

İşletim Sistemleri

Proje Raporu

Ad, Soyad: Farid Bayramov, Yunus Ege Küçük

No: 170421993, 170421048

Grup No: 29

Projenin amacı: Ana process'in yanı sıra N adet child process oluşturabilen bir program tasarlamak. N adet dosya içerisindeki k. en büyük sayıyı buldurma işlemi yapmak.

Proje içeriği:

Kod yapısı ve mantığı:

- Program, C dili kullanılarak yazılmıştır.
- Ana process, N adet child process oluşturur. Her bir child process, bir input dosyasını okuyarak içindeki sayıları sıralar ve k. en büyük sayıyı ara dosyaya yazar. Input dosyasının içindeki sayılar 1 ile 100000 arasında olacak şekilde, 10000 adet rastgele yazılmıştır.

Part 1.

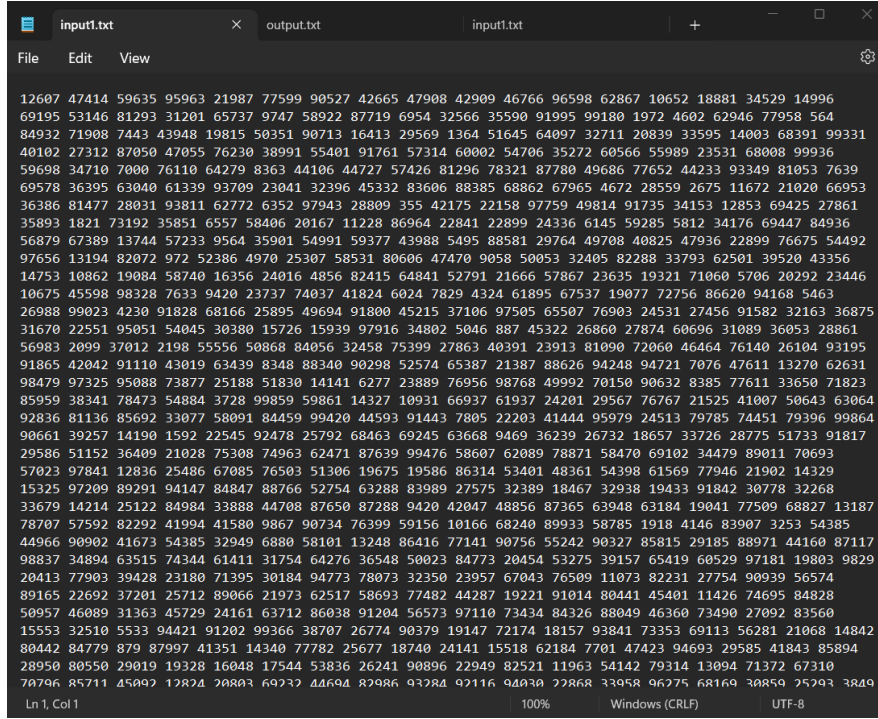
- Child processlerin işi tamamlandığında, ana process ara dosyaları okuyarak birleştirir ve output dosyasına sıralanmış k. en büyük sayıları yazar.
- İşlemler sırasında oluşan ara dosyalar temizlenir.

```
void generateRandomNumbers(const char *filename, int count) {  
    FILE *file = fopen(filename, "w");  
    if (file == NULL) {  
        perror("Error opening file");//hata tespiti  
        exit(EXIT_FAILURE);  
    }  
  
    for (int i = 0; i < count; ++i) {  
        fprintf(file, "%d ", rand() % 100000 + 1);  
    }  
}
```

Kod 1. Input dosyalarının içine rastgele sayı üreten fonksiyon

Input dosyaları:

- Her bir input dosyası, boşluk karakterleriyle ayrılmış sayılar içerir.
- Dosyalardaki sayılar rasgele oluşturulur ve 1-100000 aralığındadır.
- Toplamda N adet input dosyası bulunmaktadır.



```
12607 47414 59635 95963 21987 77599 90527 42665 47908 42909 46766 96598 62867 10652 18881 34529 14996
69195 53146 81293 31201 65737 9747 58922 87719 6954 32566 35590 91995 99180 1972 4602 62946 77958 564
84932 71908 7443 43948 19815 50351 90713 16413 29569 1364 51645 64097 32711 20839 33595 14803 68391 99331
40102 27312 87050 47055 76230 38991 55401 91761 57314 60002 54706 35272 60566 55989 23531 68008 99936
59698 34710 7000 76110 64279 8363 44106 44727 57426 81296 78321 87780 49686 77652 44233 93349 81053 7639
69578 36395 63040 61339 93709 23041 32396 45332 83606 88385 68862 67965 4672 28559 2675 11672 21020 66953
36386 81477 28031 93811 62772 6352 97943 28809 355 42175 22158 97759 49814 91735 34153 12853 69425 27861
35893 1821 73192 35851 6557 58486 20167 11228 86964 22841 22899 24336 6145 59285 5812 34176 69447 84936
56879 67389 13744 57233 9564 35901 54991 59377 43988 5495 88581 29764 49708 40825 47936 22899 76675 54492
97656 13194 82072 972 52386 4970 25307 58531 80606 47470 9058 50053 32405 82288 33793 62501 39520 43356
14753 10862 19084 58740 16356 24016 4856 82415 64841 52791 21666 57867 23635 19321 71060 5706 20292 23446
10675 45598 98328 7633 9420 23737 74037 41824 6024 7829 4324 61895 67537 19077 72756 86620 94168 5463
26988 99023 4230 91828 68166 25895 49694 91800 45215 37106 97505 65507 76903 24531 27456 91582 32163 36875
31670 22551 95051 54045 30380 15726 15939 97916 34802 5046 887 45322 26860 27874 60696 31089 36053 28861
56983 2099 37012 2198 55556 50868 84056 32458 75399 27863 40391 23913 81090 72060 46464 76140 26104 93195
91865 42042 91110 43019 63439 8348 88340 90298 52574 65387 21387 88626 94248 94721 7076 47611 13270 62631
98479 97325 95088 73877 25188 51830 14141 6277 23889 76956 98768 49992 70150 90632 8385 77611 33650 71823
85959 38341 78473 54884 3728 99859 59861 14327 10931 66937 61937 24201 29567 76767 21525 41007 50643 63064
92836 81136 85692 33077 58091 84459 99420 44593 91443 7805 22203 41444 95979 24513 79785 74451 79396 99864
90661 39257 14190 1592 22545 92478 25792 68463 69245 63668 9469 36239 26732 18657 33726 28775 51733 91817
29586 51152 36409 21028 75308 74963 62471 87639 99476 58607 62089 78871 58470 69102 34479 89011 70693
57023 97841 12836 25486 67085 76503 51306 19675 19586 86314 53401 48361 54398 61569 77946 21902 14329
15325 97209 89291 94147 84847 88766 52754 63288 83989 27575 32389 18467 32938 19433 91842 30778 32268
33679 14214 25122 84984 33888 44708 87650 87288 9420 42047 48856 87365 63948 63184 19041 77509 68827 13187
78707 57592 82292 41994 41580 9867 90734 76399 59156 10166 68240 89933 58785 1918 4146 83907 3253 54385
44966 90902 41673 54385 32949 6880 58101 13248 86416 77141 90756 55242 90327 85815 29185 88971 44160 87117
98837 34894 63515 74344 61411 31754 64276 36548 50023 84773 20454 53275 39157 65419 60529 97181 19803 9829
20413 77903 39428 23180 71395 30184 94773 78073 32350 23957 67043 76509 11073 82231 27754 90939 56574
89165 22692 37201 25712 89066 21973 62517 58693 77482 44287 19221 91014 80441 45401 11426 74695 84828
59957 46089 31363 45729 24161 63712 86038 91204 56573 97110 73434 84326 88049 46360 73490 27092 83560
15553 32510 5533 84421 91202 90366 38707 26774 90379 19147 72174 18157 93841 73353 69113 56281 21068 14842
80442 84779 879 87997 41351 14340 77782 25677 18740 24141 15518 62184 7701 47423 94693 29585 41843 85894
28950 80550 29019 19328 16048 17544 53836 26241 90896 22949 82521 11963 54142 79314 13094 71372 67310
70746 85711 45002 17824 20803 60232 44694 87486 93284 97116 40430 22868 33058 96275 68160 30850 25293 3849
```

Resim 1. Örnek input dosyası

Child process işleyişi:

1. Her child process, bir input dosyasını okur.
2. Okunan sayıları sıralamak için **bubble sort** algoritması kullanılır.
3. Sıralanan sayılar ara dosyaya yazılır.

```
for (int i = 0; i < 10000 - 1; ++i) {
    for (int j = 0; j < 10000 - i - 1; ++j) {
        if (numbers[j] < numbers[j + 1]) {
            int temp = numbers[j];
            numbers[j] = numbers[j + 1];
            numbers[j + 1] = temp;
        }
    }
}
```

Kod 2. Bubble Sort

Merge işlemi:

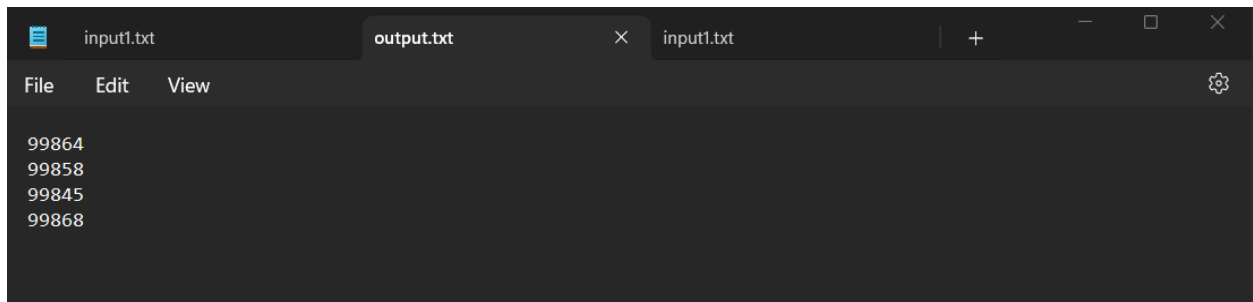
1. Ana process, N adet ara dosyayı birleştirir.
2. Birleştirme işlemi sırasında sayılar büyükten küçüğe doğru sıralanır.
3. Sonuçlar output dosyasına yazılır.

```
void mergeFiles(int n, const char *outfile) {  
    FILE *outputFile = fopen(outfile, "w");  
    if (outputFile == NULL) {  
        perror("Error opening output file");  
        exit(EXIT_FAILURE);  
    }  
  
    for (int i = 0; i < n; ++i) {  
        char intermediateFilename[20];  
        snprintf(intermediateFilename, sizeof(intermediateFilename), "intermediate%d.txt", i + 1);  
  
        FILE *intermediateFile = fopen(intermediateFilename, "r");  
        if (intermediateFile == NULL) {  
            perror("Error opening intermediate file");  
            fclose(outputFile);  
            exit(EXIT_FAILURE);  
        }  
  
        int number;  
  
        //ara dosyalardaki sayıları output dosyasına yazıyoruz  
        while (fscanf(intermediateFile, "%d", &number) == 1) {  
            fprintf(outputFile, "%d\n", number);  
        }  
  
        fclose(intermediateFile);  
    }  
    fclose(outputFile);  
}
```

Kod 3. Ara dosyaları birleştiren fonksiyon

Output dosyası:

- Output dosyasında her bir satırda yalnızca bir integer değer bulunmaktadır.
- Output dosyası, programın başarıyla tamamlandığını gösterir.



Resim 2. Örnek output dosyası

Main:

- Minimum argüman sayısı kontrol edilir, eksik argüman durumunda kullanıcıya doğru kullanım bilgisi verilir.
- Kullanıcı tarafından belirtilen k ve N değerleri kontrol edilir ve geçerli olup olmadığına dair bilgi verilir.
- Random sayılarla oluşturulan N adet input dosyası için child processler oluşturulur.
- Oluşturulan child processlerin bitmesi beklenir.
- Child processlerin ürettiği ara dosyalar birleştirilir.
- Ara dosyalar temizlenir.
- Son olarak, ara dosyalar temizlenir ve programın başarılı bir şekilde tamamlandığı bildirilir.

```
int main(int argc, char *argv[]) {
    //minimum arguman sayısı kontrol ediyor, programi cagirma seklimize uygun
    if (argc < 5 || argc > MAX_N + 4) {
        fprintf(stderr, "Usage: %s <k> <N> <infile1> ... <infileN> <outfile>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    int k = atoi(argv[1]); //k degeri
    int N = atoi(argv[2]); //N degeri

    //bu blokda k ve N nin degerleri kontrol ediliyor
    if (k < 1 || k > MAX_K || N < 1 || N > MAX_N) {
        fprintf(stderr, "Invalid values for k or N\n");
        exit(EXIT_FAILURE);
    }

    srand((unsigned int)time(NULL));

    for (int i = 0; i < N; ++i) {
        generateRandomNumbers(argv[i + 3], 10000); //i+3 giris dosyasinin program cagirilmasindaki sirasi, input dosyasna 10000 sayi yazdiriliyor
    }

    // child processler olusturuluyor
    pid_t pid;
    for (int i = 0; i < N; ++i) {
        pid = fork();

        if (pid == -1) {
            perror("fork");
            exit(EXIT_FAILURE);
        } else if (pid == 0) { // cocuk prosesdir
            childProcess(argv[i + 3], i + 1, k);
            exit(EXIT_SUCCESS);
        }
    }

    // tum child proceslerin bitmesini bekliyoruz.
    int status;
    while (wait(&status) > 0);

    // ara dosyaları birleştiriyoruz
    mergeFiles(N, argv[N + 3]);

    // ara dosyaları kaldırıyoruz
    for (int i = 0; i < N; ++i) {
        char intermediateFilename[20];
        sprintf(intermediateFilename, sizeof(intermediateFilename), "intermediate%d.txt", i + 1);
        remove(intermediateFilename);
    }

    return 0;
}
```

Kod 4. Main fonksiyonu

Part 2. POSIX Message Queues ile işlem

Bu aşamada, Part 1'deki çocuk süreçler arasındaki bilgi paylaşımı için kullanılan ara dosyaların yerine **POSIX message queues** kullanılmıştır.

Bu kısımda, **mq_open**, **mq_send**, ve **mq_receive** fonksiyonları kullanılarak mesaj kuyrukları oluşturulmuş ve yönetilmiştir.

Mesaj yapısı: Mesaj kuyruklarıyla veri iletimi için bir **struct Message** yapısı tanımlanmıştır. Bu yapı, mesaj türünü (**mtype**) ve veriyi (**data**) içermektedir.

```
struct Message {
    long mtype; // mesajın turu
    int data;   //veri
};
```

Kod 5. Struct

Mesaj kuyruğu oluşturma:

Ana süreç, **mq_open** fonksiyonu ile mesaj kuyruğu oluşturmuştur. Ayrıca, **struct mq_attr** yapısı kullanılarak message queue özellikleri belirlenmiştir.

```
mqd_t parentQueue;
struct mq_attr attr;
attr.mq_flags = 0;
attr.mq_maxmsg = 10; // kuyruktaki mesajların maksimum sayısı
attr.mq_msgsize = sizeof(struct Message);
attr.mq_curmsgs = 0;

parentQueue = mq_open("/topk_mqueue", O_CREAT | O_RDWR, 0666, &attr);
if (parentQueue == (mqd_t)-1) {
    perror("mq_open");
    exit(EXIT_FAILURE);
}
```

Kod 6. Mesaj kuyruğu

Mesaj gönderme ve Alma:

Çocuk süreçler, k. en büyük sayıları sıraladıktan sonra bu değerleri bir **struct Message** içinde **mq_send** fonksiyonu ile mesaj kuyruğuna göndermektedir. Ana süreç ise **mq_receive** fonksiyonu ile mesaj kuyruğundan bu mesajları alarak işlemektedir.

```
struct Message msg;
msg.mtype = processNumber;
msg.data = numbers[k - 1];

// Sonucu parent prosese gönderme
if (mq_send(parentQueue, (const char *)&msg, sizeof(struct Message), 0) == -1) {
    perror("mq_send");
    exit(EXIT_FAILURE);
}

free(numbers);
fclose(inputFile);
}
```

Kod 7. Mesaj gönderme

```

if (mq_receive(parentQueue, (char *)&msg, sizeof(struct Message), NULL) == -1) {
    perror("mq_receive");
    fclose(outputFile);
    mq_close(parentQueue);
    return 1;
}

```

Kod 8. Mesaj alma

Part 3. POSIX Threads ile İşlem

Part 3, part 2'deki programın aynısını **POSIX thread'leri** kullanarak yeniden tasarlamaktadır. Bu aşama, çocuk süreçlerin yerine thread'leri kullanarak çoklu dosyalar üzerinde sıralama ve k. en büyük sayıyı bulma işlemini gerçekleştirmektedir.

1. Kod Yapısı ve Çalışma Mantığı

1.1. struct ThreadData

Bu yapı, thread işlevine geçirilecek verileri içerir. İçinde dosya adı (**infile**), işlem numarası (**processNumber**), k değeri (**k**) ve sonuç (**result**) bulunmaktadır.

```

// thread verilerini tutan yapı
struct ThreadData {
    const char *infile;
    int processNumber;
    int k;
    int result;
};

```

Kod 9. Struct

1.2. threadProcess Fonksiyonu

1. Argüman alma: **pthread_create** fonksiyonu ile oluşturulan thread'in argümanı olarak bir **struct ThreadData** yapısı alır. Bu yapı, dosya adı, işlem numarası, k değeri ve sonucu içerir.
2. Dosya açma ve bellek ayırma: Verilen dosyayı açar ve okunan sayıları saklamak için bellekte yer ayırır.
3. Sayıları okuma: Dosyadan okunan sayıları, daha sonra sıralama işlemi için bellekte bulunan diziye atar.
4. Bubble sort ile dıralama: **Bubble sort** algoritması kullanılarak, dosyadan okunan sayıları azalan sırayla sıralar.
5. K. en büyük sayıyı bulma: Sıralanan diziden k. en büyük sayıyı bulup, **data -> result** alanına kaydeder.
6. Bellek ve dosya kaynaklarını serbest bırakma: Kullanılan bellek ve dosya kaynaklarını serbest bırakır.
7. Thread'i sonlandırma: **pthread_exit** fonksiyonu ile thread'i sonlandırır.

```

void *threadProcess(void *arg) {
    struct ThreadData *data = (struct ThreadData *)arg;

    FILE *inputFile = fopen(data->infile, "r");
    if (inputFile == NULL) {
        perror("Error opening input file");
        exit(EXIT_FAILURE);
    }

    int *numbers = (int *)malloc(10000 * sizeof(int));
    if (numbers == NULL) {
        perror("Error allocating memory");
        fclose(inputFile);
        exit(EXIT_FAILURE);
    }

    //dosyadan sayilari okuma
    for (int i = 0; i < 10000; ++i) {
        if (fscanf(inputFile, "%d", &numbers[i]) != 1) {
            fprintf(stderr, "Error reading from input file\n");
            free(numbers);
            fclose(inputFile);
            exit(EXIT_FAILURE);
        }
    }

    //bubble sort
    for (int i = 0; i < 10000 - 1; ++i) {
        for (int j = 0; j < 10000 - i - 1; ++j) {
            if (numbers[j] < numbers[j + 1]) {
                int temp = numbers[j];
                numbers[j] = numbers[j + 1];
                numbers[j + 1] = temp;
            }
        }
    }

    //k. buyuk sayiyi bulma
    data->result = numbers[data->k - 1];
    //bellek ve dosya kaynaklarini serbest birak
    free(numbers);
    fclose(inputFile);
    //thread'i sonlandir
    pthread_exit(NULL);
}

```

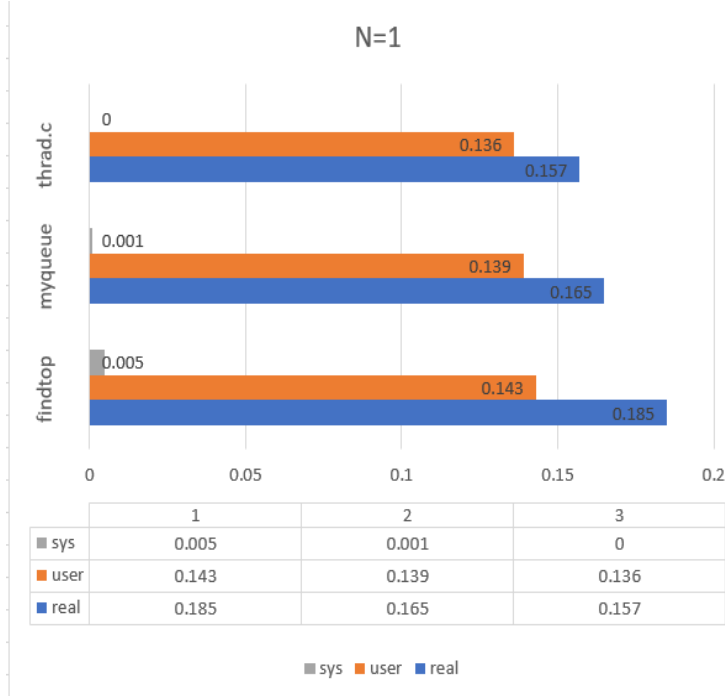
Kod 10. Thread Process

2. Çalışma Akışı

1. Komut satırından alınan argümanlar kontrol edilir ve geçerlilikleri sağlanır.
2. **generateRandomNumbers** fonksiyonu ile N adet dosyaya rastgele sayılar yazılır.
3. Her bir dosya için bir **thread** oluşturulur (**pthread_create**).
4. Oluşturulan thread'ler, **threadProcess** fonksiyonunu çalıştırarak dosyayı sıralar ve k. en büyük sayıyı bulur.
5. **pthread_join** fonksiyonu ile tüm thread'lerin bitmesi beklenir.
6. Sonuçlar, çıkış dosyasına yazılır.

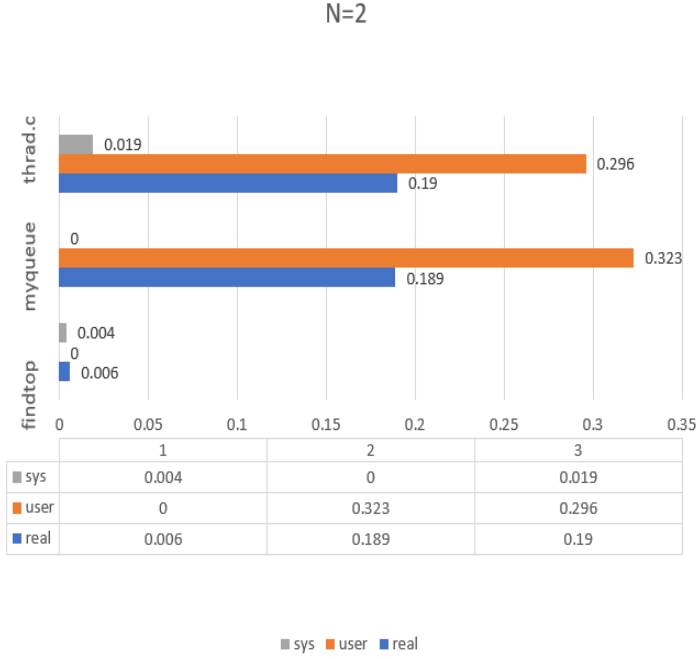
Tablolar:

Aşağıda 1 tane input dosyası olarak çağırılan farklı kodların çalışma sürelerini saniye cinsinden gösteren tablo yer almaktadır. Tabloda görüldüğü üzere en düşük süre sitsem zamanlarına (**sys**) aittir. En uzun süre ise gerçek zamana (**real**) aittir. Kodlara ayrı-ayrı olarak bakılırsa, çalışması en uzun süren kod **findtopk** (total= 0.333s) , en kısa süren ise **thread.c** (0.293s) olmuştur.



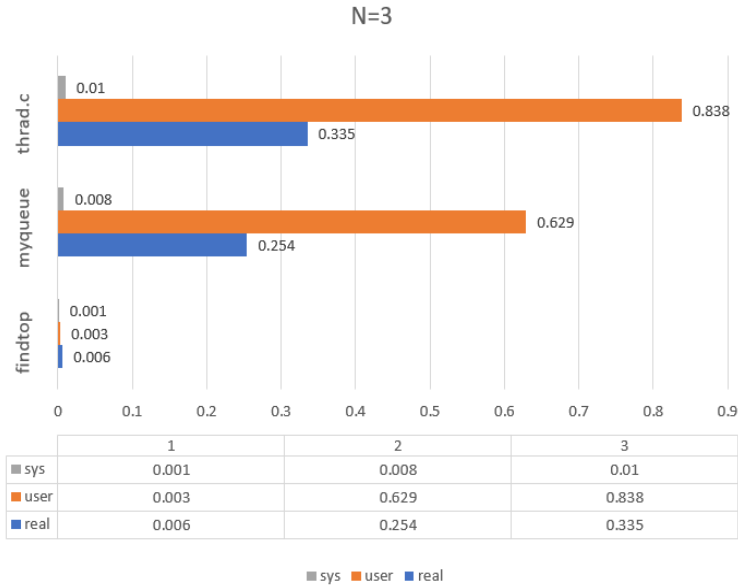
Tablo 1. *Input dosyas sayısı=1*

Aşağıda 2 tane input dosyası olarak çağırılan farklı kodların çalışma sürelerini saniye cinsinden gösteren tablo yer almaktadır. Tabloda görüldüğü üzere programın en düşük çalışma süresi sitsem zamanlarına (**sys**) aittir. En uzun süre ise kullanıcı zamanına (**user**) aittir. Kodlara ayrı-ayrı olarak bakılırsa, çalışması en uzun süren kod **thread.c** (total= 0.505s) , en kısa süren ise **findtopk** (0.01s) olmuştur.



Tablo 2. *Input dosya sayısı=2*

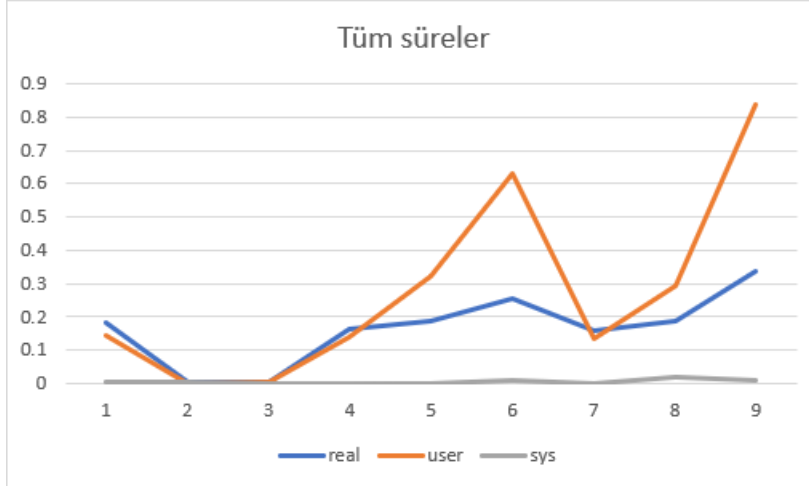
Aşağıda 3 tane input dosyası olarak çağırılan farklı kodların çalışma sürelerini saniye cinsinden gösteren tablo yer almaktadır. Tabloda görüldüğü üzere programın en düşük çalışma süresi sitsem zamanlarına (**sys**) aittir. En uzun süre ise kullanıcı zamanına (**user**) aittir. Kodlara ayrı-ayrı olarak bakılırsa, çalışması en uzun süren kod **thread.c** (total= 1.183s) , en kısa süren ise **findtopk** (0.01s) olmuştur.



Tablo 3. *Input dosya sayısı=3*

Bu tabloların sonucunda findtopk.c nin daha fazla input için daha yararlı olduğunu görebiliriz. İntput sayısı arttıkça bu kodun çalışma süresi diğerlerine göre daha azalıyor. Thread.c fonksiyonu ise tam ters olarak input dosya sayısı arttıkça çalışma süresi de artıyor. Genel olarak tüm programlarda en kısa süre sistem zamanına aittir. Toplam bakacak olursak ise en uzun çalışma süresi kullanıcı zamanına aittir.

Bu plotta da tüm çalışma sürelerinin bilgileri yer almaktadır.



Plot 1. Tüm süreler için plot

Sonuç:

Program, N adet dosyadan okunan sayıları sıralayarak k. en büyük sayıları bulmakta ve sonuçları sıralı bir şekilde output dosyasına yazmaktadır. Hata durumları kontrol edilmiş ve programın doğru çalışması sağlanmıştır. Output dosyası, beklenen çıktıları içermekte ve programın başarıyla tamamlandığını göstermektedir. Main fonksiyonları, programın kontrol akışını yönetmekte ve gerekli adımları sırasıyla gerçekleştirmektedir. Programların farklı zamanlarda Çalışma süreleri ölçülmüş , gerekli tablo ve plotlar çizilmiştir.