PL-01	07	Puga Fernández Maldonado Escobedo	Gonzalo Roberto Carlos
Nº PLo	Equipo	Apellidos	Nombre

71.779.257-Y 73208290	uo277906@uniovi.es UO297453@uniovi.es
DNI	e-mail

2	Instrumentación de un inyector de carga	
Nº Práctica	Título	Calificación

Comentarios sobre la corrección

Asignatura de

CONFIGURACIÓN Y EVALUACIÓN DE SISTEMAS

Curso 2022-2023

Área de Arquitectura y Tecnología de Computadores
Departamento de Informática de la Universidad de Oviedo

Índice

1. Objetivos de la práctica

El objetivo de esta práctica es practicar los conceptos de los temas vistos en medición y visualización. Para ello, el alumno aprenderá a instrumentar en Windows, añadiendo instrumentación al inyector desarrollado en la práctica anterior(práctica 1) y coordinándolo con el monitor de prestaciones de Windows.
A partir de los datos obtenidos se realizará un análisis gráfico.

El trabajo realizado servirá en futuras prácticas para desarrollar un análisis del estado del sistema que permita determinar cuáles son las máximas prestaciones que puede ofrecer el sistema y cuales son los elementos que las limitan.

2.Pruebas a realizar

El equipo se probará tres veces seguidas para garantizar su correcto funcionamiento y el alumno aprenderá a diferenciar entre los regímenes inestable y transitorio. El procedimiento de prueba será así:

Se utilizarán dos máquinas, una para el servidor y otra para el cliente, conectadas por un cable de red cruzado. En la máquina servidor se preparará un conjunto de recopiladores del monitor de rendimiento de Windows, con un intervalo de toma de muestras de un segundo.
Una vez reiniciado el sistema servidor, se lanzará el programa para crear el servidor, seguido del inyector de carga en la máquina cliente, y se dejará actuar 3 veces seguidas, sin interrumpir los recopiladores del monitor de rendimiento de Windows. Una vez concluidas las pruebas, se detendrá el monitor de rendimiento, y se introducirán en un Excel tanto los resultados del monitor de rendimiento como de la ejecución de los clientes para realizar un análisis del experimento..

3. Preguntas de la práctica

¿Cuál es el límite de productividad máxima alcanzable en cualquiera de los experimentos realizados? ¿Por qué?

En los tres experimentos, la productividad máxima que alcanza es 50,61.
En las 3 pruebas el valor ronda este número, por lo que el máximo alcanzable sería 50,61.

¿Cuál debería ser la duración del intervalo de arranque para que no influya en el experimento? Para responder a esta pregunta debes apoyarte en gráficas para cada prueba en la que se aprecie la evolución de alguna de las métricas (utilizaciones, errores de pag/s, tráfico de red, etc) con respecto al tiempo de medición. El alumno debe escoger el número y formato de gráficas convenientes e introducir las explicaciones textuales que considere adecuadas. Se valorará tanto la capacidad de análisis del alumno como la de síntesis, es decir, el alumno debe mostrar todas las gráficas que sean convenientes, pero no más de las necesarias. Indicar qué métrica o contador se ha utilizado para determinar la duración del transitorio.

Para que no influya en el experimento, la duración del intervalo de arranque tendría que ser menor al tiempo en que se lanza la primera petición. Entonces el tiempo que debería durar debería ser desde el momento que es 0 hasta el primer pico, que en este caso serian 2 ms.

¿Hay diferencias significativas entre las tres réplicas del experimento?

No.

Concepto	Réplica 1	Réplica 2	Réplica 3
Media del tiempo de	1,279392187	1,280729728	1,281344003

reflexión			
Media del tiempo de respuesta	477,8437952	477,594406	485,0679339
Productividad promedio	50,52333333	50,55	50,61
Media del tiempo inactivo del disco	78,41422724	75,00128989	81,09758324
Media de la longitud de la cola del disco	0,85952339	0,860316878	0,654421716
Media del número de transferencias de s	78,41422724	75,00128989	81,09758324
Media del Promedio en seg/transferencia	903,9429354	760,1684152	722,419458
Media de los bytes de caché	65577041,65	67007158,01	66703991,14
Media de los bytes disponibles	15987094171	15987663161	15978588405
Media del contador Errores de página/s.	18742,23924	16114,28641	14979,4664
Media del contador Pag./s	2802,33599	113,5580033	27,52820713
Media del % de procesador	93,54914072	93,80011714	89,73864376
Media del Total de bytes/s	0	0	0
Ancho de banda actual	100000	100000	100000

4. Código fuente del inyector

```
#include <windows.h>
#include <iostream>
#include <math.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <fstream>

using namespace std;

#define MAXPETICIONES 10000
#define MAXUSUARIOS 500
#define PUERTO 57000
#define TAM_PET 1250
#define TAM_RES 1250


// Inicializar estas variables en main antes de lanzar las pruebas
float tReflex;  // en segundos
char* ipServidor;
int numUsuarios;
int numPeticiones;

// Esta variable es global para mayor eficiencia. Se le asigna valor antes
// de lanzar varios hilos y luego la leen varios hilos en paralelo, pero como
// no le vuelven a asignar valor, no es necesario protegerla con un mutex

float ticksPorMilisegundo;
```

```
LARGE_INTEGER tickBase; // Todos los ticks seran relativos a este tiempo
LARGE_INTEGER tickInicio; // Instante en el que se empiezan a tomar tiempos: tickBase + ticksCalentamiento
LARGE_INTEGER tickFin;  // Instante en el que se acaba la prueba: tickInicio + ticksDuracion


struct datos {
	int contPet;
	float reflex[MAXPETICIONES];
	float tres[MAXPETICIONES];
	unsigned long ciclosIniPeticion[MAXPETICIONES];
	unsigned long ciclosFinPeticion[MAXPETICIONES];
	// Si se eligiera la opcion de trabajar con tiempos unicamente
	// float tinicio[MAXPETICIONES];
	// float tfinal[MAXPETICIONES];
};

datos datoHilo[MAXUSUARIOS];

// ---------------------------------------
double MilisegundosTranscurridos(LARGE_INTEGER inicio, LARGE_INTEGER final) {
	LARGE_INTEGER diferencia;
	float milisegundos;

	diferencia.QuadPart = final.QuadPart - inicio.QuadPart;
	milisegundos = diferencia.LowPart / ticksPorMilisegundo;
	if (diferencia.HighPart != 0)
		milisegundos += (float)ldexp((double)diferencia.HighPart, 32) / ticksPorMilisegundo;

	return milisegundos;
}
//--------------------------------------------------------------------------
float NumeroAleatorio(float limiteInferior, float limiteSuperior) {
	float num = (float)rand();
	num = num * (limiteSuperior - limiteInferior) / RAND_MAX;
	num += limiteInferior;
	return num;
}
//--------------------------------------------------------------------------
float DistribucionExponencial(float media) {
	float numAleatorio = NumeroAleatorio(0, 1);
	while (numAleatorio == 0 || numAleatorio == 1)
		numAleatorio = NumeroAleatorio(0, 1);
	return (-media) * logf(numAleatorio);
}
//--------------------------------------------------------------------------
// Funcion preparada para ser un thread. Simula un usuario
DWORD WINAPI Usuario(LPVOID parametro) {
	DWORD dwResult = 0;
	int numHilo = *((int*)parametro);

	SOCKET elSocket;
	sockaddr_in dirServidor;
	char peticion[TAM_PET];
	char respuesta[TAM_RES];
	int valorRetorno;
	LARGE_INTEGER tIni, tFin;
	float tmpReflex;


	datoHilo[numHilo].contPet = 0;

	srand(127 + numHilo * 5);

	do {

		//
		// Creacion del socket
		//
		elSocket = socket(AF_INET, SOCK_STREAM, 0);
		if (elSocket == INVALID_SOCKET) {
			cerr << "No se pudo crear el socket" << endl;
			WSACleanup();
			exit(EXIT_FAILURE);
		}

		//
		// Toma de tiempo inicial
```

```
		//
		QueryPerformanceCounter(&tIni);
		//
		// Conexion con el servidor
		//

		dirServidor.sin_family = AF_INET;
		dirServidor.sin_addr.s_addr = inet_addr(ipServidor);
		dirServidor.sin_port = htons(PUERTO + numHilo);
		valorRetorno = connect(elSocket, (struct sockaddr*) & dirServidor, sizeof(dirServidor));
		if (valorRetorno == SOCKET_ERROR) {
			cerr << "Error en el connect: " << WSAGetLastError() << endl;
			closesocket(elSocket);
			WSACleanup();
			exit(EXIT_FAILURE);
		}

		//
		// Enviar una cadena
		//
		valorRetorno = send(elSocket, peticion, sizeof(peticion), 0);
		if (valorRetorno == SOCKET_ERROR) {
			cerr << "Error en el send: " << WSAGetLastError() << endl;
			closesocket(elSocket);
			WSACleanup();
			exit(EXIT_FAILURE);
		}

		//
		// Recibir la respuesta
		//
		valorRetorno = recv(elSocket, respuesta, sizeof(respuesta), 0);
		if (valorRetorno != TAM_RES) {
			cerr << "Error en el recv: " << WSAGetLastError() << endl;
			closesocket(elSocket);
			WSACleanup();
			exit(EXIT_FAILURE);
		}


		//
		// Cerrar la conexion
		//

		closesocket(elSocket);

		//
		// Medicion final
		//

		QueryPerformanceCounter(&tFin);

		//
		// Comprobar si peticion valida
		//

		tmpReflex = DistribucionExponencial((float)tReflex);

		if (tIni.QuadPart > tickInicio.QuadPart&& tFin.QuadPart < tickFin.QuadPart) {

			// La implementacion de esta parte puede variar
			// dependiendo de la opcion elegida para almacenar
			// valores. Tiempos de inicio y fin, o ciclos de inicio y fin

			datoHilo[numHilo].tres[datoHilo[numHilo].contPet] =
(float)MilisegundosTranscurridos(tIni, tFin);
			datoHilo[numHilo].reflex[datoHilo[numHilo].contPet] = tmpReflex;
			datoHilo[numHilo].ciclosIniPeticion[datoHilo[numHilo].contPet] =
(unsigned)(tIni.QuadPart - tickBase.QuadPart);
			datoHilo[numHilo].ciclosFinPeticion[datoHilo[numHilo].contPet] =
(unsigned)(tFin.QuadPart - tickBase.QuadPart);
			datoHilo[numHilo].contPet++;
			if (datoHilo[numHilo].contPet > MAXPETICIONES) {
				printf("Superado el limite de peticiones para un hilo \n");
				exit(0);
			}
		}
```

```
                Sleep((int)(tmpReflex * 1000));

        } while (tFin.QuadPart < tickFin.QuadPart);


        return dwResult;
}

//--------------------------------------------------------------------
// Funci□n para cargar la libreria de sockets
int Ini_sockets(void) {
        WORD wVersionDeseada;
        WSAData wsaData;

        int error;

        wVersionDeseada = MAKEWORD(2, 0);
        if (error = WSAStartup(wVersionDeseada, &wsaData) != 0) {
                return error;
        }

        // Comprobar si la DLL soporta la versi□n 2.0

        if (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 0) {
                error = 27;
                cerr << "La libreria no soporta la version 2.0" << endl;
                WSACleanup();
        }
        return error;
}
//--------------------------------------------------------------------
// Funci□n para descargar la librer□a de sockets
void Fin_sockets(void) {
        WSACleanup();
}
//--------------------------------------------------------------------

int main(int argc, char* argv[])
{
        int i, j;
        HANDLE handleThread[MAXUSUARIOS];
        int parametro[MAXUSUARIOS];
        int segCal; // Segundos de calentamiento
        int segMed; // Segundos de medicion
        FILE* info;

        // Variables para calculos de tiempo de respuesta y productividad
        float sumaTiempos;
        float sumaTiempos2;
        float taux1, taux2; // variables auxiliares para calcular tiempo de respuesta 2
        int sumaPet;


        time_t hora_ini_exp; //Marca la hora de inicio del experimento
        time_t hora_inicio_medicion; //Hora inicio de la medicion
        time_t hora_fin_medicion; //Hora fin de la medicion


        if (argc != 6) {
                printf("Numero de parametros no valido: inyector usuarios TReflex(seg) IPservidor
Calentamiento(seg) Medidicon(seg) \n");
                exit(0);
        }
        numUsuarios = atoi(argv[1]);
        if (numUsuarios <= 0 || numUsuarios > MAXUSUARIOS) {
                printf("El numero de usuarios debe estar comprendido entre 1 y %d\n", MAXUSUARIOS);
                exit(0);
        }

        tReflex = (float)atof(argv[2]);
        if (tReflex <= 0) {
                printf("El tiempo de reflexion debe ser mayor que 0\n");
                exit(0);
        }

        ipServidor = argv[3];
```

```
	segCal = atoi(argv[4]);
	if (segCal < 0) {
		printf("El tiempo de transitorio debe ser mayor o igual que 0\n");
		exit(0);
	}

	segMed = atoi(argv[5]);
	if (segMed <= 0) {
		printf("El tiempo de medicion debe ser mayor que 0\n");
		exit(0);
	}

	Ini_sockets();

	// Calcular hora de inicio de la medicion y hora final de la medicion
	time(&hora_ini_exp);
	hora_inicio_medicion = hora_ini_exp + segCal;
	hora_fin_medicion = hora_ini_exp + segCal + segMed;


	// Preparar la medicion de tiempos
	LARGE_INTEGER ticksPorSeg;
	if (!QueryPerformanceFrequency(&ticksPorSeg)) {
		cout << "No esta disponible el contador de alto rendimiento" << endl;
		exit(-1);
	}
	ticksPorMilisegundo = (float)(ticksPorSeg.LowPart / 1E3);

	QueryPerformanceCounter(&tickBase);
	tickInicio.QuadPart = tickBase.QuadPart + (LONGLONG)(segCal * 1000 * ticksPorMilisegundo);
	tickFin.QuadPart = tickInicio.QuadPart + (LONGLONG)(segMed * 1000 * ticksPorMilisegundo);

	// Lanzar los hilos
	//!!!!!!!!!!!!!!!!!!!!!!!!! COMPLETAR CON LA PRACTICA 1 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!
		for (i = 0; i < numUsuarios; i++) {
			parametro[i] = i;
			handleThread[i] = CreateThread(NULL, 0, Usuario, &parametro[i], 0, NULL);
			if (handleThread[i] == NULL) {
				cerr << "Error al lanzar el hilo" << endl;
				exit(EXIT_FAILURE);
			}
		}

		// Hacer que el Thread principal espere por sus hijo
		//!!!!!!!!!!!!!!!!!!!!!!!!! COMPLETAR CON LA PRACTICA 1 //!!!!!!!!!!!!!!!!!!!!!!!!!!
		for (i = 0; i < numUsuarios; i++)
			WaitForSingleObject(handleThread[i], INFINITE);
		Fin_sockets();



	// Escribir los resultados a disco

	fopen_s(&info, "info.txt", "w");

	//TIEMPOS

	char cadena[26];

	ctime_s(cadena, sizeof(cadena), &hora_inicio_medicion);

	//Almacenar en el fichero la hora de inicio de la medicion y la hora de fin de la medicion.
	fprintf(info, "\nHORA INICIO DE MEDICION:  %s\n", cadena);

	ctime_s(cadena, sizeof(cadena), &hora_fin_medicion);

	fprintf(info, "HORA FIN DE MEDICION: %s \n", cadena);


	// Parametros de la prueba

	fprintf(info, "\nParametros del experimento: \n");
	fprintf(info, "Nº Usuarios: %d; Tpo. Reflex (seg): %f; IP servidor: %s; Transitorio (seg): %d; Medicion
(seg): %d \n", numUsuarios, tReflex, ipServidor, segCal, segMed);

	printf("\nParametros del experimento: \n");
	printf("Usuarios: %d; Tpo. Reflex (seg); %f; IP servidor: %s; Transitorio (seg): %d; Medicion (seg): %d \n",
numUsuarios, tReflex, ipServidor, segCal, segMed);
```

```
	fprintf(info, "N. usu.; N. pet.; Tpo Reflex(Seg); Tpo. Ini.(mseg); Tpo. Fin(mseg)\n");

	sumaTiempos = 0;
	sumaTiempos2 = 0;
	sumaPet = 0;

	for (i = 0; i < numUsuarios; i++) {
		sumaPet = sumaPet + datoHilo[i].contPet;
		for (j = 0; j < datoHilo[i].contPet; j++) {
			sumaTiempos = sumaTiempos + datoHilo[i].tres[j];
			taux1 = datoHilo[i].ciclosIniPeticion[j] / ticksPorMilisegundo;
			taux2 = datoHilo[i].ciclosFinPeticion[j] / ticksPorMilisegundo;
			sumaTiempos2 = sumaTiempos2 + (taux2 - taux1);
			fprintf(info, "%d;%d;%f;%f;%f\n", i, j, datoHilo[i].reflex[j], taux1, taux2);
		}
	}
	fprintf(info, "\n\n");

	printf("Resultados:\n");
	printf("N. Pet: %d \n", sumaPet);
	printf("Seg.Med: %d \n", segMed);
	printf("Tpo.Res1(mseg) : %f\n", (float)sumaTiempos / sumaPet);
	printf("Tpo.Res2(mseg) : %f\n", (float)sumaTiempos2 / sumaPet);
	printf("Product. (pet/seg): %f\n", (float)sumaPet / segMed);

	fclose(info);

	return 0;
}
```