MARMARA ÜNİVERSİTESİ

TEKNOLOJİ FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ

BLM3003 İŞLETİM SİSTEMLERİ

FİNAL PROJESİ RAPORU

simge, sembol, yazı tipi, logo, grafik içeren bir resim

Açıklama otomatik olarak oluşturuldu

ÖĞRENCİ NO: 171421012

AD SOYAD: Mertcan GELBAL

ÖĞRENCİ NO: 170421041

AD SOYAD: Ahmet Yasir KULAKSIZ

Bu rapor, N adet child process kullanarak belirli dosyalardaki k. en büyük sayıları bulma işlemini ele almakta ve bu süreci optimize etmek için message queue'ler ve thread'lerin nasıl kullanılabileceğini incelemektedir. Farklı N değerleri, dosya boyutları ve k değerleri üzerinde yapılan deneylerim, farklı yöntemlerin performansını değerlendirmek ve bu işlem için en etkili stratejiyi belirlemek amacıyla gerçekleştirilmiştir. Çalışmanın amacı, çoklu süreçlerin, mesaj kuyruklarının ve thread'lerin kullanımının, k. en büyük sayıları bulma görevindeki performans artışına nasıl katkıda bulunduğunu anlamak ve bu yöntemlerin avantajlarını araştırmaktır.

PART 1

Bu programın amacı, belirtilen sayıda child process oluşturarak N adet dosyadaki en büyük k. sayıları bulmak ve bunları sıralayarak bir çıkış dosyasına yazmaktır. İşlem adımları şu şekilde özetlenebilir:

1. Komut Satırı Parametrelerini Alma:

* Program, komut satırından alınan parametreleri kullanarak çalışmaktadır. Bu parametreler arasında k (1 ile 1000 arasında), N (1 ile 5 arasında), giriş dosyaları ve çıkış dosyası bulunmalıdır.

1. Child Process Oluşturma:

* Program, belirtilen sayıda (N) child process oluşturmaktadır.

1. Her Child Process'in Görevi:

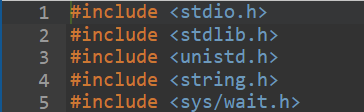
* Her child process, kendisine atanmış olan giriş dosyasını okuyarak en büyük k. sayıyı belirlemektedir. Bu sayıyı bulduktan sonra, bu değeri bir ara dosyaya yazmaktadır.

1. Ana Process'in Görevi:

* Ana process, tüm child processlerin bitmesini beklemektedir.
* Tüm ara dosyaları okuyarak içerdikleri en büyük k. sayıları birleştirmekte ve sıralamaktadır.
* Sonuçları çıkış dosyasına yazmaktadır.

1. Ara Dosyaları Silme:

* Tüm işlemler tamamlandığında, program ara dosyaları silmektedir.



Şekil 1

Şekil 1’de gerekli olan kütüphaneler eklenmektedir.

1. selectionsort

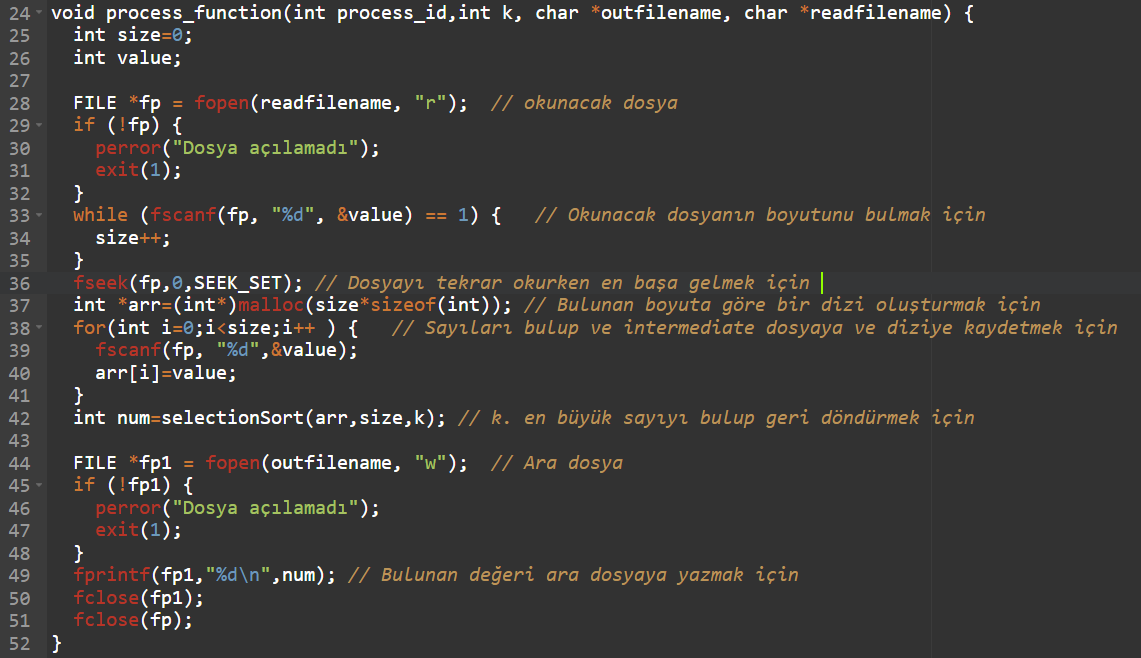
metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Şekil 2

Şekil 2’deki fonksiyon selection sort algoritmasıdır. Fakat bir dönüş değeri olarak bize dizideki en büyük k. elemanı döndürmektedir.

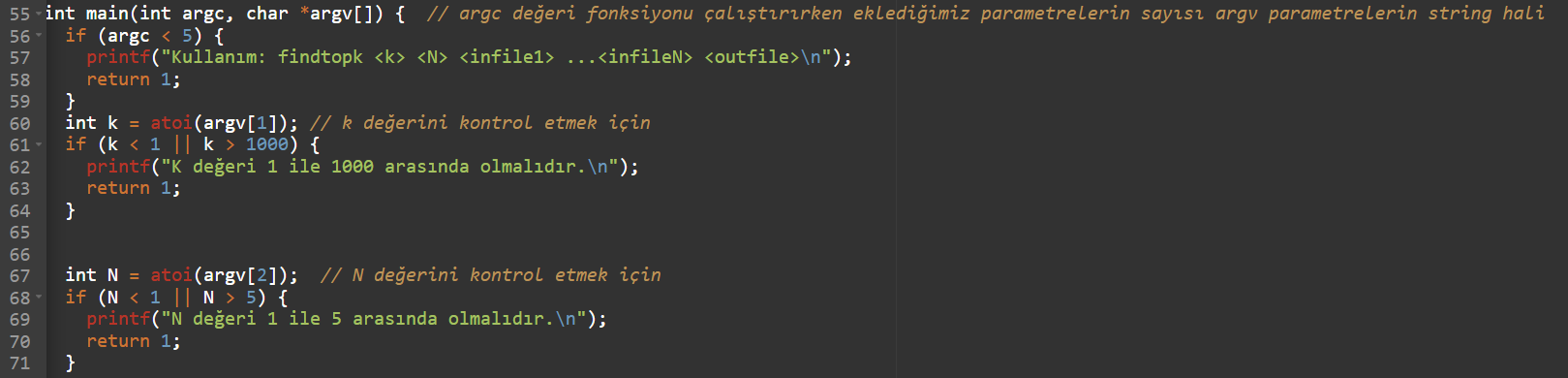
1. process\_function



Şekil 3

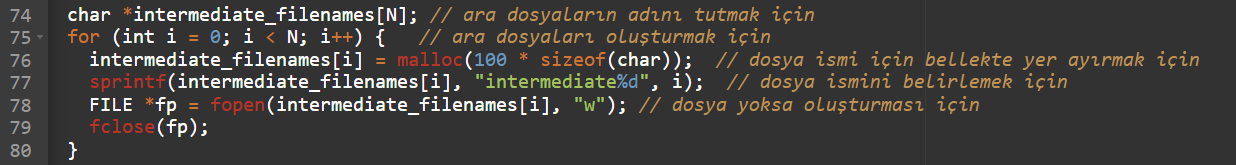
Şekil 3’teki fonksiyonda okunan dosyadaki satır sayısı bulunarak uygun bir dizi oluşturulmaktadır. Bu dizi şekil 2’deki fonksiyona gönderilerek en büyük k. değer bulunmaktadır. Bulunan değer intermediate dosyaya yazılmaktadır.

1. main



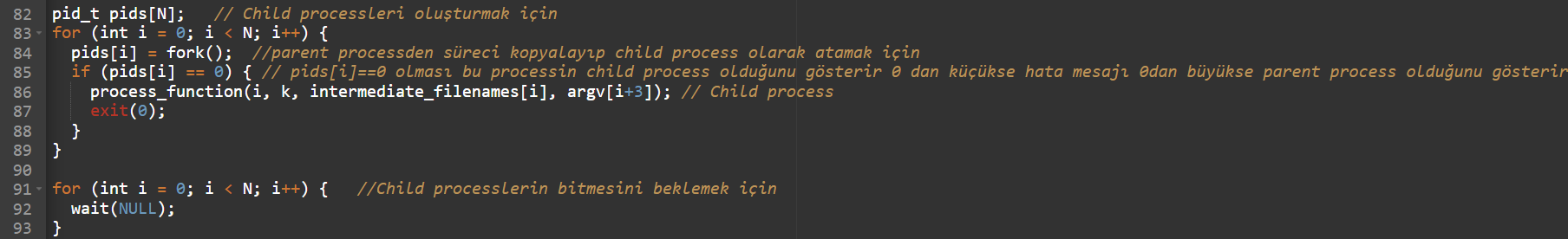
Şekil 4

Şekil 4’teki kod bloğunda parametrelerin kontrolü yapılmaktadır.



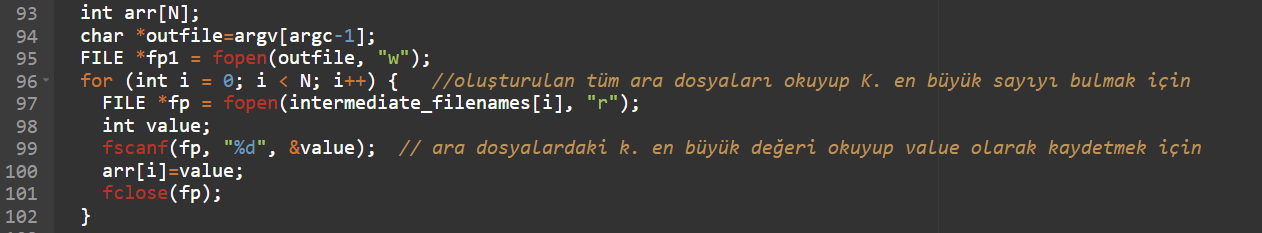
Şekil 5

Şekil 5’teki kod bloğunda ara dosyaların oluşturulması, bellekte yer ayrılması ve isimlendirme işlemleri yapılmaktadır.



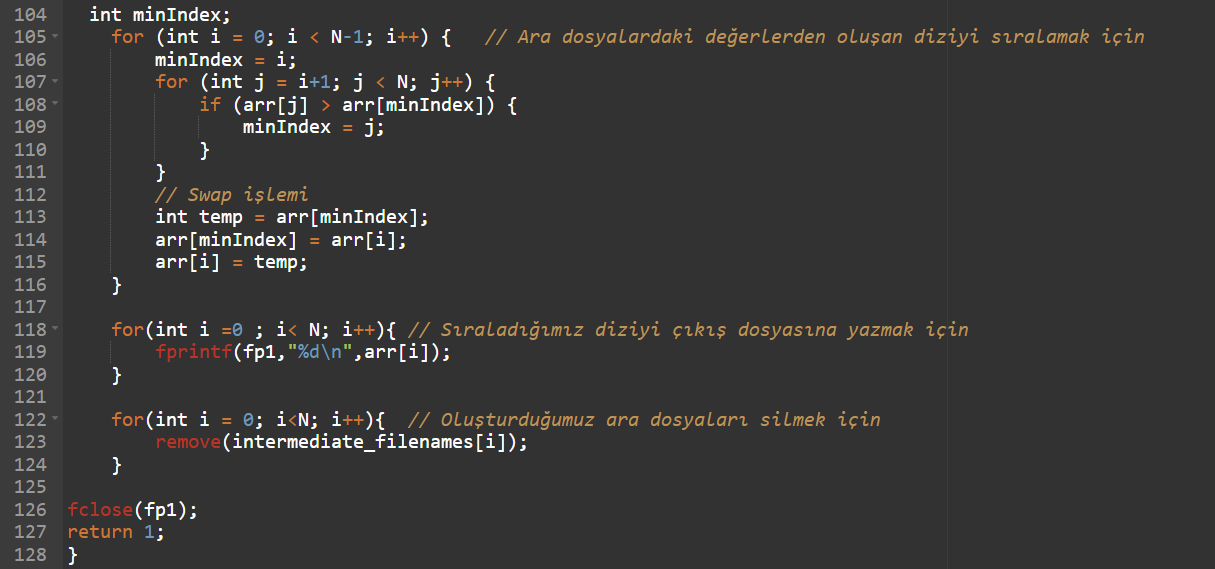
Şekil 6

Şekil 6’daki kod bloğu child process’leri oluşturup parent process’den sürecin kopyalanması sağlanmaktadır. Sonrasında ise child process’lerin işlemleri bitene kadar program beklemektedir.



Şekil 7

Şekil 7’deki kod bloğu oluşturulan ara dosyalardaki k. en büyük değeri arr dizisine kaydetmektedir.



Şekil 8

Şekil 8’deki kod bloğu elde ettiğimiz arr dizisini sıralayıp çıktı dosyasına yazmaktadır. Ayrıca oluşturulan ara dosyaları silme işlemi burada gerçekleşmektedir.

PART 2

Part 2 için işlem adımları şu şekilde özetlenebilir:

1. Komut Satırı Parametrelerini Alma:

* Program, komut satırından alınan parametreleri kullanarak çalışmaktadır. Bu parametreler arasında k (1 ile 1000 arasında), N (1 ile 5 arasında), giriş dosyaları ve çıkış dosyası bulunmaktadır.

1. Message Queue Oluşturma:

* Program, POSIX message queue'ları kullanarak parent ve child process’ler arasında bilgi paylaşımını sağlamaktadır. Message queue'ların oluşturulması için mq\_open fonksiyonu kullanılmaktadır.

1. Child Process Oluşturma:

* Program, belirtilen sayıda (N) child process oluşturmaktadır. Her bir child process, kendi message queue'larına erişim sağlamaktadır.

1. Her Child Process'in Görevi:

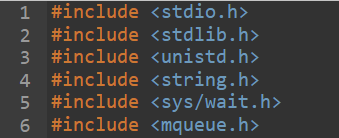
* Her child process, kendisine atanmış olan giriş dosyasını okuyarak en büyük k. sayıyı belirlemektedir.
* Bu sayıyı parent process'e iletmek için message queue kullanmaktadır.

1. Parent Process'in Görevi:

* Ana process, tüm child process’lerin bitmesini beklemektedir.
* Tüm child process’lerden alınan en büyük k. sayıları birleştirmekte ve sıralamaktadır.
* Sonuçları çıkış dosyasına yazmaktadır.

Message Queue'ların Temizlenmesi:

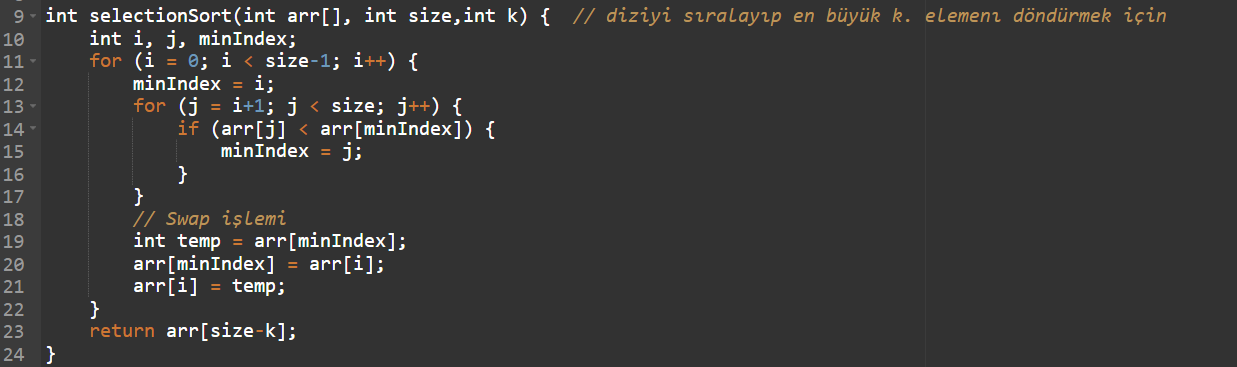
* Tüm işlemler tamamlandığında, program oluşturulan message queue'ları kapatmalı ve silmektedir.



Şekil 9

Şekil 9’da gerekli olan kütüphaneler eklenmektedir.

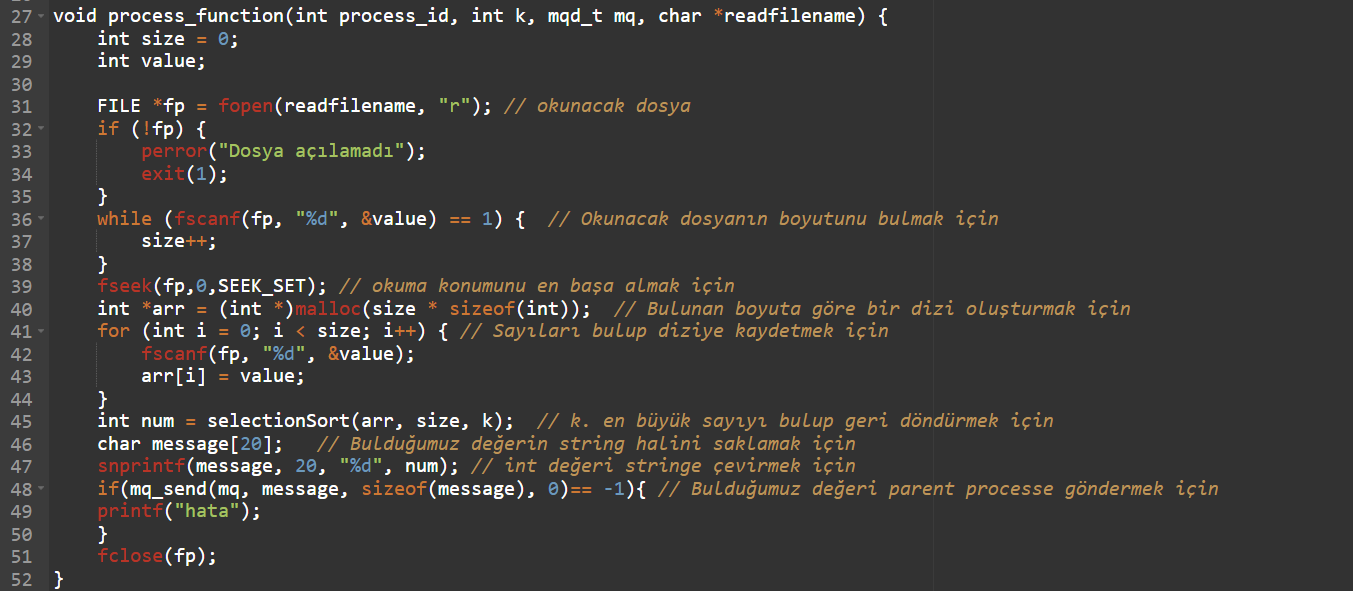
1. selectionsort



Şekil 10

Şekil 10’daki fonksiyon selection sort algoritmasıdır. Fakat bir dönüş değeri olarak bize dizideki en büyük k. elemanı döndürmektedir.

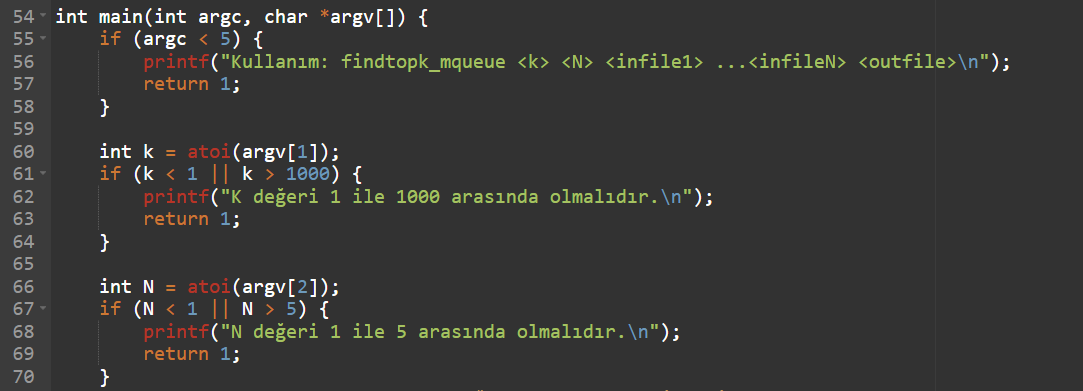
1. process\_function



Şekil 11

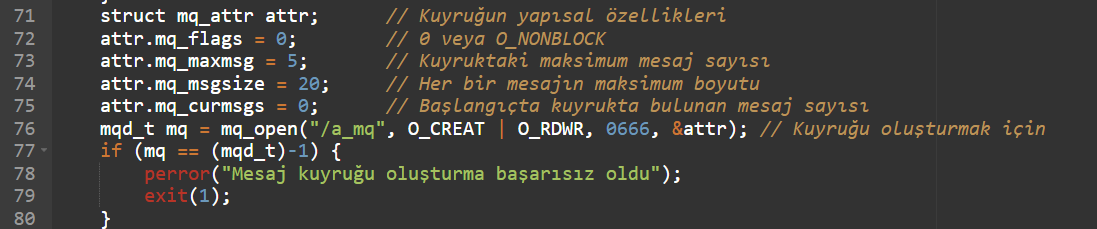
Şekil 11’deki fonksiyonda okunan dosyadaki satır sayısı bulunarak uygun bir dizi oluşturulmaktadır. Bu dizi şekil 10’daki fonksiyona gönderilerek en büyük k. değer bulunmaktadır. Bulunan değerler message queue ile parent process’e gönderilmektedir.

1. main



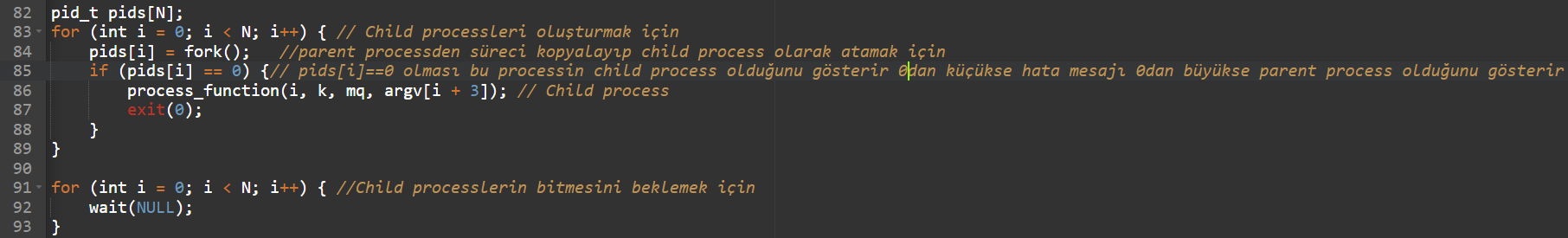
Şekil 12

Şekil 12’deki kod bloğunda parametrelerin kontrolü yapılmaktadır.



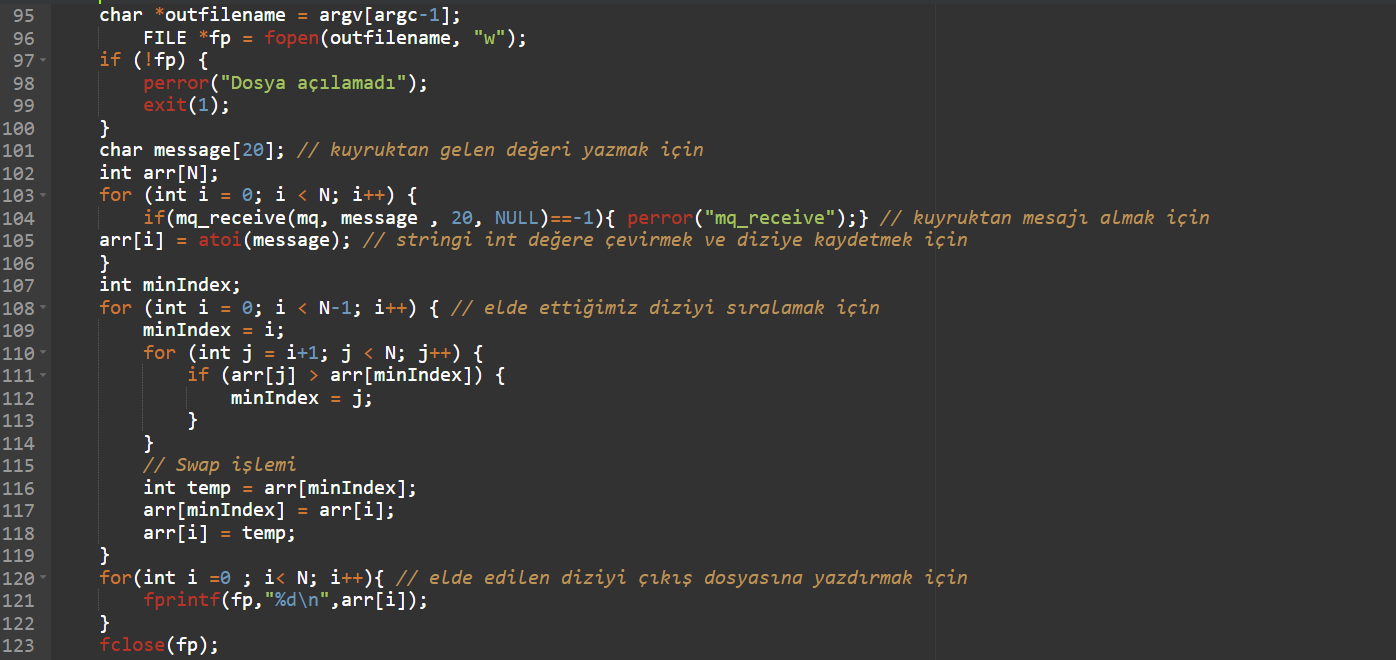
Şekil 13

Şekil 13’teki kod bloğunda kuyruğun yapısal özellikleri belirlenmekte ve kuyruk oluşturulmaktadır.



Şekil 14

Şekil 14’teki kod bloğu child process’leri oluşturup parent process’den sürecin kopyalanması sağlanmaktadır. Sonrasında ise child process’lerin işlemleri bitene kadar program beklemektedir.



Şekil 15

Şekil 15’deki kod bloğu message queue’ den k. en büyük değeri alıp arr dizisine kaydetmektedir. Daha sonra elde ettiğimiz arr dizisini sıralayıp çıktı dosyasına yazmaktadır.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Şekil 16

Şekil 16’daki kod bloğu oluşturulan kuyrukları kapatıp silmektedir.

PART 3

Part 3’te, her bir girdi dosyası için ayrı bir thread oluşturarak paralel işlemler kullanarak en büyük k. sayıları bulma işlemi gerçekleştirmektedir. Global değişkenler veya thread özel veri yapıları aracılığıyla thread'ler arasında bilgi paylaşımı yapmaktadır. İşlem adımları şu şekilde özetlenebilir:

1. Komut Satırı Parametrelerini Alma:

* Program, komut satırından alınan parametreleri kullanarak çalışmalıdır. Bu parametreler arasında k (1 ile 1000 arasında), N (1 ile 5 arasında), giriş dosyaları ve çıkış dosyası bulunmaktadır.

1. Thread Oluşturma:

* Program, belirtilen sayıda (N) POSIX thread oluşturmalıdır. Her bir thread, kendi görevini yerine getirmek üzere tasarlanmıştır.

1. Her Thread'in Görevi:

* Her bir thread, kendisine atanmış olan giriş dosyasını okuyarak en büyük k. sayıyı belirlemektedir.
* En büyük k. sayıyı bir thread global değişkenine (veya bir thread özel veri yapısına) kaydetmektedir.

1. Thread Sonuçlarını Birleştirme:

* Thread'lerin çalışması bittiğinde, ana thread bu thread'lerden alınan en büyük k. sayıları birleştirmekte ve sıralamaktadır.

1. Sonuçları Çıkış Dosyasına Yazma:

* Elde edilen sıralı en büyük k. sayıları çıkış dosyasına yazmaktadır.

metin, yazı tipi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

Şekil 17

Şekil 17’de gerekli olan kütüphaneler eklenmektedir.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Şekil 18

Şekil 18’de global değişkenler ve fonksiyon imzaları eklenmiştir.

1. selectionsort

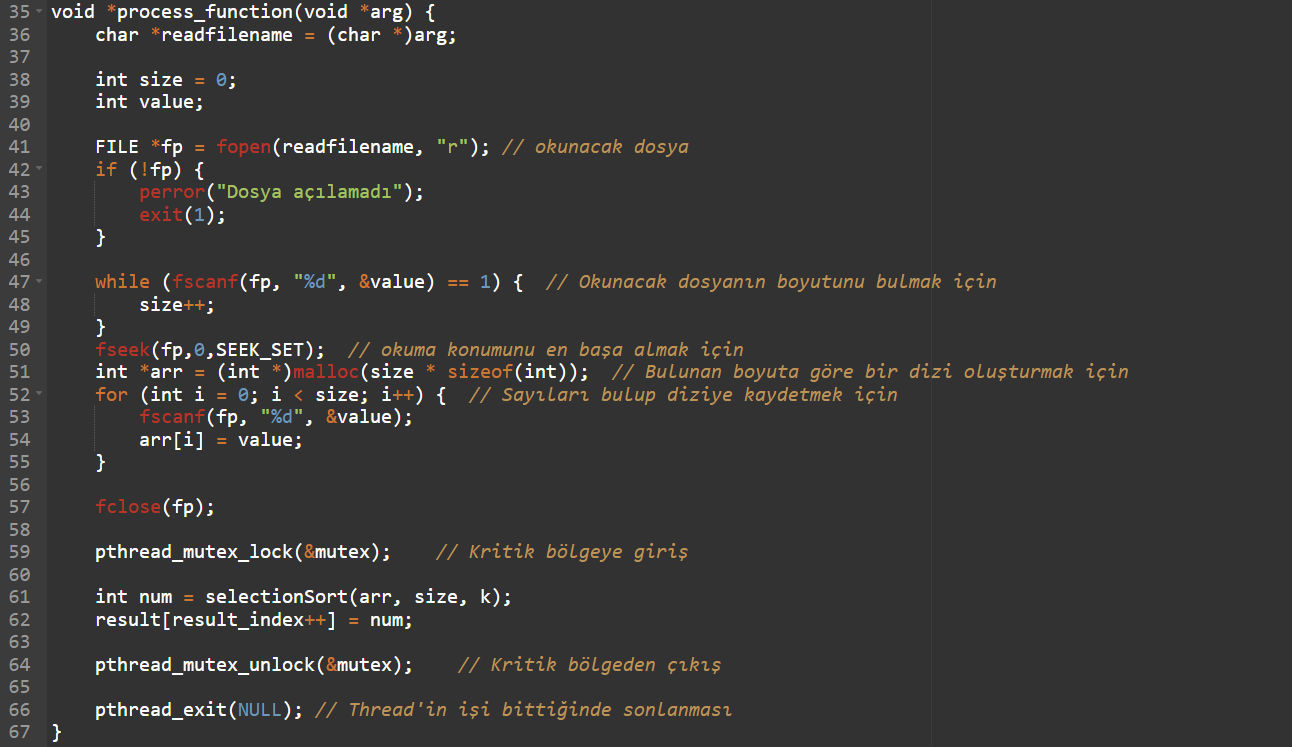
metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Şekil 20

Şekil 20’deki fonksiyon selection sort algoritmasıdır. Fakat bir dönüş değeri olarak bize dizideki en büyük k. elemanı döndürmektedir.

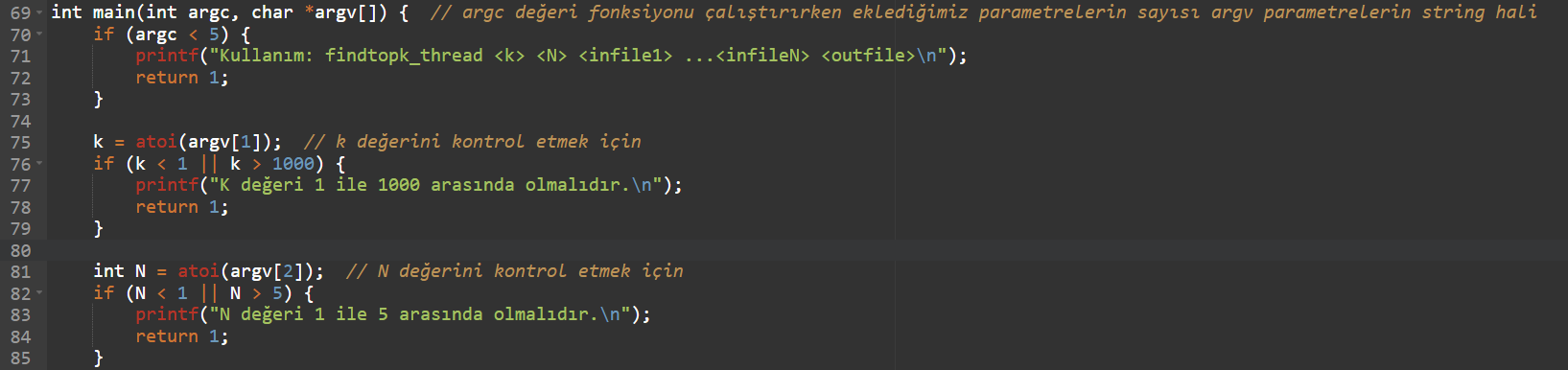
1. process\_function



Şekil 21

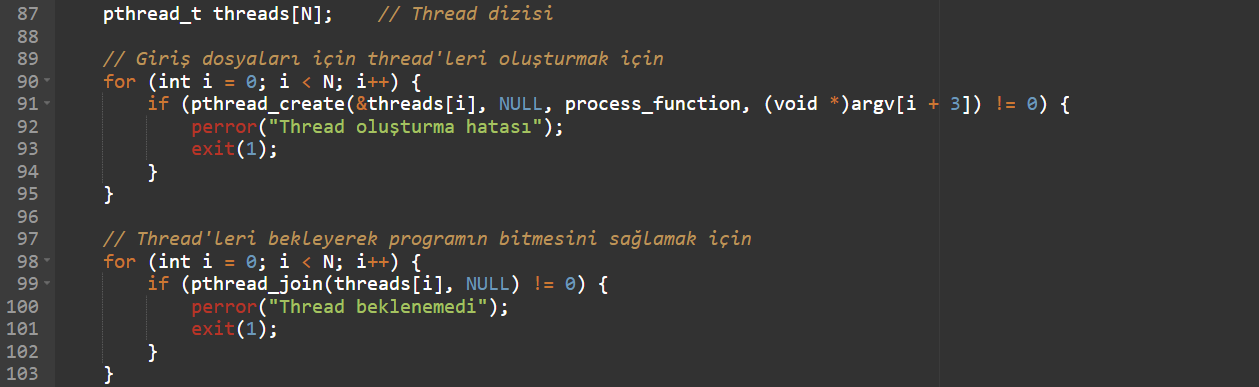
Şekil 21’deki fonksiyon, thread’lerin her birinde çalışan fonksiyondur. Fonksiyon, öncelikle okunacak dosyayı açar ve dosya boyutunu bulmaktadır. Ardından, bulunan boyuta göre bir dizi oluşturur ve dosyayı okuyarak diziyi doldurmaktadır. Son olarak, dizideki k. en büyük sayıyı bulur ve bunu global bir değişkene kaydetmektedir.

1. main



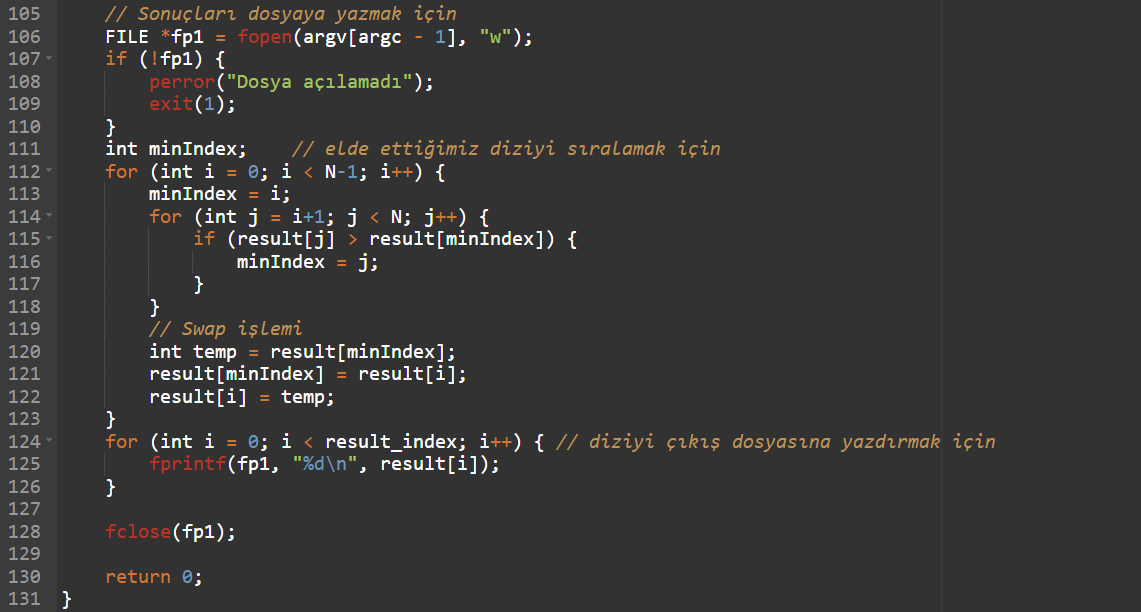
Şekil 22

Şekil 22’deki kod bloğunda parametrelerin kontrolü yapılmaktadır.



Şekil 23

Şekil 23’deki kod bloğu her bir dosya için thread oluşturarak process’i thread’lere göndermektedir. Sonrasında ise thread’leri bekleyerek programın bitmesini sağlamaktadır.



Şekil 24

Şekil 24’teki kod bloğu elde edilen result dizisi sıralanarak çıkış dosyasına yazdırma işlemi yapılmaktadır.

PART 4

Part 4'te, findtopk, findtopk\_mqueue ve findtopk\_thread programlarının performansını, çeşitli k, N ve dosya boyutları için incelenmektedir.

Tablo 1: Programların N Değerine Göre Karşılaştırma Tablosu

metin, ekran görüntüsü, sayı, numara, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Tablo 1’de dosya boyutu ve k değeri sabit tutularak N=1 ve N=5 değerleri için karşılaştırma yapılmaktadır. (k=500, dosya ise 100.000 sayıdan oluşmaktadır.)

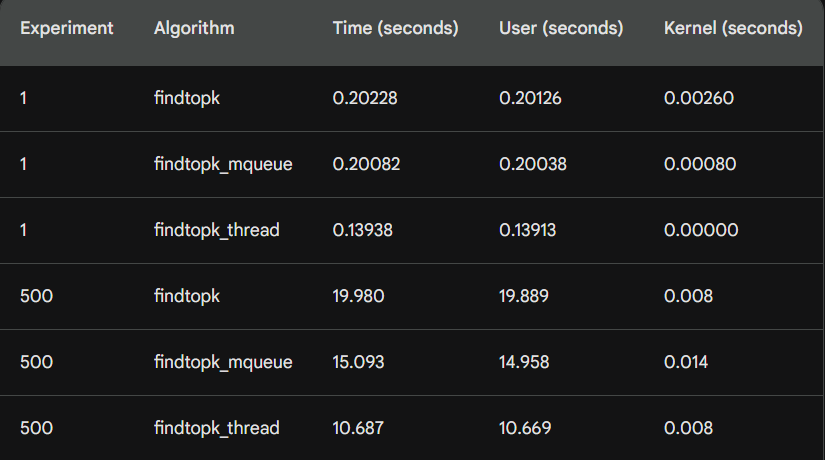
Tablo 2: Programların Dosyanın Boyut Değerine Göre Karşılaştırma Tablosu

metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

Tablo 2’de k ve N değerleri sabit tutularak dosya boyutu kıyaslanmaktadır. Dosya boyutları 1000 sayıdan ve 1000000 sayıdan oluşmaktadır. (k=500, N=1)

Tablo 3: Programların k Değerine Göre Karşılaştırma Tablosu



Tablo 3’te N değeri ve dosya boyutu sabit tutularak k=1 ve k=500 değerleri için karşılaştırma yapılmaktadır. (N=1, dosya ise 100.000 sayıdan oluşmaktadır.)

Yapılan deneyler sonucunda, findtopk\_thread programının, findtopk ve findtopk\_mqueue programlarına göre daha hızlı olduğu görülmüştür. Bu durum, findtopk\_thread programının, findtopk programının kullandığı intermediate dosyaları ve findtopk\_mqueue programının kullandığı message queue'leri kullanmamasından kaynaklanmaktadır.

findtopk\_thread programının çalışma süresi, N değerine bağlı olarak değişmektedir. N değeri arttıkça, findtopk\_thread programının çalışma süresi de artmaktadır. Bu durum, findtopk\_thread programının, N adet thread oluşturması ve bu thread'leri koordine etmesi ile ilgilidir.

findtopk\_mqueue programının çalışma süresi, N değerine bağlı olarak daha az değişmektedir. Bu durum, findtopk\_mqueue programının, N adet thread oluşturmasına gerek duymamasından kaynaklanmaktadır.

findtopk programının çalışma süresi, N değerine bağlı olarak önemli ölçüde artmaktadır. Bu durum, findtopk programının, N adet intermediate dosyayı okuması ve işlemesi ile ilgilidir.

K değerinin çalışma süresine etkisi, N değerine göre daha azdır. K değeri arttıkça, findtopk ve findtopk\_mqueue programlarının çalışma süresi artmaktadır.

Sayıların boyutunun çalışma süresine etkisi, N ve K değerlerine göre daha fazladır. Sayıların boyutu arttıkça, findtopk, findtopk\_thread ve findtopk\_mqueue programlarının çalışma süresini arttırmaktadır. Bu durum, findtopk, findtopk\_thread ve findtopk\_mqueue programlarının, daha fazla sayıyı okumaları ve işlemeleri ile ilgilidir.

Bu sonuçlara göre, findtopk\_thread programı, findtopk ve findtopk\_mqueue programlarına göre daha hızlıdır. Bu durum, findtopk\_thread programının, intermediate dosyaları ve message queue'leri kullanmamasından kaynaklanmaktadır.