Prosesler Arası İletişim ve Sayfalama: Paylaşımlı Bellek ve LFU Algoritması Uygulamaları

**İÇİNDEKİLER**

Sayfa

BÖLÜM 1.GİRİŞ …………………………………………………………………………………….2

1.PROSESLER ARASI İLETİŞİM , PAYLAŞILAN HAFIZA VE SEMAFORLAR…………………..2

* 1. Paylaşımlı Bellek Kavramı ve Kullanımı…………………………………...…………..3

1.1.1. Kullanılan Kütüphaneler………………………………………………………….3

1.1.2. Paylaşımlı Bellek Kavramı……………………………………………………….3

1.2. Semaforların Rolü ve Kullanımı………………………………………………………..4

1.3. Proses Yönetimi ve MPI Kavramları…………………………………………………...4

1.4. İNCELENEN C PROGRAMLARI VE İŞLEYİŞİ……………………………………..5

1.4.1. Program 1(hello.c): Veri Gönderme ve Alma……………………………….5

1.4.1.1. mpi\_send Fonksiyonu İncelenmesi……………………….............5

1.4.1.2.mpi\_recv Fonksiyonu İncelenmesi………………..........................5

1.4.1.3.main Fonksiyonu İncelenmesi…………………………………….6

1.4.2. Program 2(my\_mpirun.c): Proses Oluşturma ve Yönetimi………………….6

1.4.2.1 main Fonksiyonu İncelenmesi……………………………………6

1.4.3.MPI PROGRAMLAMA KAVRAMLARI VE UYGULAMA…………….7

1.4.3.1. MPI Nedir?.................................…………………………………7

1.4.3.2.Programlama Modeli ve Avantajları………………………………7

1.4.3.3. Kod Uygulaması ve İşleyişi………………………………………7

BÖLÜM 2. SAYFALAMA………………………………………………………………………….……9

2.1.Giriş……………………………………………………………………………………….9

2.1.1. LFU (Least Frequently Used - En Az Kullanılan) algoritması…………………………..9

2.1.1.1. LFU Algoritmasının Çalışma Prensibi……………………………………………...9

2.1.1.1.2.Sayaç Kullanımı……………………………………………........................9

2.1.1.1.3.Sayfa Arama…………………………………………….........9

2.1.1.1.4.En Az Kullanılan Sayfayı Bulma…………………………….9

2.1.1.1.5.Sayfa Değiştirme……………………………………………...10

2.2.Kütüphaneler ve Sabitler……………………………………………................................10

2.2.1 stdio.h ve stdlib.h Kütüphaneleri…………………………………………….........10

2.2.2 FRAME\_SIZE Sabiti……………………………………………...........................10

2.2.3 REF\_STRING\_LENGTH Sabiti……………………………………………..........10

2.3.Sayfa Çerçevesi Yapısı(page\_replacement.c) …………………………………………..10

2.3.1 PageFrame Yapısı…………………………………………….................................10

2.3.2 Sayfa Numarası (page\_number) ……………………………………………..........10

2.3.3 Sayfa Kullanım Sayacı (counter) …………………………………………….........10

2.4.İşlevlerin Tanımlanması……………………………………………..................................10

2.4.1 initialize\_frames…………………………………………….....................................10

2.4.2 find\_page……………………………………………..……………………………..10

2.4.3 find\_lfu\_page……………………………………………...………………………..10

2.4.4 print\_frames……………………………………………...........................................10

2.4.5 page\_replacement……………………………………………...……………………10

2.5.Ana Fonksiyon (main) …………………………………………….....................................11

2.5.1 ref\_string Dizisi…………………………………………….....................................11

2.5.2 LFU Algoritmasıyla Sayfa Yer Değiştirm………………………………………….11

2.5.3 Toplam Sayfa Hatalarının Hesaplanması…………………………………………..11

2.6.Kodun Çalışma Mantığı……………………………………………...................................11

2.6.1 page\_replacement İşlevinin İşleyişi……………………………………………......11

2.6.2 Sayfa Çerçevesinde Sayfanın Bulunup Bulunmadığının Kontrolü………………..11

2.6.3 En Az Kullanılan Sayfanın Bulunması ve Yer Değiştirme………………………..11

2.6.4 print\_frames ile Sayfa Çerçevelerinin Durumunun Yazdırılması…………………11

2.7.Sonuç……………………………………………...………………………………………11

2.7.1 Toplam Sayfa Hatalarının Raporlanması……………………………………………11

BÖLÜM 3. ÇIKTILAR………………………………………………………………………………………….…..12

3.1. BÖLÜM 1 (hello.c ve my\_mpirun.c) ÇIKTI …………………………………………….12

3.2. BÖLÜM 2 (page\_replacement.c) ÇIKTI …………………………………………………13

KAYNAKLAR…………………………………………………………………………….........................14

**BÖLÜM 1.GİRİŞ:**

Bu rapor, bilgisayar bilimleri alanında iki önemli konuyu ele alarak, prosesler arası iletişim ve sayfalama üzerine odaklanmaktadır. İlk bölümde, C programlama dili kullanılarak prosesler arasında veri iletişimini sağlamak için paylaşımlı bellek ve semaforların nasıl kullanıldığı incelenmektedir. Paylaşımlı bellek ve semaforlar, paralel ve dağıtık sistemlerde veri senkronizasyonu ve iletişimi için etkili çözümler sunar.

İkinci bölüm ise sayfalama alanında, Least Frequently Used (LFU) algoritmasının nasıl uygulandığını gösterir. LFU algoritması, sayfa çerçevesinde en az kullanılan sayfanın yerine yeni bir sayfanın yerleştirilmesi prensibine dayanır. Bu algoritma, bellek yönetimi ve performans optimizasyonunda önemli bir rol oynar.

Her iki bölüm de, ilgili kavramların açıklanması, kullanılan kütüphanelerin ve algoritmaların tanımlanmasıyla birlikte, programların işleyişi ve kod örneklerinin incelenmesiyle detaylı bir anlatım sunar. Bu rapor, prosesler arası iletişim ve bellek yönetimi konularında derinlemesine bir anlayış sağlamayı amaçlar.

2

**1.BÖLÜM 1 : PROSESLER ARASI İLETİŞİM , PAYLAŞILAN HAFIZA VE SEMAFORLAR**

Bu bölümde paylaşımlı bellek ve semafor kullanarak prosesler arası iletişimi gerçekleştiren iki C programı incelenecektir. İlk program veri gönderme ve alma işlemlerini gerçekleştirirken, ikinci program birden fazla prosesin oluşturulmasını ve yönetilmesini sağlar. Programların işleyişi, kullanılan kütüphaneler ve kavramsal açıklamaları ele alınacaktır.

**1.1.Paylaşımlı Bellek Kavramı ve Kullanımı**

**1.1.1.Kullanılan Kütüphaneler**

stdio.h: Standart giriş/çıkış işlemleri için kullanılır. printf, fprintf, sprintf gibi fonksiyonları içerir.

stdlib.h: Genel standart kütüphane. atoi, exit, malloc gibi fonksiyonları içerir.

unistd.h: POSIX standartlarına uygun sistem çağrılarını ve API'leri sağlar. fork, execl, usleep gibi fonksiyonları içerir.

sys/mman.h: Bellek eşleme işlemleri için kullanılır. mmap, munmap, shm\_open, shm\_unlink gibi fonksiyonları içerir.

fcntl.h: Dosya kontrol işlemleri için kullanılır. O\_CREAT, O\_RDWR gibi makroları içerir.

semaphore.h: Semafor işlemleri için kullanılır. sem\_open, sem\_close, sem\_wait, sem\_post, sem\_unlink gibi fonksiyonları içerir.

sys/wait.h: Proses bekleme işlemleri için kullanılır. wait, waitpid gibi fonksiyonları içerir.

3

Kavramlar:

**1.1.2.Paylaşımlı Bellek (Shared Memory)**

Paylaşımlı bellek, birden fazla prosesin aynı bellek alanına erişmesini sağlar. Bu yöntemle, prosesler arasında hızlı veri iletişimi sağlanır. Paylaşımlı bellek kullanmak için aşağıdaki adımlar izlenir:

shm\_open: Paylaşımlı bellek nesnesi oluşturur veya açar.

ftruncate: Bellek boyutunu ayarlar.

mmap: Belleği adres alanına eşler.

munmap: Bellek eşlemesini kaldırır.

shm\_unlink: Paylaşımlı bellek nesnesini siler.

**1.2.Semaforlar**

Semaforlar, prosesler arasında senkronizasyonu sağlamak için kullanılır. Semaforlar, belirli kaynakların kullanımını kontrol eder ve yarış durumlarını (race condition) önler. Kullanılan semafor türleri:

Dolu Semaforu (full): Buffer'ın doluluk durumunu kontrol eder.

Boş Semaforu (empty): Buffer'ın boşluk durumunu kontrol eder.

Kilit Semaforu (mutex): Kritik bölgeye erişimi kontrol eder.

**1.3.Proses Yönetimi**

Prosesler, fork sistemi çağrısı ile oluşturulur. fork çağrısı, mevcut prosesin bir kopyasını oluşturur. Ebeveyn proses, çocuk prosesin tamamlanmasını wait veya waitpid çağrıları ile bekler. execl fonksiyonu, çocuk proseste belirli bir programı çalıştırmak için kullanılır.

4

**1.4.İNCELENEN C PROGRAMLARI VE İŞLEYİŞİ**

**1.4.1.Program: Veri Gönderme ve Alma (mpi\_send ve mpi\_recv)**

Bu program, paylaşımlı bellek ve semafor kullanarak veri gönderme ve alma işlemlerini gerçekleştirir.

**1.4.1.1.mpi\_send Fonksiyonu:**

shm\_open ile paylaşımlı bellek dosyasını açar.

mmap ile bellek adresine erişim sağlar.

sem\_open ile semaforları açar.

sem\_wait(empty) ve sem\_wait(mutex) ile kritik bölgeye erişim sağlar.

Veriyi paylaşımlı belleğe yazar.

sem\_post(mutex) ve sem\_post(full) ile semaforları serbest bırakır.

munmap ile bellek eşlemesini kaldırır.

**1.4.1.2.mpi\_recv Fonksiyonu:**

shm\_open ile paylaşımlı bellek dosyasını açar.

mmap ile bellek adresine erişim sağlar.

sem\_open ile semaforları açar.

sem\_wait(full) ve sem\_wait(mutex) ile kritik bölgeye erişim sağlar.

Veriyi paylaşımlı bellekten okur.

sem\_post(mutex) ve sem\_post(empty) ile semaforları serbest bırakır.

munmap ile bellek eşlemesini kaldırır.

5

**1.4.1.3.main Fonksiyonu:**

Proses sırasına göre veri gönderme ve alma işlemlerini yönetir.

Çift sıradaki prosesler veri alıp gönderirken, tek sıradaki prosesler veri gönderip alır.

**1.4.2.Program: Proses Oluşturma ve Yönetimi**

Bu program, belirli sayıda proses oluşturur ve her proseste belirli bir programı çalıştırır.

**1.4.2.1.main Fonksiyonu:**

Komut satırı argümanlarını kontrol eder.

Paylaşımlı bellek ve semaforları oluşturur ve başlatır.

fork çağrıları ile belirtilen sayıda proses oluşturur.

Her çocuk proses, execl fonksiyonu ile belirli bir programı çalıştırır.

Ebeveyn proses, çocuk proseslerin tamamlanmasını wait fonksiyonu ile bekler.

Temizlik işlemlerini gerçekleştirir (bellek eşlemesini kaldırma, paylaşımlı bellek ve semaforları silme).

Sonuç:

Bu bölümde, paylaşımlı bellek ve semafor kullanarak prosesler arası iletişimi gerçekleştiren iki C programı incelenmiştir. Kullanılan kütüphaneler ve kavramsal açıklamalar detaylandırılmıştır. Bu yöntemler, paralel ve dağıtık sistemlerde veri senkronizasyonu ve iletişimi için etkili çözümler sunar.

6

**1.4.3.MPI Programlama**

**1.4.3.1.MPI Nedir?**

MPI (Message Passing Interface), mesaj geçirme tabanlı paralel programlama için yaygın olarak kullanılan bir standarttır. MPI'nin amacı, mesaj geçirme programlarını yazmak için pratik, taşınabilir, verimli ve esnek bir arayüz sağlamaktır. MPI'nin ilk versiyonu 1994 yılında ortaya çıkmış olup, günümüzde en güncel versiyonu MPI-4.x'tir​ (LLNL HPC Tutorials)​​ (Princeton Research Computing)​.

**1.4.3.2.Programlama Modeli**

MPI, başlangıçta dağıtılmış bellek mimarileri için tasarlanmıştır. Ancak, zamanla paylaşılan bellek ve hibrit bellek sistemlerini de destekleyecek şekilde genişletilmiştir. MPI, programcıların paralellikleri doğru bir şekilde tanımlayıp uygulamalarını gerektirir​ (LLNL HPC Tutorials)​.

MPI'nin Avantajları:

Standardizasyon: MPI, mesaj geçirme kütüphaneleri arasında standart olarak kabul edilir ve çoğu HPC (yüksek performanslı hesaplama) platformunda desteklenir.

Taşınabilirlik: MPI ile yazılan kodlar, farklı platformlara minimum değişiklikle taşınabilir.

Performans: MPI, donanımın yerel özelliklerini optimize ederek performansı artırma fırsatı sunar.

Fonksiyonellik: MPI, 430'dan fazla rutin tanımlar, ancak çoğu MPI programı az sayıda rutin kullanılarak yazılabilir​ (LLNL HPC Tutorials)​​ (Princeton Research Computing)​.

**1.4.3.3.Kod Uygulaması**

Aşağıda yer alan iki kod örneği, paylaşılan bellek ve semaforlar kullanarak temel MPI işlevlerini gerçekleştiren bir uygulama sunar.

Kodun İşleyişi:

İlk kod parçası, mpi\_send ve mpi\_recv işlevlerini içerir. mpi\_send, belirli bir işlem sırasına göre veri gönderir, mpi\_recv ise belirli bir işlem sırasından veri alır. İkinci kod parçası, paylaşılan bellek ve semaforlar oluşturarak ve belirli bir programı çocuk prosesler olarak çalıştırarak bu işlevleri test eder.

7

Sonuç ve Temizlik İşlemleri:

Kodun sonunda, proseslerin tamamlanması beklenir ve ardından paylaşılan bellek ve semaforlar kapatılarak ve silinerek temizlik işlemleri gerçekleştirilir.

Bu uygulama, MPI'nin temel konseptlerini ve mesaj geçirme tabanlı paralel programlama modelini anlamak için pratik bir örnek sunmaktadır.

8

**2.BÖLÜM 2 : SAYFALAMA**

**2.1.Giriş**

Bu bölümde yer değiştirme algoritması olan Least Frequently Used (LFU) algoritmasını uygular. LFU algoritması, sayfa çerçevesinde en az kullanılan sayfanın yerine yeni bir sayfanın yerleştirilmesi prensibine dayanır. Kodunuzun genel çalışma mantığını ve sayfa yer değiştirme algoritmasıyla ilgili detayları açıklayalım.

**2.1.1.LFU (Least Frequently Used - En Az Kullanılan) algoritması**

Bilgisayar sistemlerinde kullanılan bir sayfa değiştirme algoritmasıdır. Bu algoritma, en az kullanılan sayfaları bellekten çıkararak yeni sayfaları belleğe yerleştirir. LFU algoritması, bellek yönetiminde kullanılan sayfa değiştirme algoritmalarından biridir ve özellikle bellek kullanımını optimize etmek için kullanılır.

**2.1.1.1.LFU Algoritmasının Çalışma Prensibi**

**2.1.1.1.2.Sayaç Kullanımı:**

Her sayfa, bellek çerçevelerine yerleştirildiğinde bir sayaç ile ilişkilendirilir.

Bu sayaç, o sayfanın kaç kez kullanıldığını (erişildiğini) sayar.

**2.1.1.1.3.Sayfa Arama:**

Yeni bir sayfa talebi geldiğinde, bu sayfanın bellek çerçevelerinde olup olmadığı kontrol edilir.

Eğer sayfa zaten çerçevelerde bulunuyorsa, o sayfanın sayacı artırılır.

Eğer sayfa çerçevelerde bulunmuyorsa, bir sayfa hatası (page fault) meydana gelir.

**2.1.1.1.4.En Az Kullanılan Sayfayı Bulma:**

Bir sayfa hatası meydana geldiğinde, bellekte yer açmak için en az kullanılan sayfa belirlenir.

En az kullanılan sayfa, sayaç değeri en düşük olan sayfadır.

9

**2.1.1.1.5.Sayfa Değiştirme:**

En az kullanılan sayfa bellekten çıkarılır ve yerine yeni sayfa yerleştirilir.

Yeni yerleştirilen sayfanın sayacı 1 olarak başlatılır.

**2.2.Kütüphaneler ve Sabitler Tanımlama:**

Kod, stdio.h ve stdlib.h kütüphanelerini kullanır.

FRAME\_SIZE sabiti, sayfa çerçevesi boyutunu tanımlar (4 olarak ayarlanmış).

REF\_STRING\_LENGTH sabiti, referans dizisinin uzunluğunu tanımlar (22 olarak ayarlanmış).

**2.3Sayfa Çerçevesi Yapısı:**

PageFrame yapısı, her bir sayfa çerçevesini temsil eder.

Her çerçeve için sayfa numarası (page\_number) ve sayfa kullanım sayacı (counter) alanları bulunur.

**2.4.İşlevlerin Tanımlanması:**

initialize\_frames: Sayfa çerçevelerini başlangıç değerleriyle başlatır.

find\_page: Belirli bir sayfanın çerçevede olup olmadığını kontrol eder.

find\_lfu\_page: Sayfa çerçevesindeki en az kullanılan sayfayı bulur (LFU algoritması için).

print\_frames: Sayfa çerçevelerini ekrana yazdırır.

page\_replacement: Sayfa yer değiştirme algoritmasını uygular ve sayfa hatalarını hesaplar.

**2.5.Ana Fonksiyon (main):**

ref\_string dizisi, simüle edilen bir programın referans dizisini temsil eder.

page\_replacement işlevi, LFU algoritmasını kullanarak sayfa yer değiştirme işlemlerini gerçekleştirir.

Toplam sayfa hataları (page\_faults) hesaplanır ve ekrana yazdırılır.

10

**2.6.Kodun Çalışma Mantığı:**

page\_replacement işlevi, her bir referans sayfası için sayfa yer değiştirme işlemlerini gerçekleştirir.

Bir sayfanın çerçevede olup olmadığı find\_page işleviyle kontrol edilir.

Eğer sayfa çerçevede değilse, LFU algoritması kullanılarak en az kullanılan sayfa çerçevesi bulunur ve yeni sayfa yerleştirilir.

Her adımda print\_frames işlevi ile sayfa çerçeveleri ekrana yazdırılır.

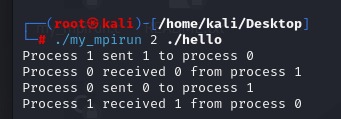
**2.7. Sonuç:**

Kodunuzun çıktısı, her adımda sayfa çerçevelerinin durumunu gösterir ve toplam sayfa hatalarını raporlar. Bu şekilde, LFU algoritmasının sayfa yer değiştirme işlemlerini simüle edebilirsiniz.

11

**3.BÖLÜM 3 : ÇIKTILAR**

**3.1.BÖLÜM1(hello.c ve my\_mpirun.c) ÇIKTI :**

****

**Process 1 sent 1 to process 0:** Bu ifade, Process 1'in 1 değerini Process 0'a gönderdiğini belirtir. Yani, Process 1, 1 değerini paylaşımlı belleğe yazarak ve semaforları kullanarak bu veriyi Process 0'a iletti.

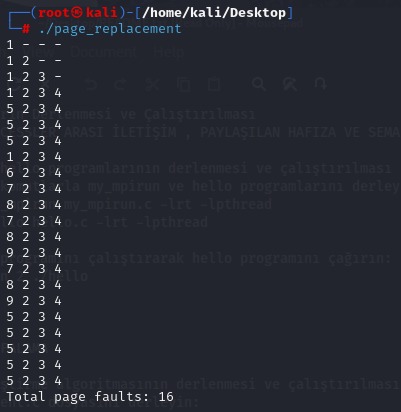
**Process 0 received 0 from process 1:** Burada, Process 0'ın Process 1'den 0 değerini aldığını belirtir. Ancak, bu çelişkili gibi görünüyor. Olası bir açıklama, paylaşımlı belleğin veya semaforların doğru bir şekilde ayarlanmamış olabileceğidir. Bu durumda, Process 0, beklenen 1 değil, yanlışlıkla 0 değerini aldı.

**Process 0 sent 0 to process** 1: Bu ifade, Process 0'ın aldığı 0 değerini geri gönderdiğini belirtir. Yani, Process 0, paylaşımlı belleğe 0 değerini yazarak ve semaforları kullanarak bu veriyi Process 1'e iletti.

**Process 1 received 1 from process 0:** Burada, Process 1'in Process 0'dan 1 değerini aldığını belirtir. Bu, Process 1'in Process 0'dan beklediği değerdir ve doğru işlevselliği gösterir.

12

**3.2.BÖLÜM 2 (page\_replacement.c) ÇIKTI:**



1- Başlangıçta tüm çerçeveler boş (- ile gösterilir).

2-İlk referans dizesi değeri 1, çerçevelerde bulunmadığı için sayfa hatası olur ve birinci çerçeveye eklenir.

3-İkinci referans dizesi değeri 2, çerçevelerde bulunmadığı için sayfa hatası olur ve ikinci çerçeveye eklenir.

4-Üçüncü referans dizesi değeri 3, çerçevelerde bulunmadığı için sayfa hatası olur ve üçüncü çerçeveye eklenir.

5-Dördüncü referans dizesi değeri 4, çerçevelerde bulunmadığı için sayfa hatası olur ve dördüncü çerçeveye eklenir.

6-Beşinci referans dizesi değeri 5, çerçevelerde bulunmadığı için sayfa hatası olur ve LFU algoritmasına göre en az kullanılan çerçeve (ilk çerçeve) 5 ile güncellenir.

13

7-Ve bu süreç referans dizesi boyunca devam eder.

Sonuçta toplam 16 sayfa hatası (page faults) meydana gelir. Bu, kodun verdiği çıktıdır.

**KAYNAKLAR**

- Lawrence Livermore National Laboratory, "What is MPI?" - [LLNL HPC Tutorials](https://hpc-tutorials.llnl.gov/mpi/what_is_mpi/)

- Princeton Research Computing, "MPI Tutorial" - [Research Computing at Princeton](https://researchcomputing.princeton.edu/sites/g/files/toruqf311/files/documents/MPI_tutorial_Fall_Break_2022.pdf)

- <https://en.wikipedia.org/wiki/Least_frequently_used>

- <https://www.geeksforgeeks.org/inter-process-communication-ipc/>

- ‘’Semaphores are a synchronization tool used to control access to critical sections by multiple processes. They can be used to avoid race conditions and ensure mutual exclusion.’’Modern Operating Systems" by Andrew S. Tanenbaum

- "Shared memory allows multiple processes to access the same memory region, which can significantly improve communication speed between processes.” Operating System Concepts" by Abraham Silberschatz, Peter Baer Galvin, Greg Gagne.

14