Proyecto Nº1

Barco, Rodrigo -LU:99204
Otarola, Luca -LU:97886

Sistemas Distribuidos-UNS
29/09/2014

Consideraciones a tener en cuenta

En el informe se presentarán los problemas que se proponen, con las correspondientes respuestas a las preguntas realizadas, y las comparaciones entre los tiempos de ejecución correspondientes. Todos los programas ejecutarán en PC, Alpha y Sun,y finalmente se anexará el presente documento, en formato PDF.

Las mediciones son resultado del promedio de ejecución entre 5 y 10 muestras realizadas. Se utilizaron las funciones clock_gettime y gettimeofday. La arquitectura alpha no presenta precisión en nanosegundos, por eso para aquellos casos en que se piden ambas precisiones, hay dos soluciones separadas.

Finalmente, para la compilación se debe ejecutar el comando **make** dentro del directorio del programa, y en el caso de que haya un código exclusivo para otra arquitectura, se procederá a ejecutar el comando make con los siguientes parámetros:

make -f [NOMBRE_MAKEFILE]

Punto 1

Realizar un experimento en el cual se obtenga el tiempo que tarda en crear un proceso. ¿Cuántos procesos se pueden crear? Muestre los resultados obtenidos, medidos en microsegundos y en nanosegundos.

El programa presentado crea un hijo mediante el fork(), no realizando este ninguna tarea, y mide el tiempo antes y después de la creación. Por excepción, hay un programa que calcula en microsegundos y otro en nanosegundos.

A continuación se presentan los tiempos promedio de ejecución para cada arquitectura:

Arquitectura(So)	**PC(Fedora)**	**Alpha (RedHat)**	**Sun (Solaris)**
Tiempo(µseg)	138 µseg	235 µseg	120µseg

La cantidad de hijos que un proceso puede crear se averiguó a partir de la utilización de la función sysconf(_SC_CHILD_MAX), arrojando los siguientes resultados:

Arquitectura(So)	**PC(Fedora)**	**Alpha (RedHat)**	**Sun (Solaris)**
Procesos Hijo	30120	511	24300

Punto 2

Realizar un experimento para medir el tiempo que tarda en crear un proceso hijo que cargue una imagen ejecutable con una tarea. La tarea debe realizar un ciclo de 1000 iteraciones y en cada iteración dos operaciones de sumas, guardar un mensaje en el archivo denominado salida.txt y para finalizar guardar en el archivo el pid del proceso hijo. Los casos a considerar son los siguientes:

a) El proceso hijo es creado en la misma máquina.

b) El proceso hijo es creado en un máquina remota.

Compárelo con la utilización de system invocando la ejecución de la misma imagen ejecutable. Muestre los resultados obtenidos medidos en microsegundos y nanosegundos. Explique los resultados.

Para realizar una comparación entre el rendimiento de los programas, se tomaron muestras de las medidas en microsegundos y luego se calculó el promedio de las mismas.

Para el caso en que ambos procesos se encuentran en la misma máquina se obtuvieron los siguientes resultados:

Arquitectura (SO)	**PC (Fedora)**	**Alpha (RedHat)**	**Sun (Solaris)**
Fork + execv	2080 µseg	6931 µseg	544 µseg
system	5291 µseg	19335 µseg	332 µseg

Si tenemos en cuenta el caso en que los procesos se encuentran en máquinas distintas, se utilizó RPC. A partir de un .x, se generaron los stubs y se implementó la función deseada. Los resultados que obtuvimos son:

Servidor	**PC (Fedora)**	**Alpha (RedHat)**	**Sun (Solaris)**	**Alpha (Red Hat)**
Cliente	**PC (Fedora)**	**PC (Fedora)**	**PC (Fedora)**	**Sun (Solaris)**
Fork + execlv	2220 µseg	7475 µseg	1748 µseg	7212 µseg
system	5479 µseg	18976 µseg	3723 µseg	18713 µseg

Punto 3

Realizar los siguientes experimentos para estimar los tiempos de ejecución medidos en nanosegundos y en microsegundos.

a) Utilizando procedimientos que se ejecutan en forma local (LC - Local Call). Implemente los siguientes casos.

1) Una llamada a un procedimiento con 1 argumento de entrada de 4 bytes y 1 argumento de respuesta de 4 bytes.

2) Una llamada a un procedimiento con 2 argumentos de entrada de 1024 bytes y un argumento de respuesta de 2048 bytes.

Considere que el procedimiento realiza cuatro operaciones aritméticas y asigna valores a los argumentos de respuesta.

b) Utilizando procedimientos que se ejecutan en forma remota (RPC - Remote Procedure Call). Implemente los casos presentados en el inciso anterior.

c) Presente una tabla comparativa con los tiempos de ejecución de los experimentos realizados en los incisos anteriores. Algunos de los casos de prueba que debe considerar son los siguientes casos:

1) El procedimiento remoto se encuentra en la misma máquina.

2) El procedimiento remoto se encuentra en otra máquina con la misma arquitectura y distribución.

3) El procedimiento remoto se encuentra en otra máquina con diferente arquitectura y distribución

Se realizaron 2 procedimientos, los cueles arrojaron los siguientes tiempos de ejecución:

Arquitectura(SO)	**PC(Fedora)**	**Alpha(RedHat)**	**Sun(Solaris)**
1 argumento (µseg)	0,5	3,2	0,5
2 argumentos (µseg)	9	91	2,24

Servidor	**PC(Fedora)**	**Alpha(RedHat)**	**Alpha(redHat)**	**Sun(Solaris)**
Cliente	**PC(Fedora)**	**PC(Fedora)**	**Sun(Solaris)**	**Alpha(redHat)**
Soft (µseg)	100	2876	598	512
Hard (µseg)	107	3747	546	460

Es evidente un aumento en el tiempo necesario utilizando RPC, principalmente con computadoras que poseen la misma arquitectura.

Punto 4

Realizar un experimento para medir cuanto tiempo requiere un servidor implementado mediante sockets con TCP para atender la solicitud de un cliente. La medición del tiempo puede estar en microsegundo o en nanosegundos.

Cuánto requiere si el tamaño del dato que envía y recibe el cliente es de 4 bytes.

Cuánto requiere si el tamaño del dato que envía y recibe el cliente es de 2048 bytes.

¿Cuánto tiempo debe esperar el cliente?

Las funciones correspondientes a los sockets se separaron entre el cliente y el servidor (*Esperar conexión, Aceptar conexión, Enviar y Recibir datos*). Se miden dos tipos de tiempo, cuánto tarda el cliente en recibir la respuesta una vez enviada la petición y cuanto tarda el servidor en atender y mandar la respuesta.

Servidor	**PC(Fedora)**	**Alpha(RedHat)**	**Alpha(RedHat)**	**Sun(Solaris)**
Cliente	**PC(Fedora)**	**PC(Fedora)**	**Sun(Solaris)**	**Alpha(RedHat)**
4 Bytes	160 µseg	97 µseg	20 µseg	242 µseg
2048 Bytes	253 µseg	75 µseg	680 µseg	564 µseg

Punto 5

Realizar un servidor MiniFunción.
El cliente solicita el servicio que desea y luego invoca al procedimiento remoto para que se lo brinde.
El proceso servidor tiene las siguientes funciones: hora del sistema, fecha del sistema, nombre de la máquina, cantidad de usuarios logueados. Los experimentos deben tener las siguientes características:
a) Utilizar la llamada de un procedimiento que se encuentre en otro proceso en la misma máquina. Utilizando comunicación entre procesos (memoria compartida, archivos, colas de mensajes, etc.).
b) Utilizar llamadas a procedimientos remotos para las funciones requeridas (RPC).

Calcular los tiempos de respuesta medidos en nanosegundos o microsegundos para cada uno de los casos

anteriores. Presentar una tabla comparativa con los experimentos realizados utilizando todas las arquitecturas disponibles.

Para la comunicación de procesos se utilizó la memoria compartida. La sincronización se realiza mediante semáforos. Para comparar los tiempos se hicieron 5 muestras, arrojando los siguientes resultados:

Arquitectura (SO)	**PC (Fedora)**	**Alpha (RedHat)**	**Sun (Solaris)**
Hora			
Fecha			
Nombre			
Cantidad de usuarios			

Utilizando RPC se realizaron nuevamente 5 muestras, las cuales dieron como resultado:

Servidor	**PC (Fedora)**	**Alpha (RedHat)**	**Sun (Solaris)**
Cliente	**PC (Fedora)**	**PC (Fedora)**	**PC (Fedora)**
Hora			
Fecha			
Nombre			
Cantidad de usuarios			

Punto 6

Comunicación de pares:

Cada proceso par quiere comunicarse con otro proceso par para intercambiar mensajes o algún dato. En este caso, el intercambio de los datos es realizado a través de mensajes que se envían los pares.

Los datos que pueden intercambiar son los siguientes

- *Los usuarios logueados en el sistema*
- *La hora del sistema*
- *El nombre de su maquina*
- *la temperatura de la localidad donde está ubicado*

a) Realizar los procesos necesarios para cumplir las tareas especificas

b) Explique consideraciones de diseño que utilizaron para modelar el problema, detallando especialmente consideraciones para iniciar y finalizar una comunicación.

c) La comunicación entre los procesos se realiza mediante sockets con conexión.

Para la resolución del ejercicio, consideramos que el turno para hacer preguntas entre cada uno de los amigos es intercalado, es decir, el primer proceso hace una pregunta, luego de que el otro responde, tiene la posibilidad de hacer una pregunta. Al momento de ejecutar, se debe poner la ip y el puerto por el cual vamos a escuchar al otro proceso.

Una vez que ya están los amigos conectados, va a aparecer un menú al par que hace de cliente con las

preguntas que puede hacer. Una vez que el segundo amigo responde, a este último le aparecerá el mismo menú con las preguntas que puede realizarle al amigo 1. Es posible además finalizar la conversación, encargándose el programa de cerrar los correspondientes sockets.

Ejercicio 7

El proceso cliente quiere conocer la variación que tiene el reloj de su máquina con respecto a la máquina del proceso servidor. Inicialmente considere que sólo quiere conocer la variación con respecto a una máquina y luego con respecto a varias máquinas.

Realice los procesos necesarios para resolver este problema.

Se realizaron dos programas:

-El servidor, que obtiene su hora y la envía.

-El cliente, que obtiene su reloj, y lo compara con el del servidor. Es evidente que todo no puede realizarse al mismo tiempo, por lo que se obtiene una aproximación bastante acertada.

Ejercicio 8

Explique cómo haría para que el/los relojes de los procesos clientes queden sincronizados con el reloj del proceso servidor

La idea detrás de este proceso es la de que los procesos clientes requieran periódicamente al servidor .

El reloj no se puede ajustar directamente, ya que atrasar o adelantar un reloj podría generar una modificación de las relaciones de causalidad o falsear medida de intervalos. La solución es cambiar la frecuencia del contador, mediante el uso de la siguiente primitiva de Linux:

int adjtimex(struct timex *buf);

```
struct timex {
    int modes; /* valores a modificar */
    long offset; /* ajuste de hora (microseg) */
    long freq; /* ajuste de frecuencia (ppm escalada) */
    long maxerror; /* error maximo (microsegundos) */
    long esterror; /* error estimado (microsegundos) */
    int status; /* orden/estado del reloj */
    long constant; /* constante de tiempo del pll */
    long precision; /* precisión del reloj (microseg) (lectura) */
    long tolerance; /* tolerancia de la frecuencia del reloj (ppm) (lectura) */
    struct timeval time; /* hora actual (lectura) */
    long tick; /* microsegundos entre ticks del reloj */
};
```

Valores a modificar:

```
#define ADJ OFFSET 0x0001 /* ajuste de la hora */
#define ADJ FREQUENCY 0x0002 /* ajuste de la frecuencia */
#define ADJ MAXERROR 0x0004 /* error maximo de la hora*/
#define ADJ ESTERROR 0x0008 /* error estimado de la hora */
#define ADJ STATUS 0x0010 /* estado del reloj */
#define ADJ TIMECONST 0x0020 /* constante de tiempo del pll */
#define ADJ TICK 0x4000 /* valor tick*/
#define ADJ OFFSET SINGLESHOT 0x8001 /* ajuste inmediato de hora */
```

Ejercicio 12

En este ejercicio, usted deberá elegir entre el problema 5 y 6 e implementarlo en Java con sockets con conexión. Estime el tiempo de ejecución y compare con los resultados obtenidos.

Se realizó el punto 5 en Java, obteniendo como resultado una mayor velocidad que en el punto 5, presumiblemente por el retardo que conlleva la ejecución en una máquina virtual.

Servidor	**PC (Fedora)**	**Sun (Solaris)**
Cliente	**PC (Fedora)**	**PC (Fedora)**
Hora(us)	22343	9385
Fecha(us)	26154	4817
Nombre(us)	22677	5785
Cantidad de usuarios(us)	5238	198892

El formato para ejecutar es:

Servidor:

java SocketServidor puerto

Cliente:

java SocketCliente ip puerto

Código fuente

Punto 1

ejercicio_01.c

Comentario [1]: <!--StartFragment-->

Comentario [2]: <!--EndFragment-->

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


int main() {

	struct timespec ini, fin;

	double nano;
	int pid, err;


	// Tiempo de Inicio
	clock_gettime( CLOCK_REALTIME, &ini);   // inicio
	// Nuevo proceso
	pid = fork();

	// proceso padre...
	if (pid != 0) {
		// Tiempo final
		 clock_gettime( CLOCK_REALTIME, &fin) ;    // fin
		// Lo transformo a nSegundos
		nano =( fin.tv_sec - ini.tv_sec )*1000000000+ (double)( fin.tv_nsec - ini.tv_nsec );
	} else {
	//proceso hijo...
		exit(0);
	};

	printf("Tiempo para crear un nuevo hijo: %g nSegs | %g uSegs\n",nano,nano/1000);
	printf ("Cantidad de procesos que puede crear:%lu \n",sysconf(_SC_CHILD_MAX));
	return 0;
}
```

ejercicio_01_alpha.c

Comentario [3]: <!--StartFragment-->

Comentario [4]: <!--EndFragment-->

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

int main() {

	struct timeval Tinicio, Tfin;

	long duracion;
	int pid, err;


	// Tiempo de Inicio
	gettimeofday(&Tinicio,0);
	// Nuevo proceso
	pid = fork();

	// proceso padre...
	if (pid != 0) {
		// Tiempo final
		gettimeofday(&Tfin,0);
		// Lo transformo a uSegundos
		duracion= ((Tfin.tv_sec * 1000000) + Tfin.tv_usec) - ((Tinicio.tv_sec * 1000000) +
Tinicio.tv_usec);
	} else {
	//proceso hijo...
		exit(0);
	};

	printf("Tiempo para crear un nuevo hijo: %d uSegs \n",duracion);
	printf ("Cantidad de procesos que puede crear:%lu \n",sysconf(_SC_CHILD_MAX));
	return 0;
}
```

Punto 2A

ejercicio2.c

Comentario [5]: <!--StartFragment-->

Comentario [6]: <!--EndFragment-->

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

int main(int argc, char *argv[])
{
	double tiemponano;
	struct timespec ti_nano, tf_nano;
	int pid;
	clock_gettime(CLOCK_REALTIME, &ti_nano);  // Instante inicial
```

```
	pid = fork();

	if (pid != 0)
	{
		wait(0);
		clock_gettime(CLOCK_REALTIME, &tf_nano);  // Instante final
		tiemponano= (tf_nano.tv_sec - ti_nano.tv_sec)*1000000000 + (tf_nano.tv_nsec - ti_nano.tv_nsec);
		printf("crear un hijo y ejecutar la tarea tardo:%g us | %g ns \n", tiemponano/1000,tiemponano);

		//realizamos lo mismo pero con la llamada system()
		clock_gettime(CLOCK_REALTIME, &ti_nano);  // Instante inicial
		system("./salida");
		clock_gettime(CLOCK_REALTIME, &tf_nano);  // Instante final
		tiemponano= (tf_nano.tv_sec - ti_nano.tv_sec)*1000000000 + (tf_nano.tv_nsec - ti_nano.tv_nsec);
		printf("Realizar la misma tarea pero ejecutada con la llamada system sin crear otro hijo tardo:%g us | %g ns \n", tiemponano/1000,tiemponano);

	}
	else
	{
		char*args[]={"salida",argv[1], (char*) NULL};
		execv("salida", args);
		printf("Return not expected. Must be an execv error \n");
	}
	return 0;
}
```

ejercicio2_alpha.c

Comentario [7]: <!--StartFragment-->

Comentario [8]: <!--EndFragment-->

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

int main()
{
	struct timeval timeInicio, timeFin;
	long total, usegInicio, usegFin;
	int pid;

	total = 0;

	gettimeofday(&timeInicio,0);
```

```
	pid = fork();

	if (pid != 0)
	{
		wait(0);
		gettimeofday(&timeFin,0);// Instante final
		usegInicio = ((timeInicio.tv_sec * 1000000) + timeInicio.tv_usec);
		usegFin = ((timeFin.tv_sec * 1000000) + timeFin.tv_usec);
		total =usegFin - usegInicio;
		printf("crear un hijo y ejecutar la tarea tardo:%ld us \n", total);

		//realizamos lo mismo pero con la llamada system()
		gettimeofday(&timeInicio,0);
		system("./salida");
		gettimeofday(&timeFin,0);// Instante final
		usegInicio = ((timeInicio.tv_sec * 1000000) + timeInicio.tv_usec);
		usegFin = ((timeFin.tv_sec * 1000000) + timeFin.tv_usec);
		total =usegFin - usegInicio;
		printf("Realizar la misma tarea pero ejecutada con la llamada system sin crear otro hijo tardo:%ld us  \n", total);

	}
	else
	{
		execvp("./salida", NULL);
		printf("Return not expected. Must be an execv error \n");
	}
	return 0
}
```

salida.c

Comentario [9]: <!--StartFragment-->

```
#include <sys/time.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
```

```
int main(int argc, char *argv[]){
  int i;
  FILE *archivo;


  archivo = fopen ("salida.txt", "w");
  i=0;

  while(i<1000)
  {
    double aux=2.5*4.3;
    fprintf(archivo, "Soy el proceso hijo \n");
    i++;
  }

  fprintf(archivo, "Mi PID es %i \n",getpid());
  fclose (archivo);
  return 0;
}
```

Comentario [10]: <!--EndFragment-->

Punto 2B

ej2b.x

Comentario [11]: <!--StartFragment-->

```
program ej2b {
        version ej2bv1 {

                int ejecutartarea(int) = 1;

        } = 1;
} = 0x20000001;
```

Comentario [12]: <!--EndFragment-->

cliente.c

Comentario [13]: <!--StartFragment-->

```
#include "ej2b.h"
#include <stdlib.h>
#include <stdio.h>
#include <sys/time.h>
```

```
#include <time.h>
#include <limits.h>
#include <unistd.h>
#include <sys/resource.h>

//time1 es para el tiempo con el metodo de creacion del proceso hijo
//time2 es para el tiempo con el metodo del system
double time1_nano, time2_nano;
struct rlimit rlim;
struct timespec ti_nano, tf_nano;
clockid_t id = CLOCK_REALTIME;
CLIENT *clnt;
int opcion;

CLIENT* establecerConexion(char *servername);

void cerrarConexion();

double calcularTiempo_nano();

void realizarFork();

void realizarSystem();

int main(int argc, char *argv[])
{
	printf("Estableciendo conexion...\n");
	clnt = establecerConexion(argv[1]);

	if(clnt != NULL)
		printf("Conexion establecida\n");
	else
	{
		printf("No se ha podido establecer conexion!\n");
		exit(1);
	}
```

```
	realizarFork();
	realizarSystem();



	printf("Se tardo en crear un proceso en una maquina remota: %g microsegundos || %u nanosegundos\n", time1_nano/1000, (uint) time1_nano);
	printf("Se tardo en hacer una llamada system en una maquina remota: %g microsegundos || %u nanosegundos\n", time2_nano/1000, (uint) time2_nano);


	printf("Cerrando conexion...\n");
	cerrarConexion();


	return 0;
}


CLIENT* establecerConexion(char *servername)
{
	return clnt_create(servername, ej2b, ej2bv1, "tcp");
}


void cerrarConexion()
{
	clnt_destroy(clnt);
}


double calcularTiempo_nano()
{
	return (   (tf_nano.tv_sec - ti_nano.tv_sec)*1000000000 + (tf_nano.tv_nsec - ti_nano.tv_nsec)   );
}


void realizarFork()
{
	//1 = fork + cargar imagen
	//!=1 = system
	opcion = 1;
```

```
	clock_gettime( id, &ti_nano);

	ejecutartarea_1( &opcion, clnt);

	clock_gettime( id, &tf_nano);

	time1_nano = calcularTiempo_nano();
}

void realizarSystem()
{
	//1 = fork + cargar imagen
	//!=1 = system
	opcion = 2;
	clock_gettime( id, &ti_nano);

	ejecutartarea_1( &opcion, clnt);

	clock_gettime( id, &tf_nano);

	time2_nano = calcularTiempo_nano();   }
```

server.c

```
#include "ej2b.h"
#include <stdlib.h>
#include <stdio.h>
#include <sys/time.h>
#include <time.h>
#include <limits.h>
#include <unistd.h>
#include <sys/resource.h>

int* ejecutartarea_1_svc(int* opcion,struct svc_req* req)
{
	static int toReturn = 0;
	int op = *opcion;
	if(op == 1) //creo proceso hijo y le cargo la imagen.
```

```
        {
                int p_hijo = fork();

                if(p_hijo >= 0)
                        if(p_hijo == 0) //hijo
                                execlp("tarea",NULL);//cargamos la tarea en el proceso hijo
                        else
                                wait(0); //esperando al hijo
                else
                {
                        toReturn = -1; //ERROR al crear el proceso hijo
                        return &(toReturn);
                }
        }
        else //realizo la tarea con system.
          system("./tarea");
        return  &toReturn;
}
```

tarea.c

Comentario [14]: <!--StartFragment-->

Comentario [15]: <!--EndFragment-->

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main()
{
        int i;
        double aux;
        FILE *file;
   file = fopen ("salida.txt", "w" );

        for( i = 0; i < 1000; i++)
        {
                 aux=2.5*4.3;
                fprintf(file, "Soy el proceso hijo con pid:%i\n",getpid());
        }
```

```
        fprintf(file, "Finalizo el ciclo, mi pid = %i \n", getpid());
        fclose(file);

        return 0;
}
```

Punto3A1

localcall.c

Comentario [16]: <!--StartFragment-->

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

void LocalCall (unsigned long entrada,unsigned long *salida);


int main (){
  //declaracion de variables
  unsigned long argumento;
  unsigned long* respuesta;
  double tiemponano;
  struct timespec ti_nano, tf_nano;
  argumento=rand();
  printf ("Se genero un argumento de tipo LONG (4 bytes) que tien el valor %lu\n",argumento);
  // ejecuto la llamada y calculo el tiempo
  clock_gettime(CLOCK_REALTIME, &ti_nano);  // Instante inicial
  LocalCall(argumento,respuesta);
  clock_gettime(CLOCK_REALTIME, &tf_nano);  // Instante final
  tiemponano= (tf_nano.tv_sec - ti_nano.tv_sec)*1000000000 + (tf_nano.tv_nsec - ti_nano.tv_nsec);
  //muestro los resultados
  printf ("Local Call dio como resultado en la salida: %lu\n",*respuesta);
  printf ("La llamada a LocalCall demoro %g us | %g ns\n",tiemponano/1000,tiemponano);
  return (0);
}
```

```
void LocalCall (unsigned long entrada,unsigned long *salida){
  *salida=entrada/8*2+9-6;
  printf ("Soy la llamada local y realice la siguiente operacion aritmetica, (entrada/8*2+6-9), que fue asignada a la salida\n");
}
```

Comentario [17]: <!--EndFragment-->

Comentario [18]: <!--StartFragment-->

localcall_Alpha.c

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

void LocalCall (unsigned long entrada,unsigned long *salida);


int main (){
  //declaracion de variables
  unsigned long argumento;
  unsigned long* respuesta;
  double tiempo;
  struct timeval ti_usec, tf_usec;
  //genero un numero aleatorio
  argumento=rand();
  printf ("Se genero un argumento de tipo LONG (4 bytes) que tien el valor %lu\n",argumento);
  // ejecuto la llamada y calculo el tiempo
  gettimeofday(&ti_usec,0);  // Instante inicial
  LocalCall(argumento,respuesta);
  gettimeofday(&tf_usec,0);  // Instante final
  tiempo= (tf_usec.tv_sec - ti_usec.tv_sec)*1000000 + (tf_usec.tv_usec - ti_usec.tv_usec);
  //muestro los resultados
  printf ("Local Call dio como resultado en la salida: %lu\n",*respuesta);
  printf ("La llamada a LocalCall demoro %g us\n",tiempo);
  return (0);
}
```

```
void LocalCall (unsigned long entrada,unsigned long *salida){
  *salida=entrada/8*2+9-6;
  printf ("Soy la llamada local y realice la siguiente operacion aritmetica, (entrada/8*2+6-9), que fue asignada a la salida\n");
}
```

Comentario [19]: <!--EndFragment-->

Punto3A2

localcall.c

Comentario [20]: <!--StartFragment-->

Comentario [21]: <!--EndFragment-->

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

void LocalCall (unsigned long *entrada1,unsigned long *entrada2,unsigned long *salida);


int main (){
  //declaracion de variables
  unsigned long argumento1  [256],argumento2  [256];
  unsigned long respuesta[512];
  double tiemponano;
  struct timespec ti_nano, tf_nano;
  int i;
  for (i=0;i<256;i++){
    *(argumento1+i)=rand();
    *(argumento2+i)=rand();
  }
  printf ("Se generaron 2 argumentos con 256 datos de tipo LONG (4 bytes) como entrada a la llamada local\n");
  // ejecuto la llamada y calculo el tiempo
  clock_gettime(CLOCK_REALTIME, &ti_nano);  // Instante inicial
  LocalCall(argumento1,argumento2,respuesta);
  clock_gettime(CLOCK_REALTIME, &tf_nano);  // Instante final
  tiemponano= (tf_nano.tv_sec - ti_nano.tv_sec)*1000000000 + (tf_nano.tv_nsec - ti_nano.tv_nsec);
```

```
 //muestro los resultados
 printf ("La llamada a LocalCall demoro %g us | %g ns\n",tiemponano/1000,tiemponano);
 return (0);
}

void LocalCall (unsigned long *entrada1,unsigned long *entrada2,unsigned long *salida){
 int i=0;
 int j=0;
 while (i<256){
  *(salida+j)=*(entrada1+i)/10*8+9-6;
  j++;
  *(salida+j)=*(entrada2+i)/10*8+9-6;
  j++;
  i++;
 }
}
```

localcall_Alpha.c

Comentario [22]: <!--StartFragment-->

Comentario [23]: <!--EndFragment-->

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

void LocalCall (unsigned long *entrada1,unsigned long *entrada2,unsigned long *salida);


int main (){
 //declaracion de variables
 unsigned long argumento1  [256],argumento2  [256];
 unsigned long respuesta[512];
 double tiempo;
 struct timeval ti_usec, tf_usec;
 int i;
 for (i=0;i<256;i++){
  *(argumento1+i)=rand();
```

```
   *(argumento2+i)=rand();
  }
  printf ("Se generaron 2 argumentos con 256 datos de tipo LONG (4 bytes) como entrada a la llamada local\n");
  // ejecuto la llamada y calculo el tiempo
  gettimeofday(&ti_usec,0);  // Instante inicial
  LocalCall(argumento1,argumento2,respuesta);
  gettimeofday(&tf_usec,0);  // Instante final
  tiempo= (tf_usec.tv_sec - ti_usec.tv_sec)*1000000 + (tf_usec.tv_usec - ti_usec.tv_usec);
  //muestro los resultados
  printf ("La llamada a LocalCall demoro %g us\n",tiempo);
  return (0);
}

void LocalCall (unsigned long *entrada1,unsigned long *entrada2,unsigned long *salida){
  int i=0;
  int j=0;
  while (i<256){
    *(salida+j)=*(entrada1+i)/10*8+9-6;
    j++;
    *(salida+j)=*(entrada2+i)/10*8+9-6;
    j++;
    i++;
  }
}
```

Punto3B1-Alpha

ej3b1.x

Comentario [24]: <!--StartFragment-->

Comentario [25]: <!--EndFragment-->

```
program ej3b1 {
        version ej3b1v1 {
                unsigned long ejecutartarea(unsigned long) = 1;
        } = 1;
} = 0x20000001
```

ej3b1_client.c

Comentario [26]: <!--StartFragment-->

```
/*
```

```
 * This is sample code generated by rpcgen.
 * These are only templates and you can use them
 * as a guideline for developing your own functions.
 */

#include "ej3b1.h"
#include <stdlib.h>
#include <stdio.h>
#include <sys/time.h>
#include <time.h>
#include <limits.h>
#include <unistd.h>
#include <sys/resource.h>

void
ej3b1_1(char *host)
{
	CLIENT *clnt;
	u_long  *result_1;
	u_long  ejecutartarea_1_arg;
	double time;
	struct timeval Tinicio, Tfin;
	clockid_t id = CLOCK_REALTIME;

#ifndef DEBUG
	clnt = clnt_create (host, ej3b1, ej3b1v1, "udp");
	if (clnt == NULL) {
		clnt_pcreateerror (host);
		exit (1);
	}
#endif	/* DEBUG */
	ejecutartarea_1_arg=rand();
	printf ("Se genero un argumento de tipo LONG (4 bytes) que tien el valor %lu\n",ejecutartarea_1_arg);
	//realizo la llamada al server
	gettimeofday(&Tinicio,0); //tiempo Inicio
	result_1 = ejecutartarea_1(&ejecutartarea_1_arg, clnt);
```

```
	gettimeofday(&Tfin,0); //tiempo final
	//calculo el tiempo de ejecucion
	time=((Tfin.tv_sec * 1000000) + Tfin.tv_usec) - ((Tinicio.tv_sec * 1000000) + Tinicio.tv_usec);
	//muestro los resultados
	printf("El valor de la salida es:%lu\n",*result_1);
	printf("Se tardo en realizar la tarea en una maquina remota: %g microsegundos \n", time);
	if (result_1 == (u_long *) NULL) {
		clnt_perror (clnt, "call failed");
	}
#ifndef DEBUG
	printf("Cerrando conexion...\n");
	clnt_destroy (clnt);
#endif	 /* DEBUG */
}


int
main (int argc, char *argv[])
{
	char *host;

	if (argc < 2) {
		printf ("usage: %s server_host\n", argv[0]);
		exit (1);
	}
	host = argv[1];
	ej3b1_1 (host);
exit (0);
}
```

Comentario [27]: <!--EndFragment-->

ej3b1_server.c

Comentario [28]: <!--StartFragment-->

```
/*
 * This is sample code generated by rpcgen.
 * These are only templates and you can use them
 * as a guideline for developing your own functions.
 */
```

```
#include "ej3b1.h"
#include <stdlib.h>
#include <stdio.h>
u_long *
ejecutartarea_1_svc(u_long *argp, struct svc_req *rqstp)
{
        static u_long  result;

        result=*argp/8*2+9-6;
        printf ("Soy el servidor y realice la siguiente operacion aritmetica, entrada/8*2+6-9), que fue asignada a la salida\n");
        return &result;
}
```

Comentario [29]: <!--EndFragment-->

Punto3B1-Sun y PC

ej3b1.x

Comentario [30]: <!--StartFragment-->

```
program ej3b1 {
        version ej3b1v1 {
                unsigned long ejecutartarea(unsigned long) = 1;
        } = 1;
} = 0x20000001;
```

Comentario [31]: <!--EndFragment-->

ej3b1_client.c

Comentario [32]: <!--StartFragment-->

```
/*
 * This is sample code generated by rpcgen.
 * These are only templates and you can use them
 * as a guideline for developing your own functions.
 */

#include "ej3b1.h"
#include <stdlib.h>
#include <stdio.h>
#include <sys/time.h>
```

```
#include <time.h>
#include <limits.h>
#include <unistd.h>
#include <sys/resource.h>

void
ej3b1_1(char *host)
{
	CLIENT *clnt;
	u_long  *result_1;
	u_long  ejecutartarea_1_arg;
	double time_nano;
	struct timespec ti_nano, tf_nano;
	clockid_t id = CLOCK_REALTIME;

#ifndef DEBUG
	clnt = clnt_create (host, ej3b1, ej3b1v1, "udp");
	if (clnt == NULL) {
		clnt_pcreateerror (host);
		exit (1);
	}
#endif  /* DEBUG */
	//genero un numero aleatorio
	srand(time(NULL));
	ejecutartarea_1_arg=rand();
	printf ("Se genero un argumento de tipo LONG (4 bytes) que tien el valor %lu\n",ejecutartarea_1_arg);
	//realizo la llamada al server
	clock_gettime( id, &ti_nano); //tiempo de inicio
	result_1 = ejecutartarea_1(&ejecutartarea_1_arg, clnt);
	clock_gettime( id, &tf_nano); //tiempo final
	//calculo el tiempo de ejecucion
	time_nano=(tf_nano.tv_sec - ti_nano.tv_sec)*1000000000 + (tf_nano.tv_nsec - ti_nano.tv_nsec);
	//muestro los resultados
	printf("El valor de la salida es:%lu\n",*result_1);
	printf("Se tardo en realizar la tarea en una maquina remota: %g microsegundos || %u nanosegundos\n",
time_nano/1000, (uint) time_nano);
```

```
	if (result_1 == (u_long *) NULL) {
		clnt_perror (clnt, "call failed");
	}
#ifndef DEBUG
	printf("Cerrando conexion...\n");
	clnt_destroy (clnt);
#endif	 /* DEBUG */
}


int
main (int argc, char *argv[])
{
	char *host;

	if (argc < 2) {
		printf ("usage: %s server_host\n", argv[0]);
		exit (1);
	}
	host = argv[1];
	ej3b1_1 (host);
exit (0);
}
```

Comentario [33]: <!--EndFragment-->

ej3b1_server.c

Comentario [34]: <!--StartFragment-->

Comentario [35]: <!--EndFragment-->

```
/*
 * This is sample code generated by rpcgen.
 * These are only templates and you can use them
 * as a guideline for developing your own functions.
 */

#include "ej3b1.h"
#include <stdlib.h>
#include <stdio.h>
u_long *
ejecutartarea_1_svc(u_long *argp, struct svc_req *rqstp)
```

```
{
        static u_long  result;

        result=*argp/8*2+9-6;
        printf ("Soy el servidor y realice la siguiente operacion aritmetica, entrada/8*2+6-9), que fue asignada a la salida\n");
        return &result;
}
```

Punto3B2-Alpha

ej3b2.x

Comentario [36]: <!--EndFragment-->

Comentario [37]: <!--StartFragment-->

```
struct entrada{
  unsigned long datos1[256];
  unsigned long datos2[256];
};

struct salida {
  unsigned long datos[512];
};

program ej3b1 {
        version ej3b1v1 {
                salida ejecutartarea(entrada) = 1;
        } = 1;
} = 0x20000001;
```

ej3b2_client.c

Comentario [38]: <!--StartFragment-->

```
/*
 * This is sample code generated by rpcgen.
 * These are only templates and you can use them
 * as a guideline for developing your own functions.
 */

#include "ej3b2.h"
#include <unistd.h>
```

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

void
ej3b1_1(char *host)
{
        CLIENT *clnt;
        salida  *result_1;
        entrada  ejecutartarea_1_arg;
        double tiempo;
        struct timeval Tinicio, Tfin;
        char aux;
        int i;
#ifndef DEBUG
        clnt = clnt_create (host, ej3b1, ej3b1v1, "udp");
        if (clnt == NULL) {
                clnt_pcreateerror (host);
                exit (1);
        }
#endif  /* DEBUG */



        for (i=0;i<256;i++){
          *(ejecutartarea_1_arg.datos1+i)=rand();
          *(ejecutartarea_1_arg.datos2+i)=rand();
        }
        printf ("Se generaron 2 argumentos con 256 datos de tipo LONG (4 bytes) como entrada a la llamada local\n");
        //ejecuto la llamada al servidor
        // Tiempo de Inicio
        gettimeofday(&Tinicio,0);
        result_1 = ejecutartarea_1(&ejecutartarea_1_arg, clnt);
        // Tiempo final
        gettimeofday(&Tfin,0);
```

```
        if (result_1 == (salida *) NULL) {
                clnt_perror (clnt, "call failed");
        }
        //calculo el tiempo de ejecucion en el servidor
        tiempo= ((Tfin.tv_sec * 1000000) + Tfin.tv_usec) - ((Tinicio.tv_sec * 1000000) + Tinicio.tv_usec);
        //muestro los resultados
        printf ("La llamada a LocalCall demoro %g us \n",tiempo);
        printf ("desea ver el resultado de la salida s/n \n");
        aux=getchar();
        if (aux=='s'){
        for (i=0;i<512;i++)
          printf("salida[%i]:%lu\n",i,(result_1->datos[i]));
        }

#ifndef DEBUG
        clnt_destroy (clnt);
#endif    /* DEBUG */
}


int
main (int argc, char *argv[])
{
        char *host;

        if (argc < 2) {
                printf ("usage: %s server_host\n", argv[0]);
                exit (1);
        }
        host = argv[1];
        ej3b1_1 (host);
exit (0);
}
```

ej3b2_server.c

Comentario [39]: <!--EndFragment-->

Comentario [40]: <!--StartFragment-->

Comentario [41]: <!--EndFragment-->

```
/*
 * This is sample code generated by rpcgen.
 * These are only templates and you can use them
 * as a guideline for developing your own functions.
 */

#include "ej3b2.h"
#include <stdlib.h>
#include <stdio.h>
salida *
ejecutartarea_1_svc(entrada *argp, struct svc_req *rqstp)
{
        static salida  result;

        int i=0;
        int j=0;
        while (i<256){
          *(result.datos+j)=*(argp->datos1+i)/10*8+9-6;
          j++;
          *(result.datos+j)=*(argp->datos1+i)/10*8+9-6;
          j++;
          i++;
        }

        return &result;
}
```

Punto3B2-Sun y Pc

ej3b2.x

Comentario [42]: <!--StartFragment-->

Comentario [43]: <!--EndFragment-->

```
struct entrada{
  unsigned long datos1[256];
  unsigned long datos2[256];
};

struct salida {
```

```
  unsigned long datos[512];
};

program ej3b1 {
        version ej3b1v1 {
                salida ejecutartarea(entrada) = 1;
        } = 1;
} = 0x20000001;
```

ej3b2_client.c

Comentario [44]: <!--StartFragment-->

Comentario [45]: <!--EndFragment-->

```
/*
 * This is sample code generated by rpcgen.
 * These are only templates and you can use them
 * as a guideline for developing your own functions.
 */

#include "ej3b2.h"
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/time.h>
#include <time.h>

void
ej3b1_1(char *host)
{
        CLIENT *clnt;
        salida  *result_1;
        entrada  ejecutartarea_1_arg;
        double tiemponano;
        struct timespec ti_nano, tf_nano;
        char aux;
        int i;
#ifndef DEBUG
        clnt = clnt_create (host, ej3b1, ej3b1v1, "udp");
        if (clnt == NULL) {
```

```
                clnt_pcreateerror (host);
                exit (1);
        }
#endif  /* DEBUG */

        //genero un numero aleatorio
        srand(time(NULL));

        for (i=0;i<256;i++){
          *(ejecutartarea_1_arg.datos1+i)=rand();
          *(ejecutartarea_1_arg.datos2+i)=rand();
        }
        printf ("Se generaron 2 argumentos con 256 datos de tipo LONG (4 bytes) como entrada a la llamada local\n");
        //ejecuto la llamada al servidor
        clock_gettime(CLOCK_REALTIME, &ti_nano);  // Instante inicial
        result_1 = ejecutartarea_1(&ejecutartarea_1_arg, clnt);
        clock_gettime(CLOCK_REALTIME, &tf_nano);  // Instante final
        if (result_1 == (salida *) NULL) {
                clnt_perror (clnt, "call failed");
        }
        //calculo el tiempo de ejecucion en el servidor
        tiemponano= (tf_nano.tv_sec - ti_nano.tv_sec)*1000000000 + (tf_nano.tv_nsec - ti_nano.tv_nsec);
        //muestro los resultados
        printf ("La llamada a LocalCall demoro %g us | %g ns\n",tiemponano/1000,tiemponano);
        printf ("desea ver el resultado de la salida s/n \n");
        aux=getchar();
        if (aux=='s'){
        for (i=0;i<512;i++)
          printf("salida[%i]:%lu\n",i,(result_1->datos[i]));
        }

#ifndef DEBUG
        clnt_destroy (clnt);
#endif   /* DEBUG */
}
```

```
int
main (int argc, char *argv[])
{
        char *host;

        if (argc < 2) {
                printf ("usage: %s server_host\n", argv[0]);
                exit (1);
        }
        host = argv[1];
        ej3b1_1 (host);
exit (0);
}
```

ej3b2_server.c

Comentario [46]: <!--StartFragment-->

```
/*
 * This is sample code generated by rpcgen.
 * These are only templates and you can use them
 * as a guideline for developing your own functions.
 */

#include "ej3b2.h"
#include <stdlib.h>
#include <stdio.h>
salida *
ejecutartarea_1_svc(entrada *argp, struct svc_req *rqstp)
{
        static salida  result;

        int i=0;
        int j=0;
        while (i<256){
          *(result.datos+j)=*(argp->datos1+i)/10*8+9-6;
          j++;
```

```
  *(result.datos+j)=*(argp->datos1+i)/10*8+9-6;
  j++;
  i++;
 }

 return &result;
}
```

Comentario [47]: <!--EndFragment-->

Punto4

cliente.c

Comentario [48]: <!--StartFragment-->

```
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/un.h>


#define BACKLOG 10 /* Tamaño de la cola de conexiones recibidas */
#define PORT 13490     /* Puerto del servidor */
#define MAXDATASIZESHORT 4
#define MAXDATASIZESOLIC 32
#define MAXDATASIZELARGE 2048

 double tiempo;
struct timeval ti, tf;    //Structuras para medir el tiempo


int socketfd, numbytes;


char buf0[MAXDATASIZESOLIC], buf1[MAXDATASIZELARGE], buf2[MAXDATASIZESHORT]; /*
Buffers donde se reciben y mandan los datos */
```

```
struct hostent * he; /* Se utiliza para convertir el nombre del host a su dirección IP */

struct sockaddr_in serv_addr; /* dirección del server donde se conectara */


int crearSocketCliente(struct sockaddr_in* serv_addr, int port, struct hostent* he)
{
	int sockdesc;

	/* Creamos el socket */
	if ((sockdesc = socket(AF_INET, SOCK_STREAM, 0)) < 0)
	{
		perror("ERROR socket");
		return(-1);
	}
	/* Establecemos serv_addr con la dirección del server */
	serv_addr->sin_family = AF_INET;
	serv_addr->sin_port = htons(port);
	serv_addr->sin_addr = *((struct in_addr *)he->h_addr);



	printf("[Cliente: Socket TCP creado]\n");

	return sockdesc;

}

int conectarCliente(int sockfd,struct sockaddr_in* server_addr)
{
  /* Intentamos conectarnos con el servidor */
	if (connect(sockfd, (struct sockaddr *)server_addr, sizeof(struct sockaddr)) < 0)
	{
//		perror("ERROR connect");
		return(-1);
```

```
	}
	printf("[Cliente: Conexión establecida con el servidor IP %s, port %d] \n",(char*)inet_ntoa(server_addr->sin_addr),ntohs(server_addr->sin_port));

	return (0);
}

int recibir(int sock, char* buf, int longitud)
{
	int numbytes;
	if ((numbytes=recv(sock, buf, longitud, 0)) == -1)
	{
		perror("ERROR recv");
		return (-1);
	}
	return numbytes;
}

int enviar(int sock, char* buf, int longitud)
{
	if (send(sock, buf, longitud, 0) == -1)
	{
		perror("ERROR send");
		return(-1);
	}
	return 0;
}


int main(int argc,char *argv[])
{
  if (argc != 2)
	{
		fprintf(stderr,"uso: client hostname\n");
		exit(1);
	}
```

```
	if ((he=gethostbyname(argv[1])) == NULL)
	{
		printf("Error con el hostname");
		exit(1);
	}

	if ((socketfd = crearSocketCliente(&serv_addr,PORT,he)) < 0) //Creamos el socket y nos fijamos si fue posible
	{
		printf("Imposible crear el cliente. \n");
		exit(1);
	}

	printf("Conectado\n");
	while(conectarCliente(socketfd, &serv_addr) != 0i);
	if ((numbytes = recibir(socketfd, buf0, MAXDATASIZESOLIC)) < 0)
	{
		printf("No se puede recibir el mensaje\n\n");
	}
	buf0[numbytes] = '\0';

	printf("\nRecibimos: %s \n\n", buf0);

	printf("Presione Intro:");
	getchar();
	//Pedimos el mensaje de 2048 bytes
	if (enviar(socketfd, "Quiero un mensaje largo", MAXDATASIZESOLIC) < -1)
	{
		printf("No se puede contactar con el servidor\n\n");
	}
	else
	{


		printf("Solicito el mensaje largo al servidor \n");
```

```
        gettimeofday(&ti, NULL);   // Instante inicial

        if ((numbytes = recibir(socketfd, buf1, MAXDATASIZELARGE)) < -1)
        {
                printf("No se recibió ninguna respuesta\n\n");
        }

        gettimeofday(&tf, NULL);   // Instante final
        tiempo= (tf.tv_sec - ti.tv_sec)*1000000 + (tf.tv_usec - ti.tv_usec);
        buf1[numbytes] = '\0';

        printf("Tiempo de espera del cliente para el mensaje largo %g microsegundos.\n\n",tiempo);
}
printf("Presione Intro");
getchar();

/* Envia mensaje de solicitud, y recibe dato de 4 bytes */
if (enviar(socketfd, "Solicito mensaje corto", MAXDATASIZESOLIC) < -1)
{
        printf("No se puede contactar con el servidor\n\n");
}
else
{
        printf("Solicito al servidor el mensaje corto\n");

        gettimeofday(&ti, NULL);   // Instante inicial

        if ((numbytes=recibir(socketfd, buf2, MAXDATASIZESHORT)) == -1)
        {
                printf("No se recibió ninguna respuesta\n\n");
        }
        gettimeofday(&tf, NULL);   // Instante final
        tiempo= (tf.tv_sec - ti.tv_sec)*1000000 + (tf.tv_usec - ti.tv_usec);
        buf2[numbytes] = '\0';
        printf("Tiempo de espera del cliente para el mensaje corto: %g microsegundos.\n\n",tiempo);
}
```

```
        /* Devolvemos recursos al sistema */
        close(socketfd);

        return 0;

}
```

Comentario [49]: <!--EndFragment-->

Comentario [50]: <!--StartFragment-->

servidor.c

```
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/un.h>

#define BACKLOG 10 /* Tamaño de la cola de conexiones recibidas */
#define PORT 13490    /* Puerto del servidor */
#define MAXDATASIZESHORT 4
#define MAXDATASIZESOLIC 32
#define MAXDATASIZELARGE 2048

double tiempo;
struct timeval ti, tf;    //Structuras para medir el tiempo
int sockdesc; //Descriptor socket: El servidor escuchara por sockdesc
int sockclient; //  Descriptor socket cliente: las transferencias de datos se realizar mediante sockclient
struct sockaddr_in my_addr; // contendra la direccion IP y el numero de puerto   local
struct sockaddr_in client_addr; // Contendra la direccion IP y numero de puerto del cliente
char buf[MAXDATASIZESOLIC]; //Buffer donde se reciben las solicitudes de cliente

int newfd;
char cuento[MAXDATASIZELARGE] = "<<<<<<<<<<<<<<<<<ACA ESTA EL MENSAJE
```

```
LARGO>>>>>>>>>>>>>>>>>>>>>";

int* status;
int sin_size = sizeof(struct sockaddr_in*);

int crearSocketServer(struct sockaddr_in* my_address, int port)
{
	int sockfd;

	if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) /*Crea un socket y verifica si hubo algun error*/
	{
		perror("socket");
		return (-1);
	}

	/* Asignamos valores a la estructura my_addr para luego poder llamar a la funcion bind() */
	my_address->sin_family = AF_INET;
	my_address->sin_port = htons(port); /* Puerto: debe convertirse a network byte order */
	my_address->sin_addr.s_addr = INADDR_ANY; /* automaticamente usa la IP local */




	/* Le asignamos un nombre al socket */
	if (bind(sockfd, (struct sockaddr *) my_address, sizeof(struct sockaddr)) < 0)
	{
		perror("ERROR bind");
		close(sockfd);
		return (-2);
	}
	printf("[Servidor: Socket TPC creado. Port %d ]\n",ntohs(my_address->sin_port));

	/* Habilitamos el socket para recibir conexiones, definiendo el tamaño de la cola de espera de conexiones */
	if (listen(sockfd, BACKLOG) < 0)
```

```
	{
		perror("ERROR listen");
		close(sockfd);
		return (-3);
	}

	return sockfd;
}

int aceptarConexion(int sockfd,struct sockaddr_in* their_addr)
{
	printf("[Servidor: a la espera de conexiones de clientes]\n");

	/*Se espera por conexiones ,*/
	if ((newfd = accept(sockfd, (struct sockaddr *)their_addr, &sin_size)) == -1)
	{
		perror("ERROR accept");
		return (-1);
	}
	printf("[Servidor: acepta conexion desde: %s, puerto %d]\n", (char*)inet_ntoa(their_addr->sin_addr), ntohs(their_addr->sin_port));

	return newfd;
}

int recibir(int sock, char* buf, int longitud)
{
	int numbytes;
	if ((numbytes=recv(sock, buf, longitud, 0)) == -1)
	{
		perror("ERROR recv");
		return (-1);
	}
	return numbytes;
}
```

```
int enviar(int sock, char* buf, int longitud)
{
	if (send(sock, buf, longitud, 0) == -1)
	{
		perror("ERROR send");
		return(-1);
	}
	return 0;
}

main()
{

	sockdesc = crearSocketServer(&my_addr,PORT); //Crea el socket, inicializa la direccion y realiza el bind y listen
	if (sockdesc < 0){
		printf("El servidor no puede arrancar \n");
		exit(1);
	}
	while(1) /* Se esperan conexiones */
	{
		sockclient = aceptarConexion(sockdesc, &client_addr);

		if (!fork()){
		// Proceso hijo, recibe y envia datos mediante sockclient
			enviar(sockclient, "Conexión Aceptada", MAXDATASIZESOLIC);

			if (recibir(sockclient,buf,MAXDATASIZESOLIC) >= 0){
				gettimeofday(&ti, NULL);   // Instante inicial
				enviar(sockclient, cuento, MAXDATASIZELARGE);
				gettimeofday(&tf, NULL);   // Instante final
				tiempo= (tf.tv_sec - ti.tv_sec)*1000000 + (tf.tv_usec - ti.tv_usec);
				printf("Tiempo para atender el mensaje largo %g microsegundos.\n\n",tiempo);
				}
			if (recibir(sockclient,buf,MAXDATASIZESOLIC) >= 0){
				gettimeofday(&ti, NULL);   // Instante inicial
```

```
                        enviar(sockclient, "Mensaje Corto", MAXDATASIZESHORT);
                        gettimeofday(&tf, NULL);   // Instante final
                        tiempo= (tf.tv_sec - ti.tv_sec)*1000000 + (tf.tv_usec - ti.tv_usec);
                        printf("Tiempo para atender el mensaje corto %g microsegundos.\n\n",tiempo);
}
                close(sockclient);
                exit(0);
            }
    close(sockclient);
            /* Se suspende la ejecucion del proceso padre hasta que finalice el proceso hijo */
            wait(status);
            printf("\n");

        }
}
```

Comentario [51]: <!--EndFragment-->

Punto5A

MiniFuncion.c

Comentario [52]: <!--StartFragment-->

```
 #include <sys/shm.h>


 /*Datos para obtener la clave unica*/
#define enteroCompartido 2
#define archivoCompartido "/bin/ls"

key_t clave;
int memory_id;
int semaforo_cliente;
int semaforo_servidor;

typedef struct msj
```

```
{
	int operacion;
	char resultado[100];
} msj;

msj *memoria;

//retorna la clave generada a partir de file e int.
int obtenerClave(char *file, int i)
{
	clave = ( ftok ( file, i) );
	if( clave == -1)
	{
		printf("No se pudo obtener la clave para la memoria compartida\n");
		exit(0);
	}
	return clave;
}

//Si esta creada solo obtenemos el id (omitiendo poner IPC_CREAT).
//Retorna 1 si la creo, otro valor caso contrario.
int crearMemoria()
{
	//intento obtener id de una memoria ya creada, caso contrario, la creo
	memory_id = ( shmget (clave, sizeof(struct msj), 0777 ) );
	if( memory_id == -1)
	{
		//creo la memoria compartida.
		memory_id = ( shmget (clave, sizeof(struct msj) , 0777 | IPC_CREAT) );
		return 1;
	}
	if( memory_id == -1)
	{
		printf("ID");
		printf("ERROR en la creacion de la memoria. Abortando...\n");
		exit(0);
```

```
        }
        return 0;
}

//Con el id obtenido, direccionamos la memoria a un puntero a int.
void direccionarMemoria()
{
        memoria  = (msj *)shmat (memory_id, (char *)0, 0);
        if( memoria == NULL)
        {
                printf("[CLNT]No se pudo obtener la memoria\n");
                exit(0);
        }
}

//Procedimiento para eliminar la memoria compartida.
//La deberia eliminar el proceso creador.
void eliminarMemoria();
```

Comentario [53]: <!--EndFragment-->

semaforo.h

```
/*
Definicion e implementacion de semaforo
*/
#ifndef SEMAFORO_H_INCLUDED
#define SEMAFORO_H_INCLUDED

# include <sys/types.h>
# include <sys/ipc.h>
# include <sys/sem.h>

union semun {
  int val;
}arg;

int crearSemaforo(int key){
```

```
  return(semget(key, 1, IPC_CREAT|0666));
}

int asociarseSemaforo(int key){
  return(semget(key,1,0));
}

int initSemaforo(int semid, int valor){
  arg.val = valor;
  if (semctl(semid, 0, SETVAL, arg)==-1){return(-1);}
  else{return(0);}
}

int signal(int semid){
  struct sembuf operacion;
  operacion.sem_num = 0;
  operacion.sem_op = 1;
  operacion.sem_flg = 0;
  if (semop(semid, &operacion, 1)==-1){return(-1);}
  else{return(0);}
}

int wait(int semid){
  struct sembuf operacion;
  operacion.sem_num = 0;
  operacion.sem_op = -1;
  operacion.sem_flg = 0;
  if (semop(semid, &operacion, 1)==-1){return(-1);}
  else{return(0);}
}


#endif // SEMAFORO_H_INCLUDED
```

cliente.c

Comentario [54]: <!--StartFragment-->

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <time.h>
#include "MiniFuncion.h"
#include "semaforo.h"

int op;
struct timeval ti,tf;
long tiempo;

void llamar();
void eliminarMemoria();

int main()
{
  clave = obtenerClave(archivoCompartido, enteroCompartido);
  crearMemoria();
  direccionarMemoria();
  semaforo_cliente=asociarseSemaforo(obtenerClave(archivoCompartido,enteroCompartido));

  semaforo_servidor=asociarseSemaforo(obtenerClave(archivoCompartido,enteroCompartido+1));

  while(1)
  {
    printf("Esperando al servidor\n");
    wait(semaforo_cliente);
    printf("---MiniFuncion---\n");
    printf("(1)Obtener hora de sistema\n");
    printf("(2)Obtener fecha de sistema\n");
    printf("(3)Obtener nombre de la máquina\n");
    printf("(4)Obtener cantidad de usuarios logueados\n");
    printf("(0)Salir\n");
     scanf("%i",&op);
    while (op<0 || op>4)
```

```
    {
      printf("Ingrese nuevamente\n");
      scanf("%i",&op);
    }

    if (op==0)
    {
      memoria->operacion=op;
      printf("Saliendo\n");
      signal(semaforo_servidor);
      exit(0);
    }

    gettimeofday(&ti, NULL);
    llamar();
    gettimeofday(&tf, NULL);
    tiempo = (tf.tv_sec - ti.tv_sec)*1000000 + (tf.tv_usec - ti.tv_usec);

    printf("Resultado=%s\n",memoria->resultado);
    printf("El tiempo que tardó es de %d us\n\n",tiempo);


  }
}

  void llamar()
  {

    memoria->operacion=op;
    signal(semaforo_servidor);
    wait(semaforo_cliente);

  }

  void eliminarMemoria()
{
```

```
        if (memory_id  != -1)
        {
                shmdt ((char *)memoria);
        }
}
```

Comentario [55]: <!--EndFragment-->

servidor.c

Comentario [56]: <!--StartFragment-->

Comentario [57]: <!--EndFragment-->

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <time.h>
#include "MiniFuncion.h"
#include "semaforo.h"

void eliminarMemoria();

int main()
{
  int ch=0;
  int lines=0;
  FILE *fp;
  clave = obtenerClave(archivoCompartido, enteroCompartido);
  printf("Obtuve la clave.\n");
  crearMemoria();
  printf("Cree la memoria.\n");
  direccionarMemoria();
  printf("Memoria LISTA para usar.\n");


  semaforo_cliente=crearSemaforo(obtenerClave(archivoCompartido,enteroCompartido));
  semaforo_servidor=crearSemaforo(obtenerClave(archivoCompartido,enteroCompartido+1));

  while(1)
```

```
{
 signal(semaforo_cliente);
 printf("Esperando solicitud\n");
 wait(semaforo_servidor);
 switch (memoria[0].operacion)
 {
  case 0:
  {
       printf("Terminado");
       eliminarMemoria();
       wait(semaforo_cliente);
       exit(0);
  }
  case 1:
  {
       time_t t;
       struct tm *tm;
       t=time(NULL);
       tm=localtime(&t);
       strftime(memoria[0].resultado, 100, "%H:%M:%S", tm);
       break;
  }

  case 2:
  {
       time_t t;
       struct tm *tm;
       t=time(NULL);
       tm=localtime(&t);
       strftime(memoria[0].resultado, 100, "%d/%m/%y", tm);
       break;
  }

  case 3:
  {
       gethostname(memoria[0].resultado,100);
```

```
      break;
}

case 4:
{

      system ("who >> usuarios.txt");

       fp = fopen("usuarios.txt","r");

       if (fp==NULL){
         printf ("No existe el archivo\n");
         return 0;
       }




       lines++;
       while ((ch = fgetc(fp)) != EOF)
       {
          if (ch == '\n')
                    lines++;

       }
      fclose(fp);
      system ("rm usuarios.txt");

      sprintf(memoria[0].resultado, "%i",lines );
      break;
}
```

```
    }
     signal(semaforo_cliente);
  }
  return 0;
}

void eliminarMemoria()
{
        shmdt (memoria);
        shmctl (memory_id , IPC_RMID, (struct shmid_ds *)NULL);
}
```

Punto5B

ej5b.x

Comentario [58]: <!--StartFragment-->

Comentario [59]: <!--EndFragment-->

```
program ej5b {
  version ver_ej5b{
    string tiempo()=1;
    string fecha()=2;
    string nombre()=3;
    int usuariosLog()=4;
  }=1;
} = 0x20009991;
```

ej5b_client.c

Comentario [60]: <!--StartFragment-->

Comentario [61]: <!--EndFragment-->

```
/*
 * This is sample code generated by rpcgen.
 * These are only templates and you can use them
 * as a guideline for developing your own functions.
 */

#include "ej5b.h"
#include <string.h>
void menu();
void
```

```
ej5b_1(char *host)
{
        CLIENT *clnt;
        char * *result_1;
        char *tiempo_1_arg;
        char * *result_2;
        char *fecha_1_arg;
        char * *result_3;
        char *nombre_1_arg;
        int  *result_4;
        char *usuarioslog_1_arg;
        int opcion;
        opcion=5;

#ifndef DEBUG
        clnt = clnt_create (host, ej5b, ver_ej5b, "udp");
        if (clnt == NULL) {
                clnt_pcreateerror (host);
                exit (1);
        }
#endif  /* DEBUG */

        while (opcion!=0){
          menu();
          scanf("%i", &opcion);
          if (opcion==1){
            result_1 = tiempo_1((void*)&tiempo_1_arg, clnt);
            if (result_1 == (char **) NULL) {
                    clnt_perror (clnt, "call failed");
            }
            printf("Hora del sistema:%s\n", *result_1);
          }
           if (opcion==2){
            result_2 = fecha_1((void*)&fecha_1_arg, clnt);
            if (result_2 == (char **) NULL) {
                    clnt_perror (clnt, "call failed");
```

```
            }
            printf("Fecha del sistema:%s\n", *result_2);
          }
          if (opcion==3){
            result_3 = nombre_1((void*)&nombre_1_arg, clnt);
            if (result_3 == (char **) NULL) {
                    clnt_perror (clnt, "call failed");
            }
            printf("Nombre del server:%s\n", *result_3);

          }
          if (opcion==4){
            result_4 = usuarioslog_1((void*)&usuarioslog_1_arg, clnt);
            if (result_4 == (int *) NULL) {
                    clnt_perror (clnt, "call failed");
            }
            printf("Cantidad de usuarios Logueados:%i\n", *result_4);

          }
          if (opcion==0){
             printf ("FIN\n");
          }

        }
}


int
main (int argc, char *argv[])
{
        char *host;

        if (argc < 2) {
                printf ("usage: %s server_host\n", argv[0]);
                exit (1);
        }
```

```
        host = argv[1];
        ej5b_1 (host);
exit (0);
}

void menu (){
  printf("\n************************ Preguntas ************************\n");
  printf("1- Solicitar la hora del sistema\n1- Solicitar la fecha del sistema\n3- Solicitar el nombre del sistema\n4- Solicitar informacion sobre la cantidad de usuarios logueados\n0- Finalizar\n \n");

}
```

ej5b_server.c

Comentario [62]: <!--StartFragment-->

Comentario [63]: <!--EndFragment-->

```
/*
 * This is sample code generated by rpcgen.
 * These are only templates and you can use them
 * as a guideline for developing your own functions.
 */

#include "ej5b.h"
#include "time.h"
#include <utmp.h>
#include <err.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
char **
tiempo_1_svc(void *argp, struct svc_req *rqstp)
{
        static char * result;
        char output[128];
        // struct para obtener tiempo
        const struct tm *loctim;
        // operaciones para obtener la hora del sistema
        time_t rawtime;
        rawtime=time(0);
```

```
        loctim = localtime ( &rawtime );
        // guardar en result la hora

    strftime(output,128,"%H:%M:%S",loctim);
    result=output;
        return &result;
}

char **
fecha_1_svc(void *argp, struct svc_req *rqstp)
{
        static char * result;

        // struct para obtener tiempo
        const struct tm *loctim;
        // operaciones para obtener la hora del sistema
        time_t rawtime;
        time ( &rawtime );
        loctim = localtime ( &rawtime );
        // guardar en result la hora
        result = asctime(loctim);

        return &result;
}

char **
nombre_1_svc(void *argp, struct svc_req *rqstp)
{
        static char * result;
        char mensaje[256];
        gethostname(mensaje, 256);
        result=mensaje;
        return &result;
}

int *
```

```
usuarioslog_1_svc(void *argp, struct svc_req *rqstp)
{
        static int result;
        FILE *fp ;
        int ch=0;
        int lines=0;
      system ("who >> usuarios.txt");
        fp = fopen("usuarios.txt","r");
        if (fp == NULL)
          return 0;
        lines++;
        while ((ch = fgetc(fp)) != EOF)
        {
          if (ch == '\n')
                lines++;
        }
        fclose(fp);
        system ("rm usuarios.txt");

       result=lines;
       return &result;
}
```

Punto 6

par.c

Comentario [64]: <!--StartFragment-->

```
#include "socket.h"
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <stdlib.h>

char buf[MSG_SIZE];
void Preguntas(char * mensaje);
char * ingresarMensaje();
void mostrar(char * msg);
```

```
void Menu();
void MostrarFuncionamiento();
void nombreMaquina(char * mensaje);
void horaMaquina(char * mensaje);
void Clima(char * mensaje);
void temperaturaXML ();
int saltosSubStringTemperatura (char* temperatura);

int main(int argc, char **argv)
{
   char * amigo;
   char * mensaje;
   char * puerto;
   //controlamos la cantidad de parametros ingresada
   if(argc<3 || argc>3)
   {
                printf("Cantidad de parametros invalido, debe ingresar <IP AMIGO,PUERTO> -\n");
                exit(1);
   }

   amigo=argv[1];
   puerto = argv[2];
   //mensaje= (char*)malloc (sizeof(char)*MSG_SIZE);
   MostrarFuncionamiento();
  if ( amigoConectado(amigo, puerto)){//Modo cliente
         while(1)
         {
         Menu();
            mensaje = ingresarMensaje();
            if(strcmp(mensaje,"5\n")==0){
                enviarMsj(mensaje);
                break;
            }
         enviarMsj(mensaje);
         mensaje = recibirMsj();
         mostrar(mensaje);
```

```
        mensaje = recibirMsj();
        if(strcmp(mensaje,"5\n")==0)
           break;
        Preguntas(mensaje);
        enviarMsj(mensaje);
         }
                cerrarSesion();
   }
   else //Modo Servidor
   {
      iniciarServicio(puerto);
         while(1){
           mensaje = recibirMsj();
           if(strcmp(mensaje,"5\n")==0)
                break;
           Preguntas(mensaje);
           enviarMsj(mensaje);
           Menu();
        mensaje = ingresarMensaje();
                if(strcmp(mensaje,"5\n")==0)
           {
              enviarMsj(mensaje);
              break;
           }
                enviarMsj(mensaje);
                mensaje = recibirMsj();
                mostrar(mensaje);
                }
                cerrarSesion();
   }
   exit(0);
}


//Lee desde la consola los caracteres necesarios
char * ingresarMensaje()
```

```
{
   buf[0]='\0';
   printf("[SD2014]$");
   //lee hasta el caracter MSG_SIZE escrito en consola
   fgets(buf, MSG_SIZE+1, stdin);
   return buf;
}

/** Muestra el mensaje recibido **/
void mostrar(char * msg)
{
   printf("> %s\n",msg);
}

//Muestra las posibles preguntas que se pueden hacer
void Menu()
{
   printf("\n#################¿QUE INFORMACION DESEA OBTENER DEL PROCESO AMIGO?#################\n");
        printf("1- Los usuarios logueados del sistema\n2- La hora del sistema \n3- Nombre de la maquina\n4- Temperatura de la localidad donde esta ubicado\n-5 'Salir'\n \n");
}
//explica el funcionamiento del programa
void MostrarFuncionamiento()
{
  printf("\nfuncionamiento:El primero en conectarse sera el proceso servidor \n\n");
}

//Nos devuelve el nombre de la maquina
void nombreMaquina(char * mensaje){

        if(gethostname(mensaje, MSG_SIZE)<0)
        {
                printf("Error al obtener el hostname.-\n");
                exit(1);
        }
```

```
}

//Nos devuelve la hora del sistema
void horaMaquina(char * mensaje)
{
   char * tiempo;
         time_t t;
         struct tm * tiem;
         t = time(NULL);
         tiem= localtime(&t);
         strftime(mensaje, 10, "%H:%M:%S", tiem);

}



//Nos devuelve el clima de BB
void Clima(char * mensaje)
{
   system("wget http://xml.tutiempo.net/xml/42815.xml");
   temperaturaXML(mensaje);
   system("rm 42815.xml");
}

void temperaturaXML (char * mensaje){
  char* temperatura=(char*)malloc(sizeof(char)*300);
  char* resu=(char*)malloc(sizeof(char)*3);
  char* buscar = "<temperatura>";
  char* aux;
  int saltos;
  FILE *xml= fopen("42815.xml","r");
  int continuar=1;
  //no se pudo conectar al servidor, por lo tanto se pide que se verifica la conexion
  if (!xml)
    strcpy(mensaje,"No se pudo conectar con el servidor del clima 'tutiempo.net', verifique la conexion a internet
del proceso amigo");
```

```
  else{
    while (feof(xml) == 0 && continuar)
          {
                fgets(temperatura,300,xml);
                aux = strstr(temperatura,buscar);
                if (aux!=NULL)
                  continuar=0;
          }
    fclose(xml);
    //obtiene la temperatura y la copia en resu
    strncpy (resu,aux+strlen(buscar),saltosSubStringTemperatura(aux));
    /*//en caso de que la temperatura sea de un digitos
    if (*(aux+strlen(buscar)+2)=='<')
          strncpy (resu,aux+strlen(buscar),1);
    //en caso de que la temperatura sea de 2 digitos
    else
      strncpy (resu,aux+strlen(buscar),2);*/
    free(temperatura);
    strncpy(mensaje,resu,2);
    strcat(mensaje,"º en la ciudad de Bahía Blanca");
  }
}

int saltosSubStringTemperatura (char* temperatura){
  char* inicio = "<temperatura>";
  int resu=0;
  while (*(temperatura+strlen(inicio)+resu)!='<')
    resu++;
  return resu;
}

//Nos informa de los usuarios logueados en el sistema
void usuariosLogueados(char * mensaje)
{
FILE * log;
    char linea [100];
```

```
    system("who > usuarios.sd");
    log = fopen ("usuarios.sd" , "r");

    strcpy(mensaje, "");

    while( fgets(linea, 100, log) != NULL ){
        strcat( mensaje, linea);
    }
    system("rm usuarios.sd");
    fclose(log);
}

//Es la encargada de ver que pergunta es la que me hicieron y llama al procedimiento correspondiente
void Preguntas(char * mensaje)
{
    if(strcmp(mensaje, "1\n")==0)
        usuariosLogueados(mensaje);
    else if(strcmp(mensaje, "2\n")==0)
            horaMaquina(mensaje);
    else if(strcmp(mensaje, "3\n")==0)
            nombreMaquina(mensaje);
    else if(strcmp(mensaje, "4\n")==0)
            Clima(mensaje);
    else
         strcpy(mensaje, "Opcion invalida");
}
```

Comentario [65]: <!--EndFragment-->

socket.c

Comentario [66]: <!--StartFragment-->

Comentario [67]: <!--EndFragment-->

```
#define MSG_SIZE        512 //tamaño del mensaje
#define MAXBUF          1025

//verifico si hay un amigo ya conectado y si es asi se inicia como cliente
int amigoConectado (char *hostname, char * port);
```

```
// Envio un mensaje al amigo
void enviarMsj (char* msj);

//Recibo un mensaje del amigo
char* recibirMsj ();

//Inicio una conexion como servidor para que luego se conecte e amigo
void iniciarServicio ();

//cerrar la conexion con el amigo
int cerrarSesion ();
```

socket.c

Comentario [68]: <!--StartFragment-->

Comentario [69]: <!--EndFragment-->

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <string.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "socket.h"

int sockAddrSize;
int clntSocket, miSocket, hostPort; //manejador para el socket
struct hostent * hostInfo; //guarda la información del host
struct sockaddr_in direccion; //struct para la info del socket

//verifico si hay un amigo ya conectado
int amigoConectado (char * nombreHost, char * puerto){
   clntSocket = 0;
   char hostName[255];
   strcpy(hostName, nombreHost);
   hostPort = atoi(puerto);
```

```
    sockAddrSize = sizeof (struct sockaddr_in);

    //creo el socket orientado a conexión (TCP)
    miSocket = socket(AF_INET, SOCK_STREAM, 0);
    if (miSocket == -1)
    {
        printf("Error al crear el socket\n");
        exit(-1);
    }
    //obtengo informacion del host a partir del hostname
    if ((hostInfo = gethostbyname(hostName)) == NULL)
    {
        printf("IP invalida \n.");
        exit(-1);
    }

    direccion.sin_addr = *((struct in_addr *) hostInfo->h_addr);//direccion del server
    direccion.sin_port = htons(hostPort); //puerto donde se hara la comunicacion, htons usa un lenguaje independiente de la arquitectura del cpu
    direccion.sin_family = AF_INET; //es el tipo de conexión (por red o interna)
    //Solicitar conexión con el servidor , Dicha función connect quedará bloqueada hasta que el servidor acepte nuestra conexión o bien si no hay servidor en el sitio indicado, saldrá dando un error retornando -1
    if (connect(miSocket, (struct sockaddr*) & direccion, sizeof (direccion)) == -1)
      return 0;
    else{
      printf("Soy el cliente y establecí una comunicación con %s en el puerto %d\n\n", hostName, hostPort);
      return 1;
    }
}

// Envio un mensaje al amigo
void enviarMsj (char* msj){
    char msg[MSG_SIZE];
    strcpy(msg, msj);
    if (clntSocket)//Entra cuando es servidor
            send(clntSocket, msg, MSG_SIZE, 0);
```

```
    else //Entra cuando es cliente
            send(miSocket, msg, MSG_SIZE, 0);
}

//Recibo un mensaje del amigo
char* recibirMsj (){
    char * msg = malloc(MSG_SIZE);
    if (clntSocket) //Entra cuando es servidor
             recv(clntSocket, msg, MSG_SIZE, 0);
    else //Entra cuando es cliente
        recv(miSocket, msg, MSG_SIZE, 0);
    return msg;
}

//Inicio una conexion como servidor
void iniciarServicio (char * port){
    //inicia en modo servidor
    sockAddrSize = sizeof (struct sockaddr_in);
    //creo el socket orientado a conexión (TCP)
    miSocket = socket(AF_INET, SOCK_STREAM, 0);

    if (miSocket == -1) {
        printf("No se puede crear el socket\n");
        exit(-1);
    }
    hostPort = atoi(port);

    direccion.sin_addr.s_addr = INADDR_ANY; //acepta cualquier cliente
    direccion.sin_port = htons(hostPort);//puerto donde se hara la comunicacion, htons usa un lenguaje independiente de la arquitectura del cpu
    direccion.sin_family = AF_INET; //es el tipo de conexión (por red o interna)

    //Avisar al sistema operativo de que hemos abierto un socket y queremos que asocie nuestro programa(puerto del mismo) a dicho socket
    if (bind(miSocket, (struct sockaddr*) & direccion, sizeof (direccion)) == -1) {
        printf("Error al intentar asociar un puerto con un socket (bind) \n");
```

```
        exit(-1);
    }

    //Avisar al sistema de que comience a atender dicha conexión de red.
    if (listen(miSocket, 1) == -1) {
        printf("Error el intentar escuchar en el puerto %i\n",hostPort);
        exit(-1);
    }

    printf("Soy el servidor y escucho en el puerto %i\n\n",hostPort);
    //Pedir y aceptar las conexiones de clientes al sistema operativo.
    clntSocket = accept(miSocket, (struct sockaddr*) & direccion, (socklen_t *) & sockAddrSize);

}

//cerrar la conexion con el amigo
int cerrarSesion (){
  printf ("Se finalizara la conexion...\n");
  //verifico la existe del socket cliente
  if (clntSocket)
    {
      // cerrar la conexión del descriptor de socket cliente
        if (close(clntSocket) == -1)
        {
            printf("No se puede cerrar la sesion\n");
            exit(-1);
             }
   }
   //verifico la existencia del socket servidor
   if (miSocket)
   {
        // cerrar la conexión del descriptor de socket servidor
        if (close(miSocket) == -1)
        {
            printf("No se puede cerrar la sesion\n");
            exit(-1);
```

```
        }
    }
  printf ("conexion Finalizada\n");
}
```

Punto7

ej7.x

Comentario [70]: <!--StartFragment-->

Comentario [71]: <!--EndFragment-->

```
struct tiempo {
        unsigned long seconds;
        unsigned long useconds;
    };

program ej7 {
        version ej7v1 {
                tiempo horaSistema() = 1;
        } = 1;
} = 0x20000001;
```

ej7_client.c

Comentario [72]: <!--StartFragment-->

Comentario [73]: <!--EndFragment-->

```
/*
 * This is sample code generated by rpcgen.
 * These are only templates and you can use them
 * as a guideline for developing your own functions.
 */

#include "ej7.h"
#include <sys/time.h>

void
ej7_1(char *host)
{
        CLIENT *clnt;
        tiempo  *TiempoServer;
        char *horasistema_1_arg;
        long ajuste,diferencia;
```

```
	struct timeval ti,tf;
#ifndef DEBUG
		clnt = clnt_create (host, ej7, ej7v1, "tcp");
		if (clnt == NULL) {
			clnt_pcreateerror (host);
			exit (1);
		}
#endif	/* DEBUG */

	gettimeofday(&ti,NULL);//tiempo inicial local
		TiempoServer = horasistema_1((void*)&horasistema_1_arg, clnt);
		gettimeofday(&tf,NULL); //tiempo final local
		if (TiempoServer == (tiempo *) NULL) {
			clnt_perror (clnt, "call failed");
		}
		ajuste = (TiempoServer->seconds*1000000+TiempoServer->useconds) +
((tf.tv_sec*1000000+tf.tv_usec)-(ti.tv_sec*1000000+ti.tv_usec))/2;
		diferencia = (tf.tv_sec*1000000+tf.tv_usec) - ajuste;
		printf("\nLa variación del reloj local con respecto a la maquina %s es : %ld us\n\n",host,diferencia);

#ifndef DEBUG
		clnt_destroy (clnt);
#endif	/* DEBUG */
}


int
main (int argc, char *argv[])
{
		char *host;
		int i=1;
		if (argc < 2) {
			printf ("usage: %s server_host\n", argv[0]);
			exit (1);
		}
		while (i<argc){
```

```
            ej7_1 (argv[i]);
            i++;
          }
exit (0);
}
```

ej7_server.c

Comentario [74]: <!--StartFragment-->

Comentario [75]: <!--EndFragment-->

```
/*
 * This is sample code generated by rpcgen.
 * These are only templates and you can use them
 * as a guideline for developing your own functions.
 */

#include "ej7.h"
#include <sys/time.h>

tiempo *
horasistema_1_svc(void *argp, struct svc_req *rqstp)
{
        static tiempo  result;
      struct timeval tiempo;

        gettimeofday(&tiempo,NULL);
        result.seconds=tiempo.tv_sec;
        result.useconds=tiempo.tv_usec;

        return &result;
}
```

Punto 12

SocketCliente.java

Comentario [76]: <!--StartFragment-->

```
import java.net.*;
import java.io.*;
import java.util.Scanner;
```

```
public class SocketCliente
 {
    public static void main (String [] args)
    {
       if (args.length<2){
          System.out.println("debes introducir un <IP,PUERTO>");
          System.exit(0);
            }
       new SocketCliente(args[0],Integer.parseInt(args[1]));
    }

    public SocketCliente(String ip, int puerto)
    {
       try
       {
              //para medir tiempos
              long time_start, time_end;
        //dato que se le va a pedir al servidor
              String pedido;
              String rta;
              //para leer por pantalla
              Scanner lector = new Scanner(System.in);
          /* Se crea el socket cliente */
          Socket socket = new Socket (ip,puerto);
          if (socket!=null)
              System.out.println ("[SD2014]> conectado");
             else{
              System.out.println ("[SD2014]> no se pudo establer conexion con el servidor: "+ip+":"+puerto);
              System.exit(0);
              }
          /* Se obtiene un stream de lectura/escritura para tipos de datos simples */
          DataInputStream entrada = new DataInputStream (socket.getInputStream());
          DataOutputStream salida = new DataOutputStream (socket.getOutputStream());
```

```
            //intercambio de informacion
            boolean continuar=true;
            while (continuar){
                //se le muestras las opciones al cliente y se le pide que eleja una
                mostrarOpciones();
                System.out.print("[SD2014]> ");
                pedido=lector.nextLine();
                if (pedido.equals("5")){
                        continuar=false;
                        salida.writeUTF(pedido);
                }
                else{
                    //enviamos el pedido al servidor
                    time_start = System.nanoTime();
                    salida.writeUTF(pedido);
                    rta=entrada.readUTF();
                    time_end = System.nanoTime();
                    //respues del servidor
                    System.out.println ("[SD2014]> Respuesta: " + rta);
                    System.out.println ("[SD2014]> Tiempo de respuesta: " + (time_end-time_start) + " ns");
                    System.out.println ();
                }
            }
            System.out.println ("[SD2014]> conexion terminada.");
        }
        catch (java.net.ConnectException a)
        {
            System.out.println ("[SD2014]> Primero debes iniciar al servidor o verificar que los datos son correctos
");
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }

    }
```

```
    private void mostrarOpciones (){
        System.out.println("[SD2014]> Elija una opcion para obtener los datos del servidor.");
        System.out.println("[SD2014]> - (1) hora del sistema - (2) fecha del sistema - (3) nombre de la maquina - (4) cantidad de usuarios logueados - (5) salir");
    }
}
```

Comentario [77]: <!--EndFragment-->

SocketServidor.java

Comentario [78]: <!--StartFragment-->

Comentario [79]: <!--EndFragment-->

```
import java.net.*;
import java.io.*;
import java.util.Calendar;
import java.util.GregorianCalendar;
public class SocketServidor
{
   public static void main (String [] args)
   {
          if (args.length==0){
        System.out.println("debes introducir un puerto para la comunicación.");
        System.exit(0);
          }
      // Se instancia la clase principal para que haga todo lo que tiene que
      // hacer el ejemplo

      new SocketServidor(args[0]);
   }


   public SocketServidor(String puerto)
   {
      try
      {
         // Se crea un socket servidor atendiendo a un determinado puerto.

         ServerSocket socket = new ServerSocket (Integer.parseInt(puerto));
```

```
// Se acepata una conexi[illegible]n con un cliente. Esta llamada se queda
// bloqueada hasta que se arranque el cliente.
System.out.println ("Esperando cliente");
Socket cliente = socket.accept();
System.out.println ("Conectado con cliente de " + cliente.getInetAddress());

/* Se obtiene un stream de lectura/escritura para tipos de datos simples */
 DataInputStream entrada = new DataInputStream (cliente.getInputStream());
 DataOutputStream salida = new DataOutputStream (cliente.getOutputStream());

// setSoLinger() a true hace que el cierre del socket espere a que
// el cliente lea los datos, hasta un máximo de 10 segundos de espera.
// Si no ponemos esto, el socket se cierra inmediatamente y si el
// cliente no ha tenido tiempo de leerlos, los datos se pierden.
cliente.setSoLinger (true, 10);

boolean continuar=true;
String pedido;
while (continuar){
      pedido = entrada.readUTF();
      switch (pedido){
      case "1": {salida.writeUTF(obtenerHora());break;}
      case "2": {salida.writeUTF(obtenerFecha());break;}
      case "3": {salida.writeUTF(obtenerNombreMaquina());break;}
      case "4": {salida.writeUTF(obtenerCantUsuariosLogueados());break;}
      case "5": {continuar=false;break;}
      default : {salida.writeUTF("opcion invalida");break;}
  }
}


// Se cierra el socket con el cliente.
cliente.close();

// Se cierra el socket encargado de aceptar clientes.
```

```
            socket.close();
            System.out.println ("Finalizo la conexión");
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }

    private String obtenerHora(){
                    Calendar calendario = new GregorianCalendar();
                    return(calendario.get(Calendar.HOUR_OF_DAY) + ":" + calendario.get(Calendar.MINUTE) +
":" + calendario.get(Calendar.SECOND));
    }

    private String obtenerFecha(){
        Calendar calendario = new GregorianCalendar();
            return(calendario.get(Calendar.DAY_OF_MONTH) + "/" + calendario.get(Calendar.MONTH) + "/" +
calendario.get(Calendar.YEAR));
    }

    private String obtenerNombreMaquina(){
      String hostname = "Unknown";
      try{
           hostname=InetAddress.getLocalHost().getHostName();
      }
      catch (UnknownHostException ex)
      {
           System.out.println("Problema al obtener el hostname");
      }
      return hostname;
    }

    private String obtenerCantUsuariosLogueados(){
    int lNumeroLineas = 0;
    try {
```

```
        Process p = Runtime.getRuntime().exec("who");
        p.waitFor();
        BufferedReader reader = new BufferedReader(new InputStreamReader(p.getInputStream()));
        String line=reader.readLine();

        while (line != null) {
           lNumeroLineas++;
           line = reader.readLine();
        }

      }
      catch(IOException e1) {e1.printStackTrace();}
      catch(InterruptedException e2) {e2.printStackTrace();}

    return (""+lNumeroLineas);
   }

}
```