sizeCARGA) {
    if (tabla[indice].nodo == nodo) {
        dift = difftime (tult, tabla[indice].tultact);
        tabla[indice].tareas = ((tabla[indice].tareas * tabla[indice].tttotal) +
(ntar*dift))/(tabla[indice].tttotal+dift);
        tabla[indice].tttotal = tabla[indice].tttotal + dift;
        tabla[indice].tultact = tult;
        tabla[indice].porcentaje = ((tabla[indice].velocini * tabla[indice].
tttotal / tabla[indice].nfirmas) * 100)/tabla[indice].tareas;
        indice = sizeCARGA;
    }
    else
        indice++;
}
}

/*****
 *          MOSTRAR TABLA
 *****/

void Mostrar_Tabla (void) {
    int i;

    printf ("%dCARGA: ***** TABLA *****\n", rankCARGA);
    printf ("NODO VELOCINI  TAREAS      PORC  TTOTAL  NFIRMAS\n");
    for (i=0; i<sizeCARGA; i++)
        printf ("%d %f %f %f %d      %ld\n", tabla[i].nodo, tabla[i].velocini,
tabla[i].tareas, tabla[i].porcentaje, tabla[i].tttotal, tabla[i].nfirmas);
}

/*****
 *          FUNCION Ordenar_Tabla
 * -----
 *  Ordena la tabla y devuelve el nodo asociado al actual
 *****/
void Ordenar_Tabla (void) {

    int orden;
    int i,j;
    struct reg aux;
    int salir, nuevonodo, nodotrans;
    MPI_Status status;
    int buffer;

    // Empleo el metodo de la burbuja
    for (i=1; i<(sizeCARGA-1); i++)
        for (j=0; j<(sizeCARGA-i); j++)
            if ((tabla[j].nfirmas-(tabla[j].porcentaje * tabla[j].nfirmas))>(tabla
[j+1].nfirmas - (tabla[j+1].porcentaje*tabla[j+1].nfirmas))){

                //Mostrar_Tabla();
                aux.nodo = tabla[j].nodo;
                aux.velocini = tabla[j].velocini;
                aux.tareas = tabla[j].tareas;
                aux.porcentaje = tabla[j].porcentaje;
                aux.tultact = tabla[j].tultact;
                aux.tttotal = tabla[j].tttotal;
                aux.nfirmas = tabla[j].nfirmas;

                tabla[j].nodo = tabla[j+1].nodo;
                tabla[j].velocini = tabla[j+1].velocini;
                tabla[j].tareas = tabla[j+1].tareas;
                tabla[j].porcentaje = tabla[j+1].porcentaje;
                tabla[j].tultact = tabla[j+1].tultact;
                tabla[j].tttotal = tabla[j+1].tttotal;
                tabla[j].nfirmas = tabla[j+1].nfirmas;

                tabla[j+1].nodo = aux.nodo;
```



```
        tabla[j+1].velocini = aux.velocini;
        tabla[j+1].tareas = aux.tareas;
        tabla[j+1].porcentaje = aux.porcentaje;
        tabla[j+1].tultact = aux.tultact;
        tabla[j+1].ttotal = aux.ttotal;
        tabla[j+1].nfirmas = aux.nfirmas;
    }

    for (i=0; i<sizeCARGA; i++)
        if (tabla[i].nodo == (rankCARGA))
            orden = i;

    // Obtengo el nodo simetrico al nodo actual
    nodotrans = tabla[sizeCARGA-1-orden].nodo;

    // Mando numero de nodo al proceso de distribucion
    MPI_Send (&nodotrans, 1, MPI_INT, rank-sizeCARGA, 4, MPI_COMM_WORLD);

    // Se siguen enviando numero de nodos mientras que el proceso de distribucion
    // siga pidiendolos
    nuevonodo = sizeCARGA-1-orden;
    orden = nuevonodo;
    salir = 0;
    while (!salir) {
        MPI_Recv (&buffer, 1, MPI_LONG, rank-sizeCARGA, 1, MPI_COMM_WORLD, &status);
        if (buffer != 0) {
            if (nuevonodo >= orden && nuevonodo < sizeCARGA)
                nuevonodo ++;
            else
                if (nuevonodo == sizeCARGA)
                    nuevonodo = orden-1;
                else
                    nuevonodo --;
            nodotrans = tabla[nuevonodo].nodo;
            MPI_Send (&nodotrans, 1, MPI_INT, rank-sizeCARGA, 4, MPI_COMM_WORLD);
        } else
            salir = 1;
    }
}

/*****
*          PROCEDIMIENTO Resto_NFirmas          *
* ----- *
*   Modifica el número de firmas total en la tabla de nodo actual, *
*   porque ha pasado carga a otro nodo *
*****/
long Resto_NFirmas (long firm){
    int i;
    long resul;

    i=0;
    while (i<sizeCARGA) {
        if (tabla[i].nodo == rankCARGA){
            tabla[i].nfirmas = tabla[i].nfirmas - firm;
            resul = tabla[i].nfirmas;
            i=sizeCARGA;
        } else
            i++;
    }
    total_firmas = total_firmas - firm;

    return (resul);
}

/*****
*          PROCEDIMIENTO Informar_NFirmas      *
* ----- *
*   El nodo actual informa a los demás de su nuevo n° de firmas, *
*   consecuencia de haber pasado carga a otro nodo *
*****/
void Informar_NFirmas (long firm) {

    int i;
```



```
        for (i=0; i<sizeCARGA; i++)
            if (i!=rankCARGA)
                MPI_Send (&firm, 1, MPI_LONG, i, 5, MPI_COMM_CARGA);
    }

/*****
 *          PROCEDIMIENTO Nuevo_NFirmas
 * -----
 *   Asigna en la tabla el nuevo nº de firmas del nodo actual,
 *   debido a la carga que le ha pasado otro nodo
 * *****/
void Nuevo_NFirmas (long firm) {
    int i;

    i=0;

    while (i<sizeCARGA){
        if (tabla[i].nodo == rankCARGA){
            tabla[i].nfirmas = firm;
            tabla[i].porcentaje = 0;
            tabla[i].ttotal = 1;
            i = sizeCARGA;
        } else
            i++;
    }
    total_firmas = firm;
}

/*****
 *          PROCEDIMIENTO Informar_Nuevo_NFirmas
 * -----
 *   Informa a los demás nodos de un nuevo número de firmas porque
 *   otro nodo le ha pasado carga
 * *****/
void Informar_Nuevo_NFirmas (long firm) {

    int i;

    for (i=0; i<sizeCARGA; i++)
        if (i!=rankCARGA)
            MPI_Send (&firm, 1, MPI_LONG, i, 6, MPI_COMM_CARGA);
}

/*****
 *          PROCEDIMIENTO Actualizar_NFirmas
 * -----
 *   Actualiza el nº firmas de un nodo que ha pasado carga a otro
 * *****/
void Actualizar_NFirmas (int nodo, long firm){

    int i;

    i=0;
    while (i<sizeCARGA)
        if (tabla[i].nodo==nodo) {
            tabla[i].nfirmas = tabla[i].nfirmas-firm;
            i = sizeCARGA;
        } else
            i++;
}

/*****
 *          PROCEDIMIENTO Actualizar_Nuevo_NFirmas
 * -----
 *   Actualiza el nuevo nº firmas de un nodo, ya que otro nodo le
 *   ha pasado carga
 * *****/
void Actualizar_Nuevo_NFirmas (int nodo, long firm){

    int i;
```



```
i=0;
while (i<sizeCARGA)
    if (tabla[i].nodo==nodo) {
        tabla[i].nfirmas = firm;
        tabla[i].ttotal = 1;
        tabla[i].porcentaje = 0;
        i = sizeCARGA;
    } else
        i++;
}

/*****
 *          PROCEDIMIENTO Informar_No_Hay_Nodos          *
 * -----
 * Informa a los demas demonios de carga que no hay nodos *
 * disponibles para realizar una transferencia de carga   *
 *****/
void Informar_No_Hay_Nodos (void){

    int i;
    long buffer;

    buffer = 0;
    for (i=0; i<sizeCARGA; i++)
        if (i!=rankCARGA)
            MPI_Send (&buffer, 1, MPI_LONG, i, 7, MPI_COMM_CARGA);
}

/*****
 *          PROGRAMA PRINCIPAL          *
 *****/

int main (argc, argv)
int argc;
char **argv;
{
    time_t tiempoini, tiempoult, tiempoact;
    int recibo, terminar;
    MPI_Status status, status2;
    long buffer;
    int error;

    Inicializar_MPI (argc, argv);

    // Calcular el tiempo inicial
    tiempoini = time (NULL);
    tiempoult = tiempoini;

    total_firmas = Numero_Firmas ("/var/tmp/distribuidoDiego/exe/firmas.en2");

    Inicializar_Tabla( time (NULL));

    // Permanece en este bucle, hasta que haya terminado el slave y no haya
    // nodos disponibles para realizar transferencias de carga
    terminar = 0;
    while (!terminar) {
        tiempoact = time (NULL);
        // Cuando pasa un periodo de tiempo TIEMPOINF actualizo informacion local
        if (tiempoact >tiempoult+TIEMPOINF) {
            Enviar_Tareas ();
            tiempoult = tiempoact;
            Actualizar_Tabla (rankCARGA, Numero_Tareas(), tiempoact);
        }
        else {
            // Compruebo si recibo mensajes de MPI_COMM_CARGA
            MPI_Iprobe (MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_CARGA, &recibo, &status);
            if (recibo) {
                MPI_Recv (&buffer, 1, MPI_LONG, status.MPI_SOURCE, status.MPI_TAG,
MPI_COMM_CARGA, &status2);
                switch (status2.MPI_TAG) {
```



```

                                case 0: // Información sobre tareas
                                Actualizar_Tabla (status2.MPI_SOURCE, buffer,
tiempoact);
                                break;

                                case 5: // otro carga informa de nº de firmas por paso a
otro
                                Actualizar_NFirmas (status2.MPI_SOURCE, buffer);
                                break;

                                case 6: // otro carga informa de nuevo nº firmas porque otro
nodo le ha pasado carga
                                Actualizar_Nuevo_NFirmas (status2.MPI_SOURCE,
buffer);
                                break;
                                case 7: // otro carga informa de que no hay nodos
disponibles

                                terminar = 1;
                                break;
                                default:
                                printf ("CARGA%d: RECIBO TAG INVALIDO", rankCARGA);
                                exit(1);
                                }
                                recibo = 0;
                                }
                                // Compruebo si recibo mensajes de MPI_COMM_WORLD
                                MPI_Iprobe (MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &recibo, &status);
                                if (!terminar && recibo) {
                                MPI_Recv (&buffer, 1, MPI_LONG, status.MPI_SOURCE, status.MPI_TAG,
MPI_COMM_WORLD, &status2);
                                switch (status2.MPI_TAG) {
                                case 1: // distrib ha terminado, y pide nº nodo para
conseguir carga
                                Ordenar_Tabla();
                                break;
                                case 2: // distrib informa nº firmas que ha pasado a otro
nodo
                                Informar_NFirmas (buffer);
                                buffer = Resto_NFirmas (buffer);
                                break;
                                case 3: // no hay nodos para transferir carga
                                Informar_No_Hay_Nodos();
                                terminar = 1;
                                break;
                                case 4: // distrib informa nº firmas que ha obtenido de otro
                                Nuevo_NFirmas (buffer);
                                Informar_Nuevo_NFirmas (buffer);
                                break;
                                default:
                                printf ("CARGA%d: RECIBO TAG INVALIDO", rankCARGA);
                                exit(1);
                                }
                                recibo = 0;
                                }
                                }
                                }

terminar = 0;

// El demonio de carga ha terminado su trabajo. Espera en el siguiente bucle
// hasta que recibe un mensaje del master para acabar
while (!terminar) {
    MPI_Iprobe (MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &recibo, &status);
    if (recibo) {
        MPI_Recv (&buffer, 1, MPI_LONG, status.MPI_SOURCE, status.MPI_TAG,
MPI_COMM_WORLD, &status2);
        // recibo petición de número de nodo para transferir. Mando -1 para
// indicar que no hay nodos disponibles
        if (status2.MPI_TAG==1){

```



```
        buffer = -1;
        MPI_Send (&buffer, 1, MPI_INT, status2.MPI_SOURCE, 4,
MPI_COMM_WORLD);
    }
    // Si recibo mensaje del master finalizo
    if (status2.MPI_SOURCE==0) terminar = 1;
}

recibo =0;
MPI_Iprobe (MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_CARGA, &recibo, &status);
if (recibo){
    MPI_Recv(&buffer, 1, MPI_LONG, status.MPI_SOURCE, status.MPI_TAG,
MPI_COMM_CARGA, &status2);
}

recibo = 0;

}

MPI_Finalize ();
exit (0);
}
```