|  |  |  |  |
| --- | --- | --- | --- |
| **PL4** | **1** | Marques RamosFrancisco Mier Montoto | **Marcel**  **Juan** |
| Nº PL | Equipo | Apellidos | Nombre |

|  |  |
| --- | --- |
| **35625337-Q**  **71777658-V** | **UO289464@uniovi.es**  **UO283319@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. Introducción 3

2. Preguntas 3

3. Tabla de mediciones 6

4. Conclusión 7

5. Código fuente 7

# Introducción

Este documento analiza los resultados de la segunda práctica, que trata sobre la instrumentación del inyector de carga. Las mediciones se han realizado en los ordenadores del aula, utilizando dos ordenadores: uno como servidor, y otro como cliente. El cliente utilizó el código incluido en este informe, mientras que el servidor utilizó el ejecutable proporcionado por los profesores de la asignatura.

# Preguntas

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

La productividad es el número de tareas partido por el tiempo que se tarda en realizar dichas tareas. Para obtener el límite de productividad máxima alcanzable por nuestro sistema, hay que modificar la fórmula para obtener datos que se conozcan:

Puesto que el número de tareas total es la cantidad de peticiones que se envían, se obtiene la siguiente fórmula:

En esta versión del programa cliente, la cantidad de peticiones es la cantidad máxima que se puedan transmitir hasta el fin del tiempo de medición:

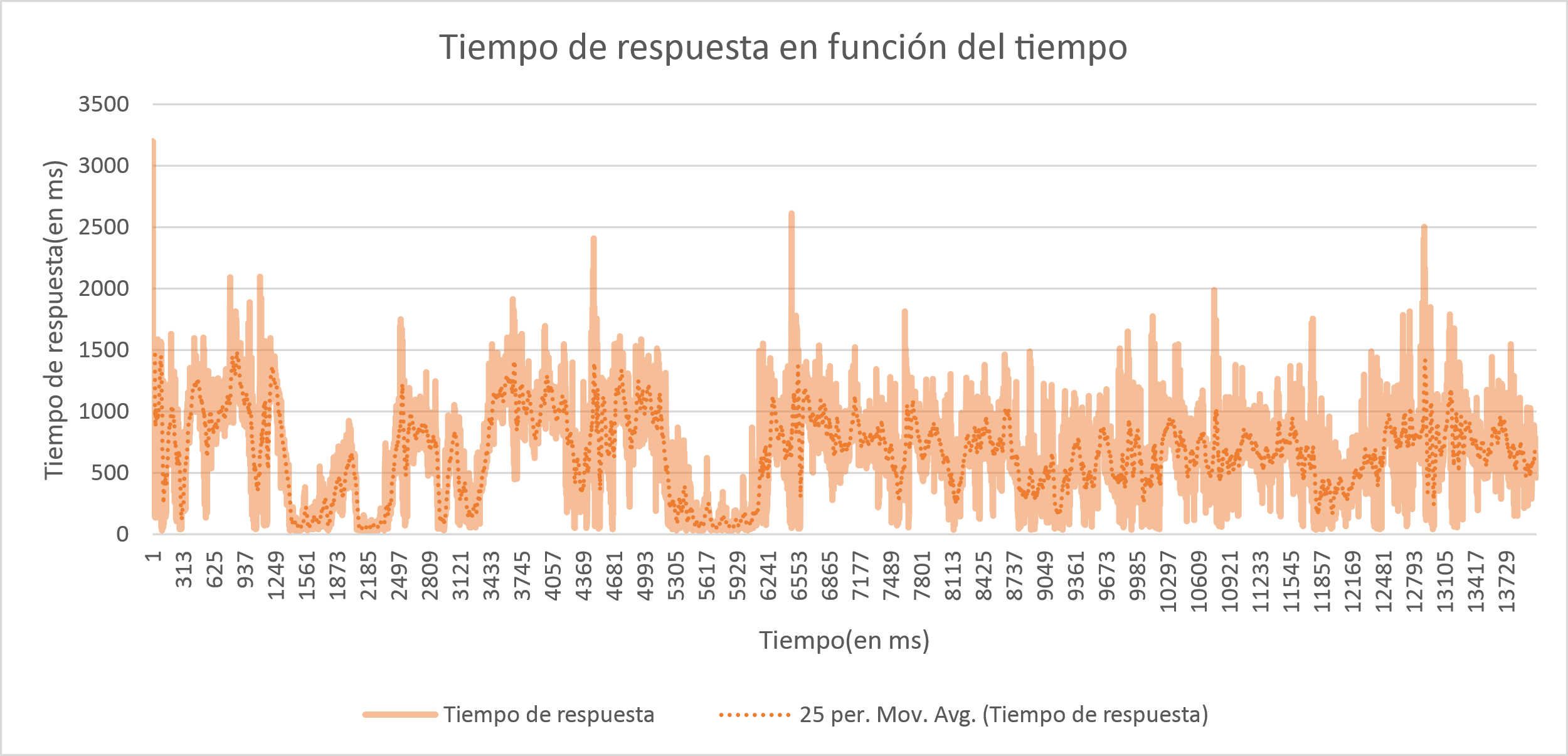
Asumiendo que el tiempo de respuesta del servidor es 0 puesto que hay conexión directa entre servidor y cliente, y que se puede simplificar la fórmula utilizando el tiempo total, se obtiene lo siguiente:

Puesto que se utilizan 90 usuarios y el tiempo de reflexión es 1.25 segundos, **la productividad máxima alcanzable por el programa es 72.**

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

Determinar el transitorio de arranque no es una tarea sencilla, puesto que las mediciones y las gráficas varían mucho entre medición y medición. Al utilizar los tiempos de respuesta para medirlo, tan solo podemos obtener un resultado significativo en la segunda medición, donde durante el primer segundo y medio las peticiones tardan más de lo normal, lo que se podría achacar al transitorio inicial. Sin embargo, en el resto de mediciones no hay un periodo obvio donde el tiempo de respuesta sea significativamente grande, como demuestran las gráficas de tiempo de respuesta de cada una.

En caso de tomar el transitorio inicial de la segunda medición, serían 1.5 segundos.

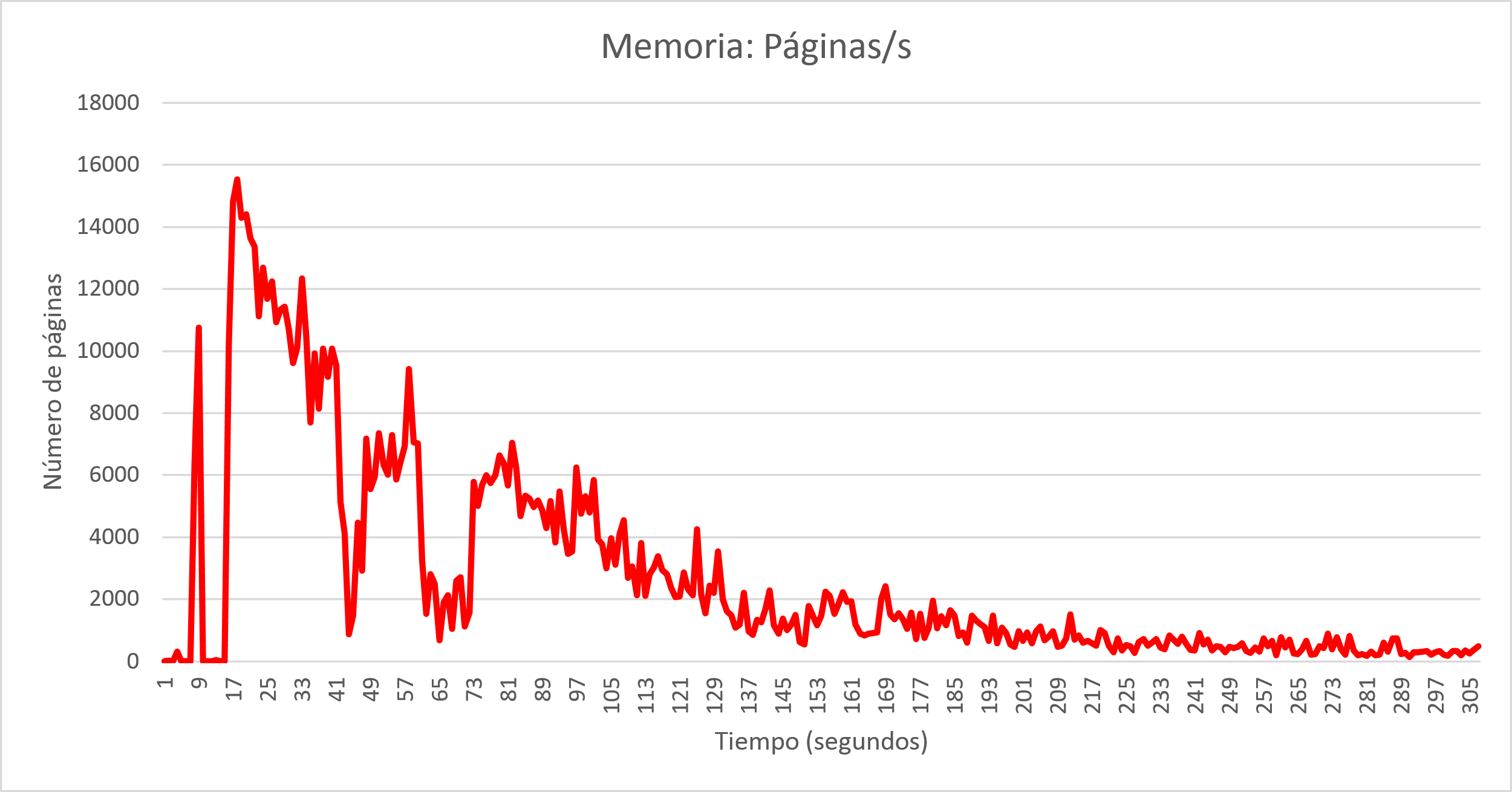


1. Tiempo de respuesta de la segunda medición, que incluye un transitorio inicial.

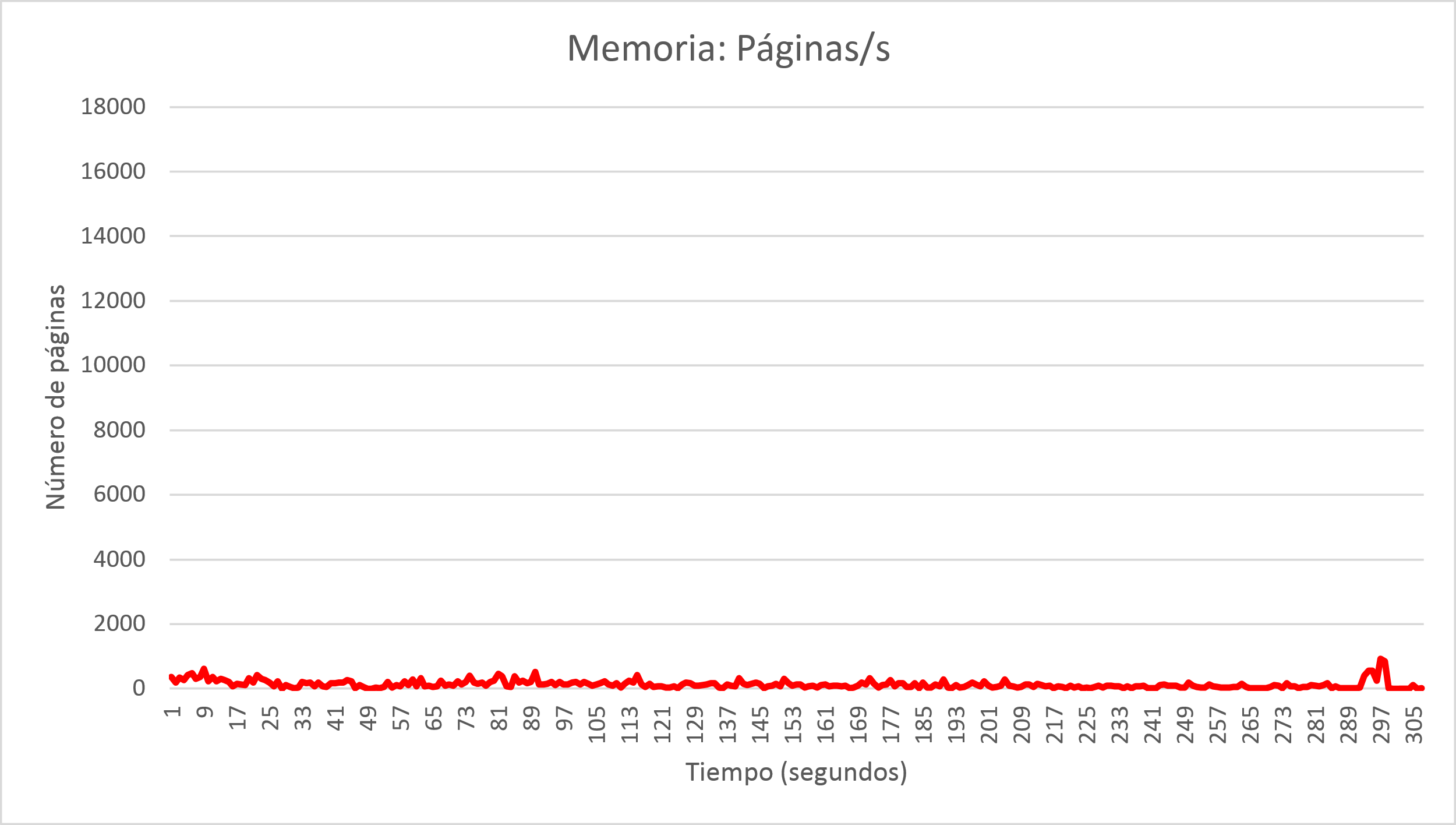
3**. ¿Hay diferencias significativas entre las tres réplicas del experimento?**

Hay muchas diferencias entre las tres réplicas del experimento. De hecho, es coherente tan solo por unos pocos detalles, como por ejemplo los valores similares de productividad.

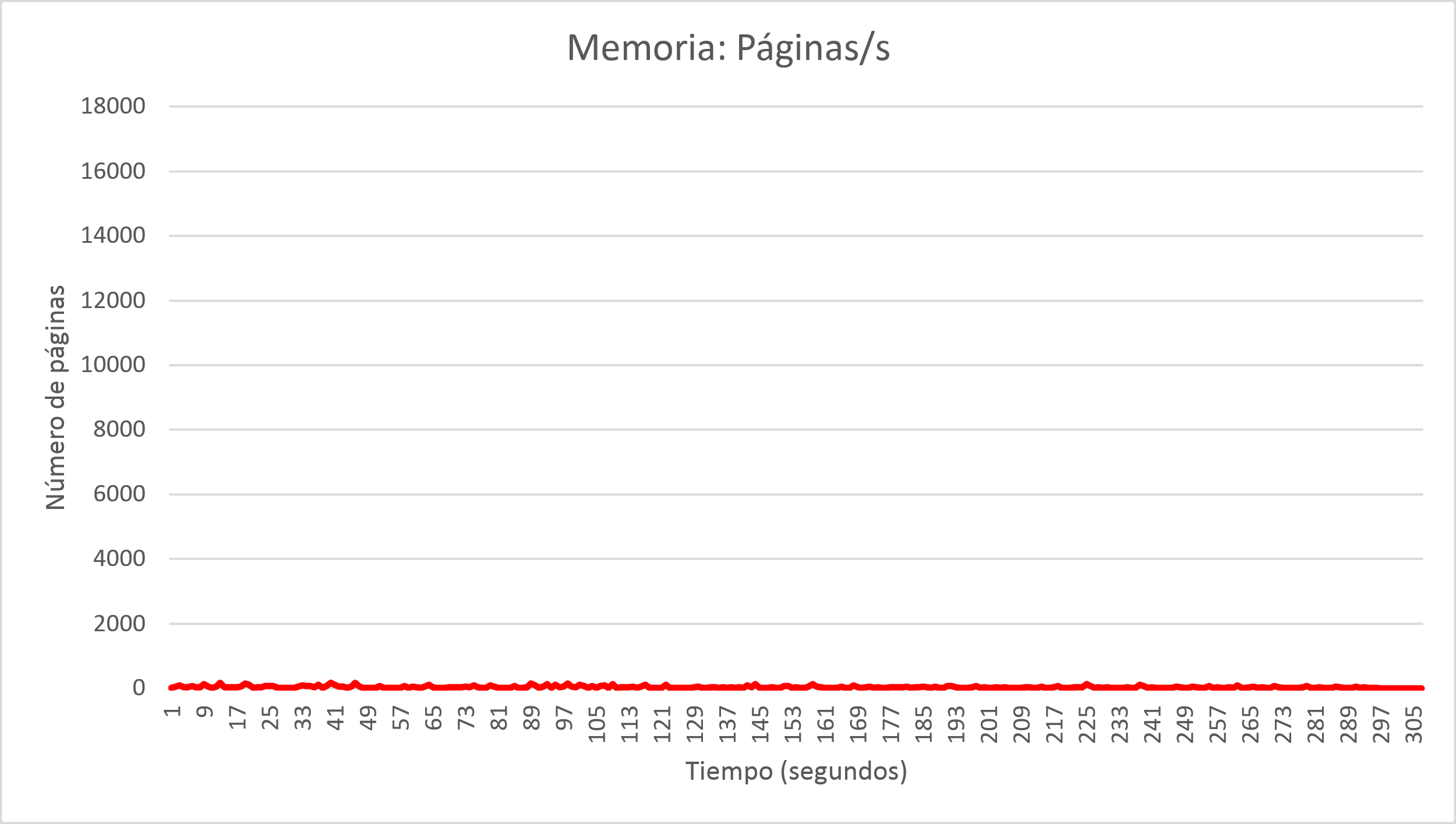
Utilizando las gráficas de la memoria de páginas por segundo, podemos apreciar grandes diferencias entre mediciones:



2. Gráfica de páginas por segundo de la primera medición.



3. Gráfica de páginas por segundo de la segunda medición.



4. Gráfica de páginas por segundo de la tercera medición.

Otras gráficas, como el porcentaje de tiempo inactivo de disco también reflejan diferencias evidentes. Estas gráficas y todos los datos recogidos se pueden encontrar en el fichero Excel entregado junto a este documento.

# Tabla de mediciones

|  |  |  |  |
| --- | --- | --- | --- |
|  | **RÉPLICA 1** | **RÉPLICA 2** | **RÉPLICA 3** |
| **Media del tiempo de reflexión** | 1,116971227 | 1,14165398 | 1,1147494 |
| **Media del tiempo de respuesta** | 684,1918556 | 658,3853249 | 670,925267 |
| **Productividad Promedio** | 49,70520583 | 49,5067912 | 49,86296307 |
| **Media del tiempo inactivo del disco** | 47,40613185 | 59,83596729 | 60,9572345 |
| **Media la longitud de la cola del disco** | 1,277299936 | 0,693902908 | 0,66280855 |
| **Media del número de transferencias seg/transferencia** | 831,1862043 | 749,4205561 | 738,235551 |
| **Media los bytes de caché** | 50006614,39 | 50661590,23 | 52601945,8 |
| **Media de los bytes disponibles** | 16044074106 | 16044645183 | 1,60E+10 |
| **Media del contador de Errores de página/s** | 17816,77717 | 16267,69558 | 15350,2691 |
| **Media del contador de pág./s** | 1604,36516 | 337,0005899 | 31,394807 |
| **Media del % de procesador** | 90,3695087 | 90,69027564 | 92,2746154 |
| **Media del Total de byte/s** | 202350,0645 | 204921,4996 | 209109,43 |
| **Ancho de banda actual** | 1000000000 | 1000000000 | 1000000000 |

# Conclusión

Los resultados de las mediciones dependen de la cantidad de veces que se repite el proceso. Además, analizando la productividad en cada una de las mediciones, podemos ver que el valor obtenido es similar entre las tres, por lo que, pese a que existan mediciones de recursos muy diferentes entre los tres intentos, la cantidad de tareas no se ve afectada.

No se ha podido detectar evidencia definitiva de la existencia de un valor de transitorio inicial fiable, por lo que puede haber ocurrido un error de procedimiento al guardar los datos.

# Código fuente

// Juan Mier (UO283319) '22

#include <windows.h> // DWORD, HANDLE, WINAPI

#include <iostream> // cout, cerr, endl

#include <fstream> // save results to file

#include <time.h> // time, time\_t, QueryPerformance

#include <math.h> // HighPart, LowPart, QuadPart

using namespace std; // this removes the need of writing "std::"" every time.

constexpr unsigned int MAXUSERS = 1000;

constexpr unsigned int MAXPETITIONS = 1000;

constexpr unsigned int PORT = 57000;

constexpr unsigned int PETITION\_SIZE = 1250;

constexpr unsigned int RESPONSE\_SIZE = 1250;

constexpr auto SERVERIP = "192.168.203.231"; // 192.168.203.231

unsigned int totalUsers;

float reflexTime;

float ticksPerMs;

LARGE\_INTEGER tickBase;

LARGE\_INTEGER tickStart;

LARGE\_INTEGER tickEnd;

typedef struct {

int petitionCounter;

float reflex[MAXPETITIONS];

float responseTime[MAXPETITIONS];

unsigned long ciclosIniPeticion[MAXPETITIONS];

unsigned long ciclosFinPeticion[MAXPETITIONS];

} threadParams;

threadParams threadInfo[MAXUSERS];

// ---

float GenerateRandomFloat(float lowerLimit, float upperLimit) {

float num = (float)rand();

num = num \* (upperLimit - lowerLimit) / RAND\_MAX;

num += lowerLimit;

return num;

}

float GenerateExponentialDistribution(float average) {

float num = GenerateRandomFloat(0, 1);

while (num == 0 || num == 1) {

num = GenerateRandomFloat(0, 1);

}

return (-average) \* logf(num);

}

void errorMessage(string message) {

cerr << message << endl;

exit(EXIT\_FAILURE);

}

double TimeDiff(LARGE\_INTEGER start, LARGE\_INTEGER end) {

LARGE\_INTEGER diff;

diff.QuadPart = end.QuadPart - start.QuadPart;

float ms = diff.LowPart / ticksPerMs;

if (diff.HighPart != 0) {

ms += (float)ldexp((double)diff.HighPart, 32) / ticksPerMs;

}

return ms;

}

// ---

DWORD WINAPI Usuario(LPVOID parameter) {

int threadNum = \*((int\*) parameter);

char petition[PETITION\_SIZE];

char response[RESPONSE\_SIZE];

LARGE\_INTEGER timeStart, timeEnd;

threadInfo[threadNum].petitionCounter = 0;

do {

//printf("[DEBUG] Peticion: %d, usuario: %d\n", i, numHilo);

SOCKET s = socket(AF\_INET, SOCK\_STREAM, 0);

if (s == INVALID\_SOCKET) {

WSACleanup();

errorMessage("Ha ocurrido un error al inicializar el socket.");

}

QueryPerformanceCounter(&timeStart);

sockaddr\_in serv;

serv.sin\_family = AF\_INET;

serv.sin\_addr.s\_addr = inet\_addr(SERVERIP);

serv.sin\_port = htons(PORT + threadNum);

// connect

if (connect(s, (struct sockaddr\*) & serv, sizeof(serv)) == SOCKET\_ERROR) {

WSACleanup();

errorMessage("Error al conectar al servidor.");

}

// send

if (send(s, petition, sizeof(petition), 0) == SOCKET\_ERROR) {

WSACleanup();

errorMessage("Error al enviar una cadena.");

}

// receive

if (recv(s, response, sizeof(response), 0) != RESPONSE\_SIZE) {

WSACleanup();

errorMessage("Error al recibir la respuesta.");

}

// close

if (closesocket(s) != 0) {

WSACleanup();

errorMessage("Error al cerrar el socket.");

}

float tiempo = GenerateExponentialDistribution((float)reflexTime);

QueryPerformanceCounter(&timeEnd);

if (timeStart.QuadPart > tickStart.QuadPart&& timeEnd.QuadPart < tickEnd.QuadPart) {

threadInfo[threadNum].responseTime[threadInfo[threadNum].petitionCounter] = (float)TimeDiff(timeStart, timeEnd);

threadInfo[threadNum].reflex[threadInfo[threadNum].petitionCounter] = tiempo;

threadInfo[threadNum].ciclosIniPeticion[threadInfo[threadNum].petitionCounter] = (unsigned)(timeStart.QuadPart - tickBase.QuadPart);

threadInfo[threadNum].ciclosFinPeticion[threadInfo[threadNum].petitionCounter] = (unsigned)(timeEnd.QuadPart - tickBase.QuadPart);

threadInfo[threadNum].petitionCounter++;

if (threadInfo[threadNum].petitionCounter > MAXPETITIONS) {

errorMessage("Superado el limite de peticiones por hilo.");

}

}

Sleep((int) (tiempo\*1000));

} while (timeStart.QuadPart < tickEnd.QuadPart);

return 0;

}

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

HANDLE handleThread[MAXUSERS];

int parametro[MAXUSERS];

int segCal;

int segMed;

time\_t timeIniExp;

time\_t timeIniMed;

time\_t timeFinMed;

if (argc != 5) {

cout << "Introducir num. usuarios: ";

cin >> totalUsers;

cout << "Tiempo de reflexion despues de cada peticion: ";

cin >> reflexTime;

cout << "Introduzca el tiempo de calentamiento (en segundos): ";

cin >> segCal;

cout << "Introduzca el tiempo de medición (en segundos): ";

cin >> segMed;

}

else {

totalUsers = atoi(argv[1]);

reflexTime = atof(argv[2]);

segCal = atoi(argv[3]);

segMed = atoi(argv[4]);

cout << "Num. usuarios: " << totalUsers << endl;

cout << "Tiempo de reflexion: " << reflexTime << endl;

cout << "Tiempo de calentamiento: " << segCal << endl;

cout << "Tiempo de medición: " << segMed << endl;

}

cout << "Utilizando IP " << SERVERIP << endl;

if (totalUsers > MAXUSERS ||

totalUsers <= 0 || reflexTime <= 0 ||

segCal < 0 || segMed <= 0) {

errorMessage("Arumentos invalidos.");

}

time(&timeIniExp);

timeIniMed = timeIniExp + segCal;

timeFinMed = timeIniMed + segMed;

LARGE\_INTEGER ticksPerSec;

if (!QueryPerformanceFrequency(&ticksPerSec)) {

cout << "No esta disponible el contador de alto rendimiento." << endl;

exit(EXIT\_FAILURE);

}

ticksPerMs = (float)(ticksPerSec.LowPart / 1E3);

QueryPerformanceCounter(&tickBase);

tickStart.QuadPart = tickBase.QuadPart + (LONGLONG)(segCal \* 1000 \* ticksPerMs);

tickEnd.QuadPart = tickStart.QuadPart + (LONGLONG)(segMed \* 1000 \* ticksPerMs);

// init socket connection

WORD wVersionRequested = MAKEWORD(2, 0);

WSAData wsaData;

if (WSAStartup(wVersionRequested, &wsaData) != 0) {

errorMessage("Ha ocurrido un error al inicializar el uso de sockets.");

}

if (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 0) {

errorMessage("La liberia no soporta la version 2.0");

}

cout << "Transmitiendo...";

for (int i = 0; i < totalUsers; i++) {

parametro[i] = i;

handleThread[i] = CreateThread(NULL, 0, Usuario, &parametro[i], 0, NULL);

if (handleThread[i] == NULL) {

errorMessage("Error al lanzar el hilo.");

}

}

for (int i = 0; i < totalUsers; i++) { // el hilo principal espera por sus hijos.

WaitForSingleObject(handleThread[i], INFINITE);

}

WSACleanup();

ofstream output("output.csv");

output << "User,Petition,Reflex,Tstart,Tend";

float responseTime = 0;

float responseTime2 = 0;

float taux1, taux2;

int totalPetitions = 0;

for (int i = 0; i < totalUsers; i++) {

totalPetitions += threadInfo[i].petitionCounter;

for (int j = 0; j < threadInfo[i].petitionCounter; j++) {

auto reflex = threadInfo[i].reflex[j];

responseTime += threadInfo[i].responseTime[j];

taux1 = threadInfo[i].ciclosIniPeticion[j] / ticksPerMs;

taux2 = threadInfo[i].ciclosFinPeticion[j] / ticksPerMs;

responseTime2 += taux2 - taux1;

output << i << "," << j << "," << reflex << "," << taux1 << "," << taux2 << endl;

}

}

cout << endl << "RESULTADOS:" << endl;

cout << "Num. peticiones: " << totalPetitions << endl;

cout << "Segmento de medicion: " << segMed << endl;

cout << "Tiempo de respuesta 1: " << (float)responseTime / ticksPerMs << endl;

cout << "Tiempo de respuesta 2: " << (float)responseTime2 / ticksPerMs << endl;

cout << "Productividad: " << (float)totalPetitions / segMed << endl;

output.close();

}