PL3	2	Santos Gómez Iglesias Manzano	Pablo Pelayo
Nº PLo	Equipo	Apellidos	Nombre

71976794-L 32893351-Q	UO290260@uniovi.es UO266600@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 2024-2025

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

Contenido

Introducción

En esta práctica aprenderemos los conceptos de medición y visualización. utilizando parte del inyector anteriormente creado y con el monitor que incorpora Windows. Para ello tendremos que medir el tiempo de inicio de cada prueba para posteriormente contrastar con el tiempo de inicio de los resultados del monitor y así poder representarlo gráficamente.

Código fuente del programa

```
#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];
```

```
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);
}

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 versión 2.0

	if (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 0) {
		error = 27;
		cerr << "La librería no soporta la versión 2.0" << endl;
		WSACleanup();
	}
	return error;
}

// Función para descargar la librería de sockets
void Fin_sockets(void) {
	WSACleanup();
}

// ----------------------------------------
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;
}
//----------------------------------------


// 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(53 + 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((unsigned int)(tmpReflex * 1000));

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


        return dwResult;
}

//---------------------------------------------------------------------


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) Mididicon(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

	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 hijos

		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;
}
```

Cuestiones

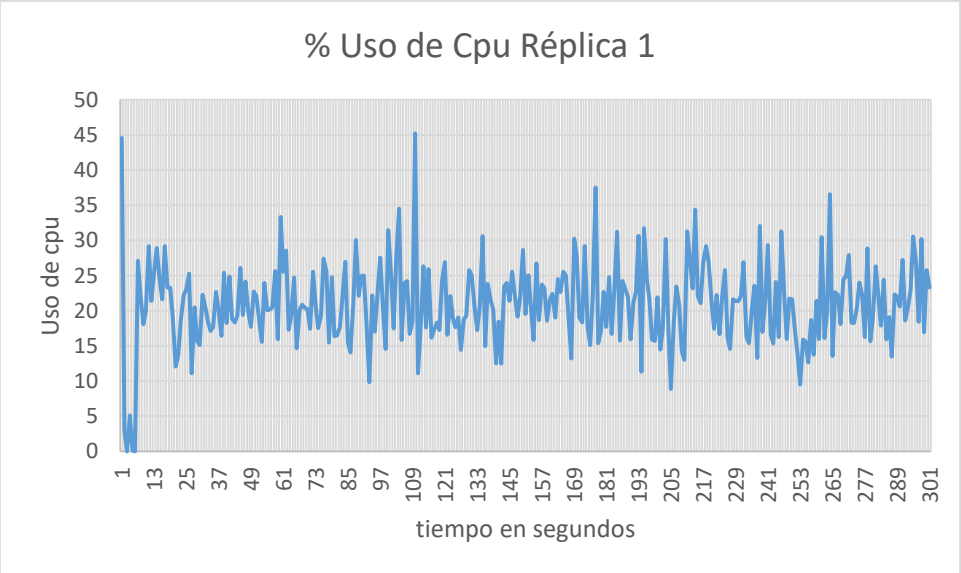
¿Cuál sería el límite de la productividad máxima alcanzable en cualquiera de los experimentos realizados, en condiciones ideales? ¿Por qué?

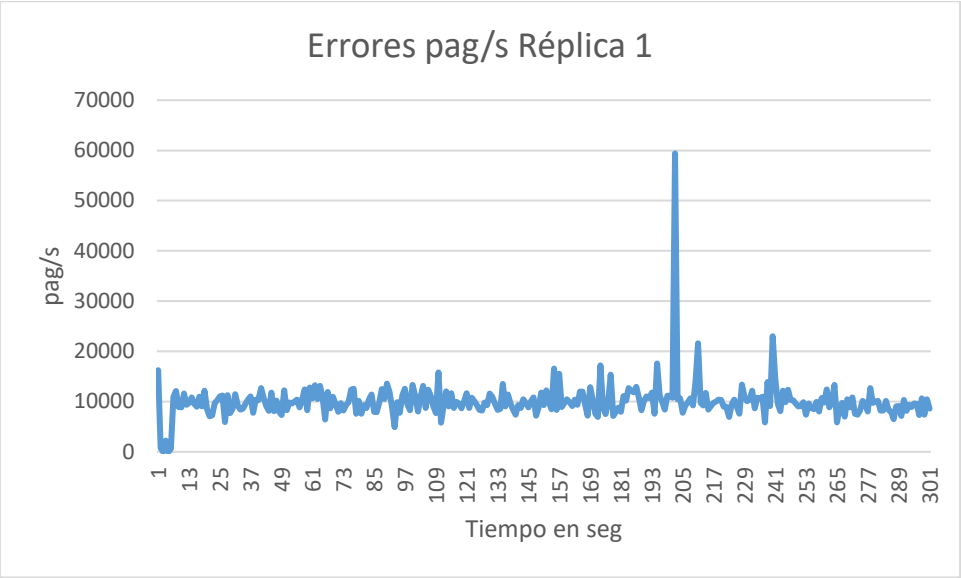
Para calcular la productividad total teórica se debe suponer que el tiempo de respuesta es 0 aplicando el tiempo de reflexión de cada una de las peticiones y el número de usuarios. En este ejemplo el número de usuarios es 60 y el tiempo de reflexión 1,5 s. Si cada usuario lanza una petición con un tiempo de respuesta de 1,5 entre cada petición el numero de peticiones por segundo sería de 40 tareas/seg en condiciones ideales.

$$\text{Producción máxima teórica} = \frac{\text{número de usuarios}}{\text{Tiempo de reflexión (s)}} = \frac{60 \text{ usuarios}}{1{,}5 \text{ s}} = 40 \text{ peticiones/s}$$

¿Cuál debería ser la duración del intervalo de arranque para que no influya en el experimento?

Desde el punto de vista teórico el intervalo de arranque debería ser igual o menor al tiempo en el que se lanza la primera petición, ya que sino esta misma quedaría descartada.

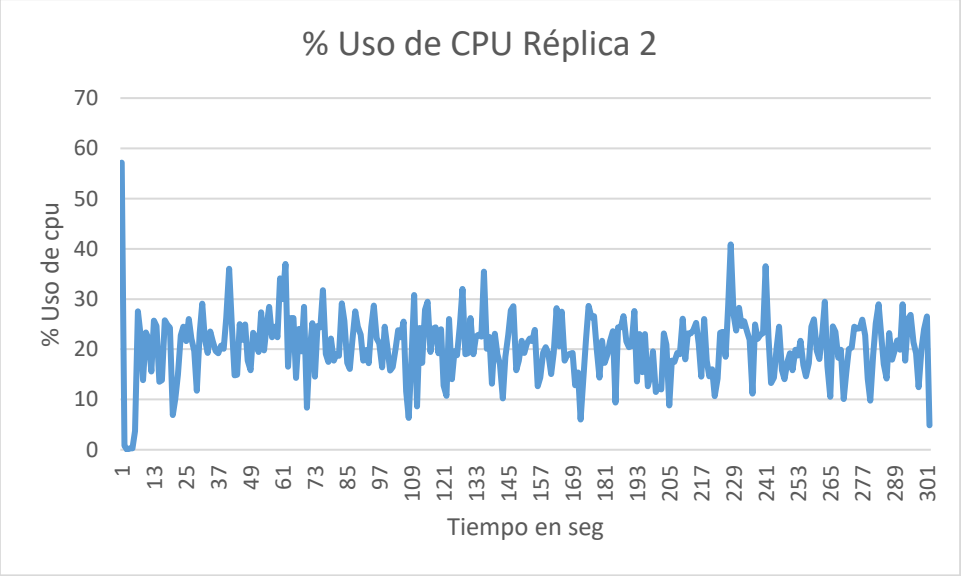




Gráficas realizadas con la primera réplica

- Primera gráfica se ha utilizado el % uso del procesador
- Segunda gráfica errores de página / s

En ambos resultados se produce un aumento nada más comenzar para una posterior caída, y luego estabilizarse. Si analizamos los datos en el Excel esta caída se produce entorno al primer segundo de lanzar la primera petición donde los valores se mantienen bajos y pasados 7 segundos se estabiliza.



Gráficas realizadas con la segunda réplica

- Gráfica se ha utilizado el % uso del procesador

Al igual que la gráfica anterior con la única diferencia de que se estabiliza cerca de unos 5s y al terminar se produce una caída en el % uso del procesador.

Conclusión: se debería estimar un intervalo de arranque de aproximadamente 5 s.

¿Hay diferencias significativas entre las tres réplicas del experimento?
No en todas las gráficas se produce un aumento para una posterior caída y se estabilizan en un rango entre [5s – 7s] y al final al terminar se vuelve a producir una bajada, la cual es meramente normal.

Tabla

Concepto	Réplica 1	Réplica 2	Réplica 3
Media del tiempo de reflexión (s)	1,53269603	1,53235624	1,53242167
Media del tiempo de respuesta (mseg)	49,931328	45,7253464	45,2676833
Productividad promedio (peticiones / s)	37,9	38,00333333	38,03333333
Media del tiempo inactivo del disco (%)	40,88598081	71,62652255	71,48192416
Media de la longitud de la cola de disco	1,384423425	0,338125928	0,34194486
Media del número de transferencia de s	905,8991412	795,5639962	784,5690711
Media del promedio en seg/transferencia	0,001572377	0,000421138	0,000431757
Media de los bytes de caché (bytes)	43183461,21	44130032,74	44702043
Media de los bytes disponibles (bytes)	15843140618	15854460345	15852238052
Media de contador Errores de pag/s (pag /s)	10021,24739	8355,769328	8265,942439
Media de contador pag/s (pags/ s)	20,95786984	117,8720894	34,28883175
Media de % de procesador (%)	210376,0197	20,55399229	20,57762552
Media de total de bytes/s (bytes/s)	2626637846	129616,38	131636,0172
Ancho de banda actual (bystes/s)	2500000000	2500000000	2500000000

Conclusión

Con esta práctica logramos entender mejor los conceptos de medición y visualización utilizando herramientas de Windows, como también a relacionar datos del monitor con la hora en la que se lanzaron las peticiones y a comprender mejor la información proporcionada por las gráficas. Gracias a ello logramos comprender los distintos factores que pueden limitar el rendimiento del sistema.