İşletim Sistemleri
Proje Ödevi
Raporu

İsim/Soy isim: Emre Okçelen
Öğrenci Numarası:170421929
Bilgisayar Mühendisliği
emreokcelen03@gmail.com

İsim/Soy isim: Karun Acar
Öğrenci Numarası:170421049
Bilgisayar Mühendisliği
acarkarun@gmail.com

1.0 Part 1 Kod Gösterimi (Findtopk)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <string.h>

#define MAX_SIZE 10000

void deleteTemporaryFiles() {
    for (int i = 1; i <= MAX_SIZE; i++) {
        char file_name[20];
        sprintf(file_name, "intermediate%d.txt", i);
        remove(file_name);
    }

    printf("Temporary files deleted.\n");
}

int compare(const void* a, const void* b) {
    return (*(int*)b - *(int*)a);
}

void writeRandomNumbersToFile(const char* fileName, int count) {
    FILE* file = fopen(fileName, "w");
    if (file == NULL) {
        perror("File opening error");
        exit(EXIT_FAILURE);
    }
```

```
    int i, *random_numbers;
    random_numbers = (int *)malloc(count * sizeof(int));

    srand((unsigned int)getpid());

    for (i = 0; i < count; i++) {
        random_numbers[i] = rand() % 1000 + 1;
    }

    qsort(random_numbers, count, sizeof(int), compare);

    for (i = 0; i < count; i++) {
        if (i == 0 || random_numbers[i] != random_numbers[i - 1]) {
            fprintf(file, "%d\n", random_numbers[i]);
        }
    }

    free(random_numbers);
    fclose(file);
}

int findKthLargest(const char* fileName, int k) {
    FILE* file = fopen(fileName, "r");
    if (file == NULL) {
        perror("File opening error");
        exit(EXIT_FAILURE);
    }

    int number, count = 0;
    int lastNumber = -1, lastCount = 0;
```

```
    while (fscanf(file, "%d", &number) == 1) {
        if (number != lastNumber) {
            lastCount++;
        }

        if (lastCount == k) {
            fclose(file);
            return lastNumber;
        }

        lastNumber = number;
    }

    fclose(file);
    return -1;
}
void mergeAndSortResults(int numChildProcesses, int k) {
    int* results = (int*)malloc(numChildProcesses * sizeof(int));

    for (int i = 1; i <= numChildProcesses; i++) {
        char file_name[20];
        sprintf(file_name, "intermediate%d.txt", i);
        results[i-1] = findKthLargest(file_name, k);
    }

    qsort(results, numChildProcesses, sizeof(int), compare);

    FILE* output_file = fopen("output.txt", "w");
    if (output_file == NULL) {
        perror("Output file opening error");
```

```
        exit(EXIT_FAILURE);
    }

    for (int i = 0; i < numChildProcesses; i++) {
        fprintf(output_file, "%d\n", results[i]);
    }

    fclose(output_file);
    free(results);
}

void processFile(int process_id, int k) {
    char file_name[20];
    sprintf(file_name, "intermediate%d.txt", process_id);
    writeRandomNumbersToFile(file_name, 1000);
}

void createChildProcesses(int numChildProcesses, int k) {
    for (int i = 1; i <= numChildProcesses; i++) {
        pid_t pid = fork();

        if (pid == -1) {
            perror("Fork error");
            exit(EXIT_FAILURE);
        } else if (pid == 0) {
            processFile(i, k);
            exit(EXIT_SUCCESS);
        }
    }
```

```
    for (int i = 0; i < numChildProcesses; i++) {
        wait(NULL);
    }
}

int main(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "Usage: %s <K> <N>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    int K = atoi(argv[1]);
    int N = atoi(argv[2]);

    if (N < 1 || N > 5) {
        fprintf(stderr, "Invalid number of child processes. Should be between 1 and 5.\n");
        exit(EXIT_FAILURE);
    }

    K++;

    if (K < 1 || K > 1000) {
        fprintf(stderr, "Invalid value of K. Should be between 1 and 1000.\n");
        exit(EXIT_FAILURE);
    }

    createChildProcesses(N, K);

    mergeAndSortResults(N, K);
```

```
    deleteTemporaryFiles();


    return 0;
}
```

1.1 Part 1 Çağırılma Şekli (Findtopk)

Bu C dosyasını çağırılırken girilmesi gereken değişkenler şu şeklinde tasarlanmıştır. K sıralanmış dosyalar arasında aranan değeri, N ise oluşturulacak çocuk işlem sayısı ve ara dosya sayısını belirtir.

findtopk <K> <N>

1.2 Part 1 Açıklama (Findtopk)

Bu C programı, komut satırından alınan K ve N değerleriyle, N adet çocuk işlem oluşturur. Her bir çocuk işlem, "intermediateX.txt" adında geçici dosyalar oluşturur ve bu dosyaların içine rastgele sayı ataması yapar. Ardından bu dosyalardaki sayıları sıralar, ardışık aynı sayıları filtreler ve her bir dosyadan K. en büyük sayıyı bulur.

Ana süreç, tüm çocuk süreçlerin K. en büyük sayılarını bir dizi yardımı ile birleştirir, sıralar ve "output.txt" adında bir dosyaya yazar. Bu işlem için çocuk süreçlerle oluşturulan geçici dosyalar kullanılır. Çocuk süreçlerin oluşturulması ve sonlandırılması, "fork" ve "wait" sistem çağrıları kullanılarak gerçekleştirilir.

Geçici dosyaların silinmesi için "deleteTemporaryFiles()" fonksiyonu kullanılır. Program, işlevsel modüler yapıya sahiptir; "writeRandomNumbersToFile", "findKthLargest", "mergeAndSortResults" gibi fonksiyonlar belirli görevleri yerine getirir ve bu görevler, başlık dosyaları ve sistem çağrılarıyla desteklenir. Ayrıca, "compare" fonksiyonu "qsort" ile kullanılarak sıralama işlemi gerçekleştirilir. Bu program, çok sayıda dosya işlemini ve çocuk süreç yönetimini etkin bir şekilde kullanarak verimli bir K. en büyük sayıları bulma işlevselliği sunar.

2.0 Part 2 Kod Gösterimi (Findtopk_mqueue)

```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

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

```
#include <sys/msg.h>
#include <sys/wait.h>
#include <string.h>

struct message {
    long mtype;
    char mtext[100];
};

int compare(const void *a, const void *b) {
    return (*(int *)b - *(int *)a);
}

int main(int argc, char *argv[]) {
    if (argc < 6) {
        fprintf(stderr, "KullanÄ±m: %s <K> <Num_message> <infile1> ... <infileN> <outfile>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    int K = atoi(argv[1]);
    int num_messages = atoi(argv[2]);
    char output_filename[100];
    strcpy(output_filename, argv[argc - 1]);



    int *k_values = (int *)malloc(num_messages * sizeof(int));
    if (k_values == NULL) {
        perror("Bellek tahsisi hatasÄ±");
        exit(EXIT_FAILURE);
    }
```

```
key_t key = ftok(".", 'a');
int msgid = msgget(key, 0666 | IPC_CREAT);

if (msgid == -1) {
    perror("msgget");
    exit(EXIT_FAILURE);
}

pid_t child_pid = fork();

if (child_pid == -1) {
    perror("fork");
    exit(EXIT_FAILURE);
}

if (child_pid == 0) {

    for (int i = 0; i < num_messages; ++i) {
        struct message msg;
        msg.mtype = 1;

        strcpy(msg.mtext, argv[i + 3]);

        FILE *file = fopen(msg.mtext, "w");
        if (file == NULL) {
            perror("Dosya aÃ§ma hatasÄ±");
            exit(EXIT_FAILURE);
        }
```

```
        for (int j = 0; j < 1000; ++j) {
            int random_value = rand() % 1000 + 1;
            fprintf(file, "%d\n", random_value);
        }

        fclose(file);

        if (msgsnd(msgid, &msg, sizeof(msg.mtext), 0) == -1) {
            perror("msgsnd");
            exit(EXIT_FAILURE);
        }
    }

    exit(EXIT_SUCCESS);
} else {

    wait(NULL);

    for (int i = 0; i < num_messages; ++i) {
        struct message received_msg;
        if (msgrcv(msgid, &received_msg, sizeof(received_msg.mtext), 1, 0) == -1) {
            perror("msgrcv");
            exit(EXIT_FAILURE);
        }

        FILE *file = fopen(received_msg.mtext, "r");
        if (file == NULL) {
            perror("Dosya aÃ§ma hatasÄ±");
            exit(EXIT_FAILURE);
        }
```

```
int values[1000];
 int count = 0;

 while (fscanf(file, "%d", &values[count]) != EOF && count < 1000) {
    count++;
 }

 fclose(file);

 qsort(values, count, sizeof(int), compare);

 int filtered_values[1000];
 int filtered_count = 0;

 for (int j = 0; j < count; ++j) {
    if (j == 0 || values[j] != values[j - 1]) {
        filtered_values[filtered_count++] = values[j];
    }
 }

 k_values[i] = filtered_values[K - 1];

 file = fopen(received_msg.mtext, "w");
 if (file == NULL) {
    perror("Dosya aÃ§ma hatasÄ±");
    exit(EXIT_FAILURE);
 }

 for (int j = 0; j < filtered_count; ++j) {
    fprintf(file, "%d\n", filtered_values[j]);
```

```
            }

            fclose(file);
        }

        qsort(k_values, num_messages, sizeof(int), compare);

        FILE *output_file = fopen(output_filename, "w");
        if (output_file == NULL) {
            perror("Dosya aÃ§ma hatasÄ±");
            exit(EXIT_FAILURE);
        }

        for (int i = 0; i < num_messages; ++i) {
            fprintf(output_file, "%d\n", k_values[i]);
        }

        fclose(output_file);

        free(k_values);

        if (msgctl(msgid, IPC_RMID, NULL) == -1) {
            perror("msgctl");
            exit(EXIT_FAILURE);
        }
    }

    return 0;
}
```

2.1 Part 2 Çağırılma Şekli (Findtopk_mqueue)

Bu C dosyasını çağırırken girilmesi gereken değişkenler şu şekilde tasarlanmıştır. K sıralanmış dosyalar arasında aranan değeri, Num_message aile ile çocuk arasında iletişimi sağlayacak mesaj sayısını, infile dosyaları oluşturulacak dosyaların isimlerini ve outfile dosyası en son sonuçları döndürecek dosyanın adını sembolize eder.

findtopk_mqueue <K> <Num_message> <infile1> ... <infileN> <outfile>

2.2 Part 2 Açıklama (Findtopk_mqueue)

Bu program, komut satırı arayüzü üzerinden kullanıcı tarafından sağlanan dosya adlarını işleyerek bir dizi adı aldıktan sonra, her bir dosyaya rastgele sayılar ekler. Daha sonra, her dosyanın içeriği sıralanır ve ardışık aynı değerler filtrelenir bu filtreleme işlemi sırasında aynı değere sahip değişkenlerden sadece 1 tanesi kalacak şekilde veri tabanı güncellenir böylece K. Değere ulaşılması sırasında kolaylık sağlanır.

Program, kullanıcı tarafından belirlenen bir K değeri ile her dosyanın bu sıralı ve filtrelenmiş veri setlerinden K. elemanları alır. Her dosyadan alınan K. elemanlar arasında sıralama yapılarak proje geliştiricilerden istenen çıkış bilgisi dosyasında K. Elemanlar arasında büyükten küçüğe bir sıralama yapılır ve çıkış dosyasında her dosyadan gelen K. değerler her satırda 1 kez olmak üzere yazılır.

Bu süreç, dosya işlemleri için C dilinin standart kütüphane fonksiyonlarını kullanarak gerçekleştirilir. Ayrıca, programda hata kontrolü de sağlanmaktadır. Çalışma sonunda, dinamik olarak tahsis edilen bellek alanlarını serbest bırakır ve IPC mekanizmalarıyla mesaj kuyruğunu düzenler, böylece sistem kaynaklarını verimli bir şekilde yönetir. Bu programın çalışma sürecinde gerekli bilgilerin aktarımı mesajlar yolu ile yapılmıştır bu nedenle ara dosya kullanılmadan iş yükü azaltılmış bir şekilde gerekli işlemler gerçekleştirilmiştir.

3.0 Part 3 Kod Gösterimi (Findtopk_thread)

```
#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <stdint.h>

#define MAX_FILENAME_LENGTH 50

#define MAX_THREADS 5

#define NUM_RANDOM_NUMBERS 1000
```

```
char filenames[MAX_THREADS][MAX_FILENAME_LENGTH];

char mainOutputFileName[MAX_FILENAME_LENGTH];

int K;

pthread_mutex_t mutex;

int kthValues[MAX_THREADS];

int compare(const void *a, const void *b) {
    return (*(int *)b - *(int *)a);

void *threadFunction(void *arg);

int main(int argc, char *argv[]) {
    if (argc < 4 || argc > MAX_THREADS + 3) {
        printf("Usage: %s <K> <N> <infile1> ... <infileN> <outfile>\n", argv[0]);
        return 1;
    }

    K = atoi(argv[1]);
    int threadCount = atoi(argv[2]);

    if (threadCount < 1 || threadCount > MAX_THREADS || K < 1) {
        printf("Invalid input. K and N should be positive integers, and N should be between 1 and
%d.\n", MAX_THREADS);
        return 1;
    }

    snprintf(mainOutputFileName, MAX_FILENAME_LENGTH, "%s", argv[argc - 1]);
```

```
if (pthread_mutex_init(&mutex, NULL) != 0) {
    fprintf(stderr, "Mutex creation error\n");
    return 1;
}
pthread_t threads[MAX_THREADS];

for (int i = 0; i < threadCount; ++i) {
    snprintf(filenames[i], MAX_FILENAME_LENGTH, "%s", argv[i + 3]);

    if (strstr(filenames[i], ".txt") == NULL) {
        strcat(filenames[i], ".txt");
    }

    if (pthread_create(&threads[i], NULL, threadFunction, (void *)(intptr_t)(i + 1)) != 0) {
        fprintf(stderr, "Thread creation error\n");
        return 1;
    }
}

for (int i = 0; i < threadCount; ++i) {
    pthread_join(threads[i], NULL);
}

qsort(kthValues, threadCount, sizeof(int), compare);

FILE *mainOutputFile = fopen(mainOutputFileName, "w");
if (mainOutputFile == NULL) {
    fprintf(stderr, "Error opening the main output file\n");
    return 1;
```

```
    }

    for (int i = 0; i < threadCount; ++i) {
        fprintf(mainOutputFile, "%d\n", kthValues[i]);
    }

    fclose(mainOutputFile);

    pthread_mutex_destroy(&mutex);

    printf("Program completed successfully.\n");

    return 0;
}

void *threadFunction(void *arg) {
    int threadID = (int)(intptr_t)arg;

    char filename[MAX_FILENAME_LENGTH];
    snprintf(filename, MAX_FILENAME_LENGTH, "%s", filenames[threadID - 1]);

    if (strstr(filename, ".txt") == NULL) {
        strcat(filename, ".txt");
    }

    FILE *file = fopen(filename, "w");
    if (file == NULL) {
        fprintf(stderr, "Error creating the file\n");
        pthread_exit(NULL);
    }
```

```
for (int i = 0; i < NUM_RANDOM_NUMBERS; ++i) {
    fprintf(file, "%d\n", rand() % 1000 + 1);
}

fclose(file);

int numbers[NUM_RANDOM_NUMBERS];
file = fopen(filename, "r");
if (file == NULL) {
    fprintf(stderr, "Error opening the file\n");
    pthread_exit(NULL);
}

for (int i = 0; i < NUM_RANDOM_NUMBERS; ++i) {
    fscanf(file, "%d", &numbers[i]);
}

fclose(file);

qsort(numbers, NUM_RANDOM_NUMBERS, sizeof(int), compare);

int uniqueNumbers[NUM_RANDOM_NUMBERS];
int uniqueCount = 1;
uniqueNumbers[0] = numbers[0];

for (int i = 1; i < NUM_RANDOM_NUMBERS; ++i) {
    if (numbers[i] != numbers[i - 1]) {
        uniqueNumbers[uniqueCount] = numbers[i];
        uniqueCount++;
    }
```

```
}

file = fopen(filename, "w");
if (file == NULL) {
    fprintf(stderr, "Error opening the file\n");
    pthread_exit(NULL);
}

for (int i = 0; i < uniqueCount; ++i) {
    fprintf(file, "%d\n", uniqueNumbers[i]);
}

fclose(file);

file = fopen(filename, "r");
if (file == NULL) {
    fprintf(stderr, "Error opening the file\n");
    pthread_exit(NULL);
}

int kthValue;
for (int i = 0; i < K; ++i) {
    if (fscanf(file, "%d", &kthValue) != 1) {
        fprintf(stderr, "Error reading Kth value from the file\n");
        fclose(file);
        pthread_exit(NULL);
    }
}

fclose(file);
```

```
    pthread_mutex_lock(&mutex);
    kthValues[threadID - 1] = kthValue;
    pthread_mutex_unlock(&mutex);

    pthread_exit(NULL);
}
```

3.1 Part 3 Çağırılma Şekli (Findtopk_thread)

Bu C dosyasını çağırırken girilmesi gereken değişkenler şu şekilde tasarlanmıştır. K sıralanmış dosyalar arasında aranan değeri, N oluşturulacak iş parçacığı sayısı, infile dosyaları oluşturulacak dosyaların isimlerini ve outfile dosyası en son sonuçları döndürecek dosyanın adını sembolize eder.

findtopk_thread <K> <N> <infile1> ... <infileN> <outfile>

3.2 Part 3 Açıklama (Findtopk_thread)

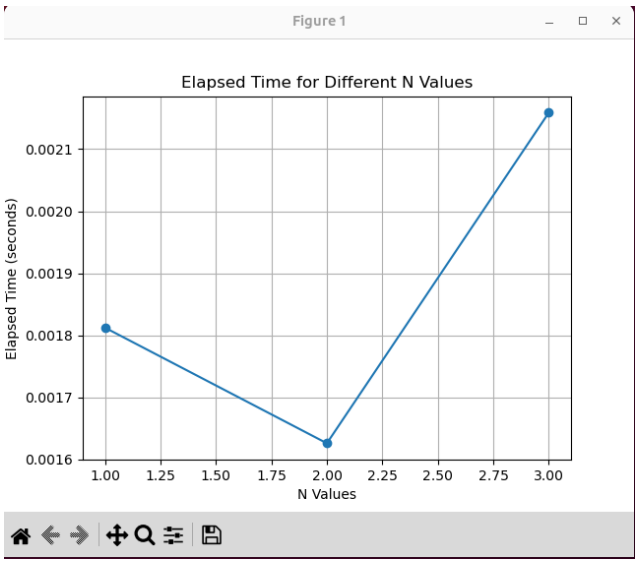
Bu program, birden fazla iş parçacığı(N) (thread) ile çalışarak, komut satırından alınan parametrelere dayalı olarak dosyalarda rastgele sayılar oluşturur, sıralar ve ardından her dosya için belirli bir sırada gelen en büyük sayıları(K) bulur Program, ana iş parçacığından diğer iş parçacıklarına veri paylaşımı için mutex kullanarak senkronize edilmiş çoklu iş parçacığı yürütme kabiliyetine sahiptir. Ana fonksiyon, komut satırı argümanlarını denetler ve ardından belirtilen sayıda iş parçacığı oluşturur. Her iş parçacığı, dosyaya rastgele sayılar yazar, bu sayıları sıralar, ardışık aynı sayıları filtreler ve komut satırından alınan K değerine göre en büyük sayıyı bulur. Her iş parçacığı, bu değeri ana iş parçacığındaki bir diziye kaydeder. Sonuçlar, ana iş parçacığı tarafından toplanır, sıralanır ve belirtilen çıkış dosyasına yazılır. Bu çoklu iş parçacıklı yapı, dosya işlemlerini eş zamanlı olarak gerçekleştirirken verimliliği artırır ve hata kontrolleriyle güvenilirlik sağlar.

4.0 Part 4 Experimentation

Python kullanarak geliştirdiğimiz üç farklı C programının performansını değerlendirmek amacıyla, her bir programın çalışma sürelerini ölçerek karşılaştırma yapmayı hedefledik. Bu süreçte, her bir C programını Python dilinde tasarlanmış bir test senaryosu içinde çalıştırdık ve elde ettiğimiz süre verilerini kullanarak performans analizi yaptık. Daha sonra, elde ettiğimiz verileri grafiksel olarak ifade etmek amacıyla bir çizgi grafiği oluşturduk.

Analizimiz, her bir C programının işlem sürelerini karşılaştırarak hangi programın daha etkili ve verimli olduğunu belirlemeyi amaçlamaktadır. Çalışma sürelerinin grafik üzerindeki görsel karşılaştırması, programların performansını daha anlaşılır bir şekilde ifade etmemizi sağlamaktadır. Bu analiz, programların çeşitli giriş verileri üzerindeki performans farklılıklarını belirlemek ve hangi durum için hangi programın daha uygun olduğunu anlamak için yapılmıştır.

4.1 Findtopk Performans Hesaplamaları



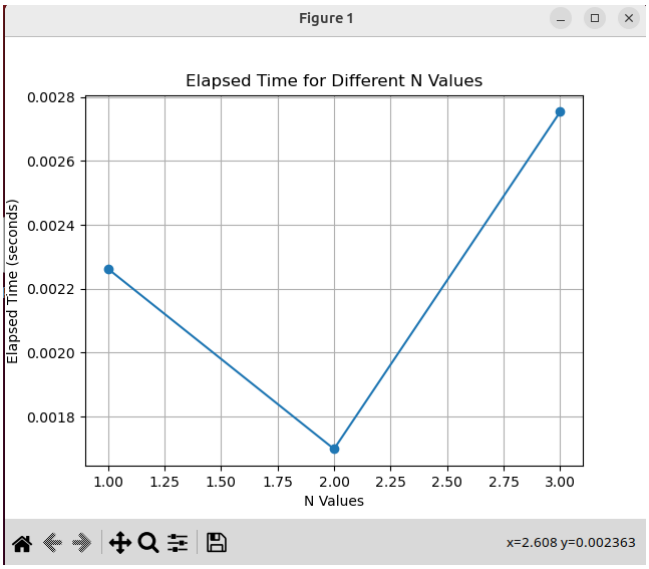
Resim1.0: Findtopk Programının Plot Tablosu

	Standard	Standard
1	N Values	Elapsed Time (seconds)
2	1	0.0018124580383300781
3	2	0.0016262531280517578
4	3	0.002158641815185547

Resim1.1: Findtopk Programının Excel Tablosu

Tablolar incelendiğinde, programda N parametresine 1 değeri verildiğinde çalışma süresi yaklaşık olarak 0.0018, 2 değeri verildiğinde yaklaşık olarak 0.0016, 3 değeri verildiğinde ise yaklaşık olarak 0.0022 süresince gerekli işlemleri gerçekleştirmektedir. En kısa sürede gerçekleşen programda, N değerinin 2 olduğu saptanmıştır.

4.2 Findtopk_mqueue Performans Hesaplamaları



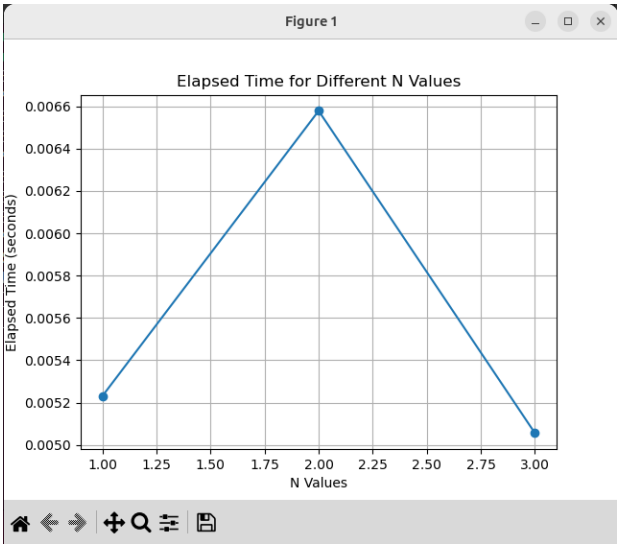
Resim2.0: Findtopk_mqueue Programının Plot Tablosu

	Standard	Standard
1	N Values	Elapsed Time (seconds)
2	1	0.0022628307342529297
3	2	0.0016982555389404297
4	3	0.00275421142578125

Resim2.1: Findtopk_mqueue Programının Excel Tablosu

Tablolar incelendiğinde, programda N parametresine 1 değeri verildiğinde çalışma süresi yaklaşık olarak 0.0023, 2 değeri verildiğinde yaklaşık olarak 0.0017, 3 değeri verildiğinde ise yaklaşık olarak 0.0028 süresince gerekli işlemleri gerçekleştirmektedir. En kısa sürede gerçekleşen programda, N değerinin 2 olduğu saptanmıştır.

4.3 Findtopk_thread Performans Hesaplamaları



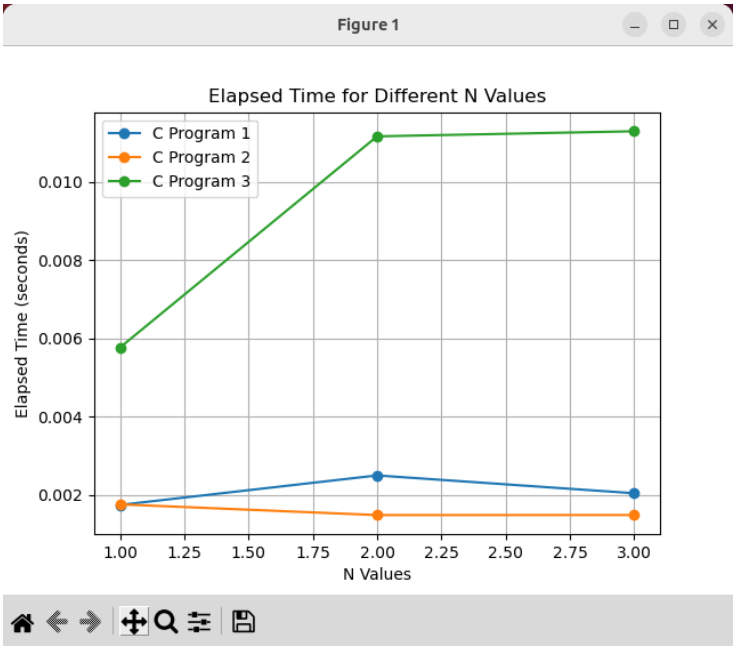
Resim3.0: Findtopk_thread Programının Plot Tablosu

N Values	Elapsed Time (seconds)
1	0.00523090362548828
2	0.00657796859741211
3	0.00505614280700684

Resim3.1: Findtopk_thread Programının Excel Tablosu

Tablolar incelendiğinde, programda N parametresine 1 değeri verildiğinde çalışma süresi yaklaşık olarak 0.0052, 2 değeri verildiğinde yaklaşık olarak 0.0066, 3 değeri verildiğinde ise yaklaşık olarak 0.0051 süresince gerekli işlemleri gerçekleştirmektedir. En kısa sürede gerçekleşen programda, N değerinin 3 olduğu saptanmıştır.

4.4 Findtopk, Findtopk_mqueue ve Findtopk_thread Performans Karşılaştırması



Resim4.0: Performans Karşılaştırma Plot Tablosu

N Values	Elapsed Time (seconds)
1	0.00173711776733398
2	0.00249147415161133
3	0.00203561782836914

N Values	Elapsed Time (seconds)
1	0.00175213813781738
2	0.00147771835327148
3	0.00147914886474609

N Values	Elapsed Time (seconds)
1	0.00575971603393555
2	0.0111582279205322
3	0.0112881660461426

Resim4.1: Performans Karşılaştırma Excel Tabloları

Tablolar incelendiğinde "Findtopk_mqueue" programının daha kısa sürede çalıştığı sonucuna varılmaktadır. Diğerlerinden daha uzun sürede çalışan programın ise "Findtopk_thread" programı olduğu gözlemlenmiştir.