OS-codes

****

***** lek1-2: *****

****

1.2 islemciler:

"Register", bilgisayar mimarisinde, işlemcinin veri ve komutları geçici olarak sakladığı küçük hafıza bölgeleridir. İşlemci, bu kayıtları kullanarak veri manipülasyonu ve işlem yapar.

Register'lar, işlemcinin hızlı erişebildiği ve kullanabildiği en hızlı bellek türüdür. Bir işlemcinin içinde bulunan register sayısı ve özellikleri, işlemcinin tasarımına ve mimarisine bağlı olarak değişir.

Register'lar genellikle genel amaçlı (general-purpose) veya özel amaçlı (special-purpose) olarak sınıflandırılır.

Genel amaçlı register'lar, işlemcinin genel veri işleme işlevleri için kullanılır. Örneğin, işlem yapılacak verilerin saklanması, işlem sonuçlarının geçici olarak tutulması gibi işlemler için kullanılabilirler.

Özel amaçlı register'lar ise belirli işlevlere veya özelliklere sahiptir. Örneğin, yönlendirme işlemleri için kullanılan "program counter" (PC) ve "instruction register" (IR) gibi özel register'lar bulunmaktadır.

Register'lar, işlemci performansını artırmak için önemli bir rol oynarlar çünkü işlemci, verilere register'lar aracılığıyla çok hızlı bir şekilde erişebilir ve manipüle edebilir.

Program sayacı (Program Counter veya Instruction Pointer), işlemcinin mevcut olarak yürütmekte olduğu komutun bellekteki adresini işaret eden bir işlemci kaydıdır. Program sayacı, işlemci komutlarını sırayla işlemek için kullanılır. Her bir komutun bellekteki adresi, bir sonraki komutun adresini içerecek şekilde artırılır ve program sayacı bu adresi izler.

Bellek işaretçisi (Memory Address Register), işlemcinin bellekteki veri veya komutları erişmek için kullanılan bir diğer kaydıdır. Bellek işaretçisi, işlemcinin bellek üzerindeki adreslerle iletişim kurmasına ve veri okuma/yazma işlemlerini gerçekleştirmesine yardımcı olur.

Örneğin, bir veriyi belleğe yazmak istediğinizde, bellek işaretçisi ilgili bellek adresini içerecek şekilde ayarlanır ve yazma işlemi gerçekleştirilir.

Bu iki kayıt, işlemcinin komutları işleyebilmesi ve veriye erişebilmesi için temel öneme sahiptir. Program sayacı, işlemcinin program akışını yönlendirirken, bellek işaretçisi veriye erişim için gerekli adres bilgisini sağlar.

Sistem veri yolu (System Bus), bir bilgisayar sisteminde çeşitli bileşenler arasında veri, adres ve kontrol sinyallerini taşıyan bir iletişim yoludur. Bu yol, işlemci, bellek, giriş/çıkış (I/O) cihazları ve diğer sistem bileşenleri arasında veri alışverişini sağlar.

Sistem veri yolu, genellikle üç ana bileşenden oluşur:

OS-codes

1. **Veri Yolu (Data Bus)**: Bu bileşen, veri aktarımını sağlar. İşlemci, bellek ve diğer cihazlar arasında veri taşınmasını mümkün kılar.

2. **Adres Yolu (Address Bus)**: Bu bileşen, bellek veya diğer cihazlara erişim için adres bilgisini taşır. İşlemci veya bir cihaz, bellek veya I/O cihazlarına erişmek için adres sinyali gönderir.

3. **Kontrol Yolu (Control Bus)**: Bu bileşen, veri ve adres sinyallerini yönetir. Okuma/yazma işlemlerinin, bellek erişiminin, kesmelerin ve diğer kontrol sinyallerinin yönetimini sağlar.

Bu bileşenler arasındaki veri yolu, veri transferini sağlar. Adres yolu, işlemcinin ve diğer bileşenlerin bellek ve I/O cihazlarına erişimini belirlerken, kontrol yolu ise bu işlemleri yönetir.

Birlikte, bu bileşenler, bilgisayar sistemlerinin temel iletişim altyapısını oluşturur ve veri akışını sağlar.

1.4 ოპერაციულ სისტემასთან დაკავშირებული ძირითადი ცნებები

Bir işlem (process), bilgisayar bilimlerinde çalışan bir programın çalışma sırasında sistem kaynaklarını (bellek, CPU zamanı, dosya ve giriş/çıkış cihazları gibi) kullanma sürecidir. İşlem, bir programın çalışması sırasında sistemdeki birimlerle etkileşime geçen ve kaynakları kullanarak belirli bir görevi yerine getiren bir varlıktır.

İşlem genellikle şu bileşenleri içerir:

1. **Program Kodu**: İşlemi başlatan programın kodu veya yürütülebilir dosyası.

2. **Bellek Alanı**: İşlem için ayrılmış bellek alanı. Bu alan, işlem tarafından kullanılan değişkenler, veri yapıları ve yürütme sırasında oluşturulan diğer bilgileri içerebilir.

3. **İşlem Kontrol Bilgileri**: İşlem durumu, öncelik, kaynakları tahsis etme bilgileri gibi işlemle ilgili kontrol bilgileri.

4. **Kaynaklar**: İşlem, sistem kaynaklarını kullanarak çalışır. Bunlar arasında bellek, işlemci zamanı, dosya ve cihazlar bulunabilir.

İşlem, bir kullanıcının bilgisayar üzerinde çalıştırdığı herhangi bir programı veya bir sistem hizmetini temsil edebilir. Her işlem, sistemde benzersiz bir kimlik (PID - Process Identifier) ile tanımlanır. İşletim sistemi, işlemleri yönetmek, zamanlamak, kaynakları tahsis etmek ve işlem arasında iletişim sağlamak gibi işlevleri gerçekleştirir.

Bir işlem, genellikle başlatılma, duraklatılma, devam ettirme ve sonlandırma gibi çeşitli durumlara geçebilir.

Adres uzayı, bir bilgisayarın belleğindeki her bir hücre veya byte için benzersiz bir adresin atanmış olduğu bir alanı ifade eder. Bilgisayarların bellekleri genellikle adreslenebilir bir yapıya sahiptir, yani her bir bellek hücresine veya veri birimine erişmek için bir adres kullanılır.

OS-codes

Adres uzayı, genellikle "byte" olarak adlandırılan en küçük bellek birimlerinden oluşur. Her bir byte, benzersiz bir adrese sahiptir. Bu adresler, bellek hücrelerini veya byte'ları tanımlamak için kullanılır ve bilgisayarın işlemci veya diğer donanım birimleri tarafından okunabilir veya yazılabilir.

Bir bilgisayarın adres uzayı, genellikle bellek yönetimi tarafından yönetilir ve işlemci veya diğer donanım birimleri tarafından bellek erişim işlemleri için kullanılır. Bellek adres uzayı, fiziksel bellek (RAM) ve sanal bellek (disk üzerindeki sanal bellek alanı) gibi farklı bellek türlerini içerebilir.

Bir bilgisayarın adres uzayı genellikle çok geniş bir aralıkta olabilir ve kullanılan mimariye ve bellek yönetimi yöntemlerine bağlı olarak değişir. Örneğin, 32-bit bir işlemci genellikle 2^32 (yaklaşık 4 milyar) bellek adresini desteklerken, 64-bit bir işlemci 2^64 (çok daha büyük) bellek adresini destekleyebilir.

terimler:

1. პროცესი. ოპერაციულ სისტემაში მნიშვნელოვან როლს თამაშობს პროცესის ცნება. პროცესი თავისი არსით წარმოადგენს პროგრამას შესრულების მომენტში.

2. მისამართების სივრცე შეიცავს შესრულებად პროგრამას, მის მონაცემებსა და სტეკს.

3. მრავალ ოპერაციულ სისტემაში ყველა პროცესზე ინფორმაცია (პროცესის მისამართების სირვცეში არსებული მონაცემების გამოკლებით) ინახება ე.წ. პროცესების ცხრილში და ოპერაციულ სისტემაში არსებული ყოველი პროცესისთვის ის წარმოადგენს სტრუქტურათა მასივს.

4. ოპერაციული სისტემის ამოცანას წარმოადგენს მომხმარებლისგან დამალოს მყარი დისკის და შეტანა/გამოტანის სხვა მოწყობილობების მუშაობის სპეციფიკა და

პროგრამისტს შესთავაზოს მოწყობილობებისგან დამოუკიდებელი, მოხერხებული და გასაგები აბსტრაქტული მოდელი - ფაილი, როგორც მყარ დისკზე განთავსებული მონაცემების გარკვეული ერთობლიობა.

5. ოპერაციული სისტემა ფაილების ნაკრების ერთ ჯგუფში გასაერთიანებლად იყენებს კატალოგის (ფოლდერი) აბსტრაქციას როგორც მეთოდს.

6. ოპერაციულ სისტემაში ფაილების და კატალოგების შესაქმნელად, წასაშლელად და მათზე დასაშვები სხვა ოპერაციების განსახორციელებლად გამოიყენება სისტემური გამოძახებები.

7. ოპერაციულ სისტემაში შეტანა/გამოტანის სხვადასხვა ფიზიკური მოწყობილობების სამართავად გამოიყენება პროგრამათა ქვესისტემა.

8. UNIX-ში ფაილს დაცვისათვის ენიჭება 9-თანრიგა კოდი. ეს

კოდი შედგება 3 ბიტიანი ველებისაგან. რომელთაგან პირველი სამეული განეკუთვნება

მომხმარებელს, მეორე სამეული მომხმარებლის ჯგუფს და მესამე სამეული კი - სხვა დანარჩენ

მომხმარებლებს. თითოეულ ველში შესაბამისად არსებობს კითხვის, რედაქტირების და

შესრულების უფლების აღმნიშვნელი ბიტი. ამ ბიტებს ეწოდებათ rwx-ბიტები (read, write,

execute).

********Çekirdek (kernel) ve kullanıcı modu, bir işletim sistemi içindeki iki farklı çalışma modunu ifade eder.

I. **Çekirdek Modu (Kernel Mode)**:

- Çekirdek modu, işletim sistemi çekirdeğinin (kernel) çalıştığı moddur.

- Bu modda işletim sistemi çekirdeği, işlemciye tam erişim yetkisine sahiptir ve sistemin tüm kaynaklarını yönetir.

- Önemli sistem çağrıları, donanım kaynaklarına erişim ve bellek yönetimi gibi işlemler çekirdek modunda gerçekleştirilir.

- Bu modda çalışan kodlar genellikle işletim sistemi çekirdeğine aittir ve doğrudan donanım ile iletişim kurarlar.

II. **Kullanıcı Modu (User Mode)**:

- Kullanıcı modu, normal uygulamaların çalıştığı moddur.

- Bu modda çalışan uygulamalar, işletim sistemi çekirdeğine göre sınırlı erişim yetkisine sahiptir.

- Kullanıcı modunda, genellikle kullanıcıların çalıştırdığı uygulamalar yer alır. Bu uygulamaların çoğu, kullanıcının etkileşimde bulunduğu programlar veya kullanıcının talebine göre çalışan işlemlerdir.

- Kullanıcı modunda çalışan kodlar, işletim sistemi çekirdeğinin sağladığı hizmetleri (örneğin, dosya okuma/yazma, ağ iletişimi) kullanarak işlemlerini gerçekleştirirler.

Çekirdek modu ve kullanıcı modu, işletim sistemi tarafından sağlanan bir güvenlik önlemidir. Çekirdek modunda çalışan kodlar, sistem kaynaklarına doğrudan erişebilirlerken,

kullanıcı modunda çalışan kodlar sadece belirlenmiş izinlerle sınırlıdır. Bu, işletim sistemi ve uygulamalar arasında bir izolasyon sağlar ve istenmeyen erişim veya hatalı davranışları önler.

9. როდესაც მომხმარებლის პროგრამა საჭიროებს გარკვეული პრივილეგირებული ბრძანების შესრულებას ის მიმართავს ოპერაციულ სისტემას შესაბამისი ბრძანებით, რომელსაც სისტემური გამოძახება ეწოდება, და პრივილეგირებულ

რეჟიმში ასრულებს საჭირო მოქმედებებს. სისტემური გამოძახება ეს არის ინტერფეისი ოპერაციულ სისტემასა და გამოყენებით პროგრამას შორის. მათ შეუძლიათ შექმნან, წაშალონ და გამოიყენონ სხვადასხვა ობიექტები.

გამოყენებითი პროგრამა სისტემური გამოძახების მეშვეობით მიმართავს ოპერაციულ სისტემას გარკვეული სერვისების მისაღებად.

********Bir "interrupt" (kesme), bir bilgisayar sistemine dışarıdan bir olayın, genellikle bir donanım cihazından gelen bir sinyalin veya bir yazılım komutunun, normal çalışmayı durdurarak işletim sistemi veya işlemci dikkatini olaya odaklamasıdır.

OS-codes

İşletim sistemi veya işlemci, kesmeyi işleyerek normal çalışmayı durdurur, kesmeyi işleyen bir hizmet veya işlem başlatır ve ardından normal çalışmaya geri döner.

Kesmeler, bilgisayar sistemlerinde çeşitli durumları işlemek için kullanılır.

10. წყვეტა (system calls).interrupt). ეს არის (პროცესორთან მიმართებაში) გარე მოწყობილობის მიერ გენერირებული მოვლენა. აპარატურა აპარატული წყვეტის მეშვეობით ახდენს ცენტრალური პროცესორის ინფორმირებას, რომ მოხდა გარკვეული მოვლენა და საჭიროა დაუყოვნებელი

რეაგირება ან, აცნობებს შეტანა/გამო-ტანის ასინქრონული ოპერაციის დასრულების შესახებ.

********Bir "exception" (istisna), bir programın normal akışını bozan ve genellikle istenmeyen bir durumu temsil eden bir olay veya durumdur. İstisnalar, programın normal işleyişini etkileyen hataları işlemek, istenmeyen durumları ele almak veya özel koşullar altında özel davranışlar sağlamak için kullanılır.

İstisnalar, genellikle şu durumlarda ortaya çıkar:

I. **Hata Durumları**: Programın çalışması sırasında beklenmeyen durumlar meydana gelebilir. Örneğin, bir dosyanın bulunamaması, bellek tahsisi sırasında hata olması veya bir işlemcinin tanımlanamaması gibi durumlar istisna olarak kabul edilebilir.

II. **İşletim Sistemi ve Donanım Hataları**: İşletim sistemi veya donanım düzeyindeki hatalar, bir programın normal akışını etkileyebilir. Örneğin, diskteki bir okuma hatası veya ağ bağlantısının kesilmesi gibi durumlar istisna olabilir.

III. **Programın Tanımladığı Özel Durumlar**: Bir program belirli koşullar altında özel davranışlar sergilemek isteyebilir. Örneğin, bir dosyanın sonuna ulaşılması veya bir ağ isteğinin zaman aşımına uğraması gibi durumlar program tarafından özel olarak ele alınabilir.

İstisnalar, programın normal işleyişini etkileyen bir olay olduğunda genellikle istisna yönetimi mekanizması kullanılarak işlenir. Programcılar, istisna durumları belirler ve bu durumları uygun şekilde ele almak için kodlarını yazabilirler. Bu, hata ayıklamayı ve kodun daha güvenilir olmasını sağlayabilir.

Örneğin, bir C++ programında, `try`, `catch` ve `throw` anahtar kelimeleriyle istisnalar yakalanabilir ve yönetilebilir. Bir istisna oluştuğunda, programın normal akışı `try` bloğundan `catch` bloğuna geçer ve istisna orada ele alınır. Böylece, programın hata durumlarına uygun şekilde yanıt vermesi sağlanır.

11. განსაკუთრებული შემთხვევა (system calls).exception) არის პროგრამის მიერ ბრძანების

შესრულების მცდელობის შედეგად წარმოშობილი მოვლენა, რომელიც გარკვეული მიზეზების

გამო შეუძლებელია შესრულდეს. ასეთი ბრძანების მაგალითი შეიძლება იყოს არასაკმარისი

პრივილეგიის ქონისას შეზღუდულ რესურსზე წვდომის მცდელობა ან მეხსიერების არარსე-ბულ

გვერდზე მიმართვა. განსაკუთრებული შემთხვევები შეიძლება დაიყოს ორ ნაწილად:

გამოსწორებადი და გამოუსწორებელი. გამოსწორებად განსაკუთრებულ შემთხვევას

OS-codes

მიეკუთვნება ისეთი განსაკუთრებული შემთხვევა, როგორიცაა დროის მიმდინარე მომენტისთვის

ოპერატიულ მეხსიერებაში საჭირო ინფორმაციის არარსებობა. გამოსწორებადი განსაკუთრებული

სიტუაციის გამომწვევი მიზეზის აღმოფხვრის შემდეგ პროგრამას შეუძლია გააგრძელოს შესრულება.

გამოუსწორებელი განსაკუთრებული შემთხვევა ძირითადად წარმოიშობა პროგრამული

შეცდომებისას (მაგალითად, ნულზე გაყოფა). ჩვეულებრივ, ასეთ შემთხვევებში ოპერაციული

სისტემა აჩერებს იმ პროგრამის შესრულებას, რომელმაც წარმოშვა გამოუსწორებელი

განსაკუთრებული შემთხვევა.

1.5 ოპერაციული სისტემის სტრუქტურა

*********Monolitik işletim sistemi, tüm işletim sistemi hizmetlerinin tek bir büyük program olarak çalıştığı bir mimariye sahiptir. Bu tip işletim sistemlerinde, çekirdek (kernel) adı verilen ana işletim sistemi bileşeni, tüm hizmetleri sağlar ve kullanıcı işlemleri doğrudan çekirdek içinde çalışır.

Monolitik işletim sistemleri genellikle aşağıdaki özelliklere sahiptir:

I. **Tek Parça Yapı**: Tüm işletim sistemi hizmetleri (dosya sistemi, bellek yönetimi, cihaz sürücüleri, ağ protokolleri vb.), birlikte tek bir büyük yazılım parçası olan çekirdek içinde bir araya getirilir.

II. **Hızlı İletişim**: Çünkü tüm hizmetler aynı adres uzayında çalışır, işletim sistemi bileşenleri arasındaki iletişim ve veri paylaşımı hızlıdır. Bu, işletim sistemi hizmetlerinin birbiriyle etkileşimini kolaylaştırır.

III. **Yüksek Performans**: Monolitik yapının doğası gereği, sistem çağrıları (system calls) gibi işletim sistemi hizmetlerinin çağrılması hızlıdır, çünkü işlem doğrudan çekirdeğe yönlendirilir ve birçok ara katman bulunmaz.

IV. **Daha Fazla Bellek Kullanımı**: Monolitik işletim sistemleri genellikle daha fazla bellek tüketir, çünkü tüm hizmetler bir arada bulunur ve her hizmetin ayrı bir bellek alanı olmaz. Bu, özellikle sınırlı belleğe sahip sistemlerde önemli olabilir.

V. **Bakım Zorluğu**: Monolitik işletim sistemlerinin bakımı genellikle daha zordur, çünkü tüm bileşenler bir arada olduğu için bir bileşen değişikliği diğerlerini etkileyebilir ve sistemin bütünlüğünü korumak daha zor olabilir.

Örnek olarak, Linux işletim sistemi, monolitik bir yapıya sahiptir. Çekirdek, tüm işletim sistemi hizmetlerini sağlar ve kullanıcı işlemleri, sistem çağrıları aracılığıyla çekirdeğe yönlendirilir. Bu yapı, Linux'un yüksek performans, esneklik ve güvenilirlik sağlamasına yardımcı olur. Ancak, bakımı ve geliştirilmesi daha karmaşık olabilir.

OS-codes

1. მონოლიტური სისტემა. მონოლიტური არქიტექტურით ორგანიზებული ოპერაციული სისტემა

ფართოდაა გავრცელებული. ამ შემთხვევაში მთლიანი სისტემა მუშაობს როგორც ერთი პროგრამა, ანუ

ოპერაციული სისტემის კოდი დაწერილია როგორც პროცედურათა ნაკრები, რომლებიც ერთ დიდ

შესრულებად პროგრამაში არის გაერთიანებული

2. მრავალდონიანი არქიტექტურა. მონოლიტური მიდგომის განზოგადებას წარმოადგენს

ოპერაციული სისტემის ორგანიზება დონეების იერარქიის სახით, რომელშიც თითოეული

დონეზე თავმოყრილია სისტემაში მსგავსი ფუნქციების განმახორციელებელი კომპონენტები.

ასეთნაირად ორგანიზებულ ოპერაციულ სისტემაში, სხვა დონეებზე ზემოქმედების გარეშე შესაძლებელია,

ცალკეული დონეების მოდიფიცირება. დონეებს შორის გადასვლა საჭიროებს შუალედური ელემენტების

გამოყენებას, რაც ამცირებს სისტემის წარმადობას. მრავალი თანამედროვე ოპერაციული სისტემა, მათ შორის

Windows XP და Linux-ი შეიძლება გარკვეული თვალსაზრისით განეკუთვნებოდნენ

მრავალდონიან სისტემებს.

3.

****

***** seminar1-2: *****

****

1.2. ფაილის სრული და მიმართებითი სახელები

ფაილი ეს არის მყარ დისკზე განთავსებული ერთ მონაცემებთან დაკავშირებული მეხსიერების მისამართების ერთობლიობა (ნაკრები).

OS-codes

UNIX-მსგავს ოპერაციულ სისტემაში ყოველი ფაილი გაერთიანებულია ხისებრ ლოგიკურ სტრუქტურაში. ფაილები შეიძლება გაერთიანდნენ დირექტორიებში ან კატალოგებში

ყველა დირექტორიას შორის არსებობს ერთადერთ დირექტორია, რომელიც არ წარმოადგენს სხვა დირექტორის ქვედირექტორიას, ასეთ დირექტორიას საბაზოან root დირექტორია ეწოდება.

UNIX ოპერაციულ სისტემაში ფაილისთვის სახელის მინიჭებისას არსებობს გარკვეული შეზღუდვები. POSIX სტანდარტში UNIX ოპერაციული სისტემისთვის სისტემური გამოძახების ინტერფეისი

შეიცავს მხოლოდ სამ ცხად შეზღუდვას:

I სახელის სიგრძე არ უნდა აღემატებოდეს სისტემაში გათვალისწინებულ სიგრძეს (Linuxსთვის - 255 სიმბოლო);

II არ შეიძლება NULL სიმბოლოს გამოყენება;

III არ შეიძლება ‘/’ სიმბოლოს გამოყენება

UNIX ოპერაციულ სისტემაში ფაილთან დაკავშირებით გვაქვს ფაილის სრული და მიმართებითი სახელის ცნება.

სრული: ლოგიკურ ხეზე საბაზო დირექტორიიდან დაწყებული ამ ფაილამდე (/ usr home jack bin a.out).

pwd ბრძანებას გამოაქვს მიმდინარე დირექტორიის სახელი მუშა ბრძანებათა ინტერპრეტატორისთვის.

`man` komutu, Linux ve diğer Unix benzeri işletim sistemlerinde komut satırından belirli bir komut veya konu hakkında kullanılabilir bilgiyi görüntülemek için kullanılır. "man" kısaltması "manual" kelimesinden gelir ve komutların veya sistem özelliklerinin belgelendirilmiş kılavuzlarını gösterir.

`man` komutunun kullanımı şu şekildedir:

```
man [OPTION] [COMMAND/KEYWORD]
```

Örneğin:

- Belirli bir komut için kılavuzu görüntülemek için: `man ls` (Bu, `ls` komutunun kılavuzunu görüntüler.)

- Belirli bir bölümdeki kılavuzu görüntülemek için: `man 5 passwd` (Bu, "passwd" komutuyla ilgili 5. bölüm kılavuzunu görüntüler.)

OS-codes

- Belirli bir anahtar kelime veya konu hakkında kılavuzu aramak için: `man -k printf` (Bu, "printf" anahtar kelimesiyle ilgili tüm kılavuz girişlerini listeler.)

`man` komutunun bazı yaygın kullanım seçenekleri şunlardır:

- `-k` veya `--apropos`: Anahtar kelimelerle kılavuz girişlerini arar.

- `-f` veya `--whatis`: Komutların kısa tanımlarını gösterir.

- `-a` veya `--all`: Tüm uygun bölümlerdeki kılavuzları gösterir.

- `-S` veya `--sections`: Belirli bölümlerde kılavuzları arar.

Özellikle belirli bir konu hakkında bilgi almak istediğinizde `man` komutu oldukça faydalıdır ve sistemde yüklü olan tüm kılavuzlar `man` komutu ile erişilebilir.

`cat`, Unix ve Unix benzeri işletim sistemlerinde kullanılan bir komuttur ve "concatenate" kelimesinin kısaltmasıdır. Genellikle dosyaları birleştirmek ve içeriklerini görüntülemek için kullanılır.

`cat` komutunun genel kullanımı şu şekildedir:

```
cat [OPTION] [FILE]...
```

Örneğin:

- Bir dosyanın içeriğini konsola yazdırmak için: `cat filename.txt`

- Birden fazla dosyanın içeriğini birleştirmek ve konsola yazdırmak için: `cat file1.txt file2.txt`

- Bir dosyanın içeriğini başka bir dosyaya kopyalamak için: `cat source.txt > destination.txt` veya `cat source.txt >> destination.txt` (varsa hedef dosyanın sonuna ekler)

`cat` komutunun bazı yaygın kullanım seçenekleri şunlardır:

- `-n` veya `--number`: Satırları numaralandırır.

- `-b` veya `--number-nonblank`: Boş olmayan satırları numaralandırır.

- `-E` veya `--show-ends`: Satır sonlarını `$` ile gösterir.

- `-T` veya `--show-tabs`: Sekme karakterlerini `^I` ile gösterir.

`cat` komutu oldukça esnek bir komuttur ve birçok farklı senaryoda kullanılabilir. Dosya içeriğini birleştirmek, görüntülemek veya başka bir dosyaya kopyalamak gibi işlemler için sıklıkla tercih edilir.

OS-codes

ახალი ფაილის შექმნა ტერმინალიდან ასევე შესაძლებელია nano ტექსტური რედაქტორის გამოყენებით, რომელიც გადაგვიყვანს მისი რედაქტირების რეჟიმში. მისი სინტაქსისია nano FILENAME

ფაილურ სისტემაში ახალი დირექტორიის შესაქმნელად გამოიყენება ბრძანება mkdir (make

directory), მისი სინტაქსისია

mkdir DIRNAME

ერთი დირექტორიიდან მეორეში ფაილის (ან ფაილების) გადასაკოპირებლად (copy-paste) გამოიყენება ბრძანება cp (copy). ის, ასევე, გამოიყენება ერთი დირექტორიის

(ან დიერექტორიების) სხვა დირექტორიაში რეკურსიული10 გადაკოპირებისთვის. მისი

სინტაქსისია

cp FILENAME DESTINNAME

cp FILENAME1 FILENAME2 ... FILENAMEN DESTINNAME

cp -r SDIRNAME DESTINNAME

cp -r DIR1 DIR2 ... DIRN DESTINNAME

სადაც SDIRNAME (source directory) საწყისი დირექტორიაა, რომლიდანაც უნდა მოხდეს

ფაილების გადაკოპირება, DESTINNAME (destination directory) დანიშნულების დირექტორია,

სადაც უნდა მოხდეს მონაცემების გადაკოპირება, -r ოფცია გამოიყენება რეკურსიული

გადაკოპირებისთვის.

ფაილის ერთი დირექტორიიდან მეორეში გადასაადგილებლად (cut-paste) გამოიყენება

ბრძანება mv (move). მისი სინტაქსისია

mv SOURCENAME DFILENAME

mv FILENAME1 FILENAME2 ... FILENAMEN DESTINNAME

დირექტორიიდან ფაილის (ან ფაილების) წასაშლელად გამოიყენება ბრძანება rm (remove).

მისი სინტაქსისია

rm FILENAME1 FILENAME2 ... FILENAMEN

OS-codes

მომხმარებლის იდენტიფიკატორის - UID, და მისი ჯგუფის იდენტიფიკატორის - GID,

მნიშვნელობის მისაღებად შესაბამისად გამოიყენება სისტემური გამოძახებები getuid() და getgid().

#include<sys/types.h>

#include<unistd.h>

uid_t getuid(void);

gid_t getgid(void);

1.6. ფაილზე დაშვების უფლებები

გამოთვლით სისტემას შეიძლება ჰყავდეს ბევრი მომხმარებელი. თითოეული მომხმარებელი ინახავს მისთვის მნიშვნელოვან სხვადასხვა ფაილებს (იურიდიული, ფინანსური და ა.შ.).

ოპერაციულ სისტემას უნდა შეეძლოს მომხმარებლისთვის მნიშვნელოვანი ფაილების დაცვა

არასანქცირებული დაშვებისგან. ამ მიზნით UNIX-მსგავს სისტემებში განასხვავებენ დაშვების სამ

უფლებას:

 კითხვის უფლება - r (read);

 რედაქტირების უფლება - w (write);

 შესრულების უფლება - x (execute).

`chown` ve `chgrp` komutları, Linux ve diğer Unix tabanlı işletim sistemlerinde dosya veya dizinlerin sahibini ve grubunu değiştirmek için kullanılır.

1. **chown**: "change owner" kelimelerinin kısaltmasıdır. Bu komut, dosyanın veya dizinin sahibini değiştirmek için kullanılır. Örneğin:

   ```
   chown kullanici_adı dosya_adı
   ```

   Yukarıdaki komut, `dosya_adı`'nın sahibini `kullanici_adı` olarak değiştirir.

2. **chgrp**: "change group" kelimelerinin kısaltmasıdır. Bu komut, dosyanın veya dizinin grubunu değiştirmek için kullanılır. Örneğin:

   ```
   chgrp grup_adı dosya_adı
   ```

OS-codes

```

Yukarıdaki komut, `dosya_adı`'nın grubunu `grup_adı` olarak değiştirir.

Bu komutlar genellikle sistem yöneticileri tarafından kullanılır ve dosya ve dizinlerin sahiplik ve grup bilgilerini yönetmek için çok önemlidir.

`chmod` komutu, dosya ve dizinlerin erişim izinlerini değiştirmek için kullanılır. Yukarıdaki komut şablonunda şu unsurlar bulunur:

- `[who]`: Hangi kullanıcı grubunun izinleri değiştirileceğini belirtir. İzinler dosya sahibi (owner), dosya sahibinin grubu (group) veya diğer kullanıcılar (others) için değiştirilebilir. `[who]` kısmı aşağıdaki karakterlerden biriyle belirtilir:
  - `u`: Dosya sahibi (owner)
  - `g`: Dosya sahibinin grubu (group)
  - `o`: Diğer kullanıcılar (others)
  - `a`: Tümü (all) - Dosya sahibi, dosya sahibinin grubu ve diğer kullanıcılar
- `{ + | – | = }`: İzinlerin nasıl değiştirileceğini belirtir.
  - `+`: İzinleri ekler
  - `-`: İzinleri kaldırır
  - `=`: İzinleri belirtilen değere ayarlar
- `[perm]`: Değiştirilecek izinlerin belirtildiği kısım. İzinler üç haneli bir sayı ile temsil edilir. Her hane, okuma, yazma ve yürütme izinlerini belirtir ve değerler 0 ile 7 arasında olabilir. İzinlerin anlamları şöyledir:
  - 0: İzin yok
  - 1: Yürütme izni (execute)
  - 2: Yazma izni (write)
  - 3: Yazma ve yürütme izni
  - 4: Okuma izni (read)
  - 5: Okuma ve yürütme izni
  - 6: Okuma ve yazma izni
  - 7: Okuma, yazma ve yürütme izni
- `FILENAME1 FILENAME2 ... FILENAMEN`: İzinlerin değiştirileceği dosya veya dizinlerin adlarını belirtir. Birden fazla dosya veya dizin adı verilebilir.
```

OS-codes

Örneğin:

- `chmod u+x myfile.txt`: `myfile.txt` dosyasının sahibine yürütme izni ekler.

- `chmod go-w mydir`: `mydir` dizininin dosya sahibi olmayan ve dosya sahibinin grubu dışındaki kullanıcılardan yazma iznini kaldırır.

- `chmod a=r myfile.txt`: `myfile.txt` dosyasının tüm kullanıcılar için okuma iznini (read permission) ayarlar.

seminar-lek 4

//განვიხილოთ მაგალითი. შევქმნათ პროცესი, რომელშიც მასივში არსებული მონაცემების

//ჩაწერა მოხდება პროგრამით შექმნილ ფაილში.

/*

IO cihazları ve işlem arasındaki iletişim bağlantısı dosya yoluyla sağlanır. Bir işlem, bir dosyayı okuyabilir, yazabilir veya üzerinde değişiklik yapabilir.

Bu, işlemlerin veri akışını yönetmek ve dosyalarda depolanan bilgilere erişmek için kullandığı yaygın bir iletişim yöntemidir.

Dosya işlemleri sırasında, işletim sistemi bir dosyayı temsil etmek için kullanılan bir veri yapısı olan "dosya tanımlayıcı tablosu"nu kullanır.

Dosya tanımlayıcı tablosu, bir işlem tarafından açılan tüm dosyaların bilgilerini içerir ve bu dosyalarla ilgili işlemleri gerçekleştirmek için kullanılır.

Her dosya için bir tanımlayıcı atanır ve bu tanımlayıcı, dosyanın diğer özelliklerine ve verilerine erişim sağlar.

Dosya tanımlayıcı tablosu, genellikle bir tablo veya veri yapısı şeklinde tutulur ve her dosya tanımlayıcısına bir dizi bilgi bağlıdır. Bu bilgiler arasında dosya konumu,

dosya boyutu, okuma/yazma konumları ve dosya erişim izinleri gibi bilgiler bulunabilir. Dosya tanımlayıcı tablosu, işletim sistemi tarafından yönetilir ve işlem tarafından dosyalarla etkileşimde bulunmak için kullanılır.

Dosya tanımlayıcı tablosunun kullanımı, işletim sistemi ve programlama dili arasında farklılık gösterebilir. Örneğin, Unix/Linux sistemlerinde dosya tanımlayıcıları genellikle bir tamsayı olarak temsil edilir (örneğin, 0, 1, 2 gibi),

Windows sistemlerinde ise bir işaretçi olarak temsil edilir. Ancak temel prensip her durumda aynıdır: dosyaların işlenmesi ve yönetilmesi için kullanılan bir veri yapısı sağlamak.

**********Ornek kod anlatimi1**********/

#include <fcntl.h> //fcntl.h: Dosya kontrol işlemleri için sembollerin tanımlandığı başlık dosyası. Örneğin, fcntl() işlevi, dosya açma modunu, dosya kilitlerini ve diğer dosya özelliklerini kontrol etmek için kullanılır.

#include <stdio.h> //stdio.h: Standart giriş/çıkış işlevlerinin tanımlandığı başlık dosyası. Bu dosya, konsoldan veya diğer standart giriş/çıkış cihazlarından veri okumak veya veri yazmak için printf, scanf, fopen, fclose gibi işlevleri sağlar.

#include <stdlib.h>//stdlib.h: Genel amaçlı işlevlerin tanımlandığı başlık dosyası. Bellek yönetimi (malloc, free), programın sonlandırılması (exit) ve diğer çeşitli işlemler için kullanılır.

#include <unistd.h>//unistd.h: UNIX sistem çağrılarının tanımlandığı başlık dosyası. Bu dosya, düşük seviyeli işlemler için kullanılır, örneğin dosya işlemleri, işlem kontrolü, bellek yönetimi ve diğer sistem düzeyi işlemleri.

#include <sys/types.h>//sys/types.h: Temel veri türlerinin tanımlandığı başlık dosyası. Özel türlerin tanımlarını içerir, örneğin size_t, pid_t, time_t gibi.

#include <sys/stat.h>//sys/stat.h: Dosya ve dosya sistemi durumuyla ilgili sabitlerin, yapıların ve işlevlerin tanımlandığı başlık dosyası. Dosya özelliklerini almak ve dosya durumunu değiştirmek için kullanılır.

int main(){

int fd; // ცვლადი ფაილური დესკრიპტორის მნიშვნელობისთვის. Unix/Linux sistemlerinde dosya tanımlayıcıları genellikle bir tamsayı olarak temsil edilir

size_t s1, s2; // ცვლადები მონაცემების ზუსტად გადაცემის შესამოწმებლად

char mass[] = "Operating System";

s2 = sizeof(mass);

// გავანულოთ შესაქმნელი ფაილისთვის ოპერაციული სისტემის მიერ გაჩუმებით

// დანიშვნადი დაშვების უფლებები

OS-codes

(void) umask(0);//umask işlevi, işlem tarafından oluşturulan dosyaların varsayılan izinlerini ayarlar.

//umask(0) çağrısı, mevcut umask değerini değiştirir ve sıfır olarak ayarlar. Bu, dosya oluşturulduğunda varsayılan izinlerin herhangi bir sınırlama olmaksızın (yani, tüm izinlerin açık olduğu) ayarlanmasını sağlar.

// სამუშაო დირექტორიაში შევმნქათ test.txt ფაილი (უფლებებით 0765)

fd = open("test.txt", O_WRONLY | O_CREAT, 0765);

/*open()

open() işlevi, Unix ve Unix benzeri işletim sistemlerinde dosya veya cihazlar üzerinde dosya açma işlemi gerçekleştirmek için kullanılır. Dosya açma işlemi sırasında dosyanın adı ve açma modu (dosya yoksa oluşturma, mevcutsa açma, vs.) belirtilir

open() işlevi başarılı olursa, dosyanın bir dosya tanımlayıcısı (file descriptor) döndürür. Başarısız olması durumunda -1 değeri döner ve errno değişkeni ilgili hatayı belirtir.

İşlevin prototipi genellikle şu şekildedir: int open(const char *path, int flags);

path: Açılacak dosyanın yolunu ifade eden bir karakter dizisi.*/

/*flags

flags parametresi, open() işlevi tarafından belirli dosya açma modlarını belirlemek için kullanılan bir parametredir. Bu bayraklar, dosyanın açılma şeklini, erişim izinlerini ve diğer seçenekleri belirler.

Bazı yaygın kullanılan flags bayrakları şunlardır:

O_RDONLY: Sadece okuma modunda dosyayı açar.

O_WRONLY: Sadece yazma modunda dosyayı açar.

O_RDWR: Okuma ve yazma modunda dosyayı açar.

O_CREAT: Dosya mevcut değilse oluşturur. Eğer dosya varsa bu bayrak göz ardı edilir.

O_EXCL: Dosya mevcut değilse oluşturur. Eğer dosya varsa hata döndürür.genelde creatla beraber kullanilir

O_TRUNC: Dosya açılırken boyutunu sıfırlar. Yani, dosyayı temizler ve içeriğini siler.

O_APPEND: Dosyaya yazma işlemi yaparken, imleci dosyanın sonuna konumlandırır.

O_NONBLOCK veya O_NDELAY: Blok olmayan modda (non-blocking mode) dosyayı açar. Okuma veya yazma işlemi sırasında dosya hemen hazır olmadığında işlem bloklanmaz, hemen geri döner.

O_SYNC veya O_DSYNC: Eşzamanlı (synchronous) dosya açma modunu belirler. Verinin disk üzerine yazılması tamamlanmadan işlem tamamlanmaz.

Bu bayraklar, bitwise veya (|) operatörü ile birleştirilerek aynı anda birden fazla bayrağın belirtilebileceği anlamına gelir.

OS-codes

*/

/*mode

mode, Unix/Linux sistemlerinde dosya oluşturma işlemi sırasında kullanılan bir parametredir. Bu parametre, oluşturulan dosyanın erişim izinlerini ve dosya tipini belirler.

mode, open() veya creat() gibi dosya oluşturma işlevlerine geçirilen bir parametredir.

Sayısal izinler, bir dosyanın veya dizinin okunabilirlik, yazılabilirlik ve çalıştırılabilirlik gibi erişim izinlerini temsil eder. Bir dosyanın sayısal izinleri, sekizlik (octal) sayı sistemi kullanılarak ifade edilir.

İzinler üç rakamdan oluşur: dosya sahibinin izinleri, dosya sahibinin grubunun izinleri ve diğer kullanıcıların izinleri.

Sayısal izinler, bir dosyanın veya dizinin okunabilirlik, yazılabilirlik ve çalıştırılabilirlik gibi erişim izinlerini temsil eder. Bir dosyanın sayısal izinleri, sekizlik (octal) sayı sistemi kullanılarak ifade edilir. İzinler üç rakamdan oluşur: dosya sahibinin izinleri, dosya sahibinin grubunun izinleri ve diğer kullanıcıların izinleri.

Her rakam, r, w ve x gibi belirli bir erişim türünü temsil eder. Bu izinler aşağıdaki şekilde tanımlanır:

- **r (read)**: Dosyayı okuma izni.
- **w (write)**: Dosyaya yazma izni.
- **x (execute)**: Dosyayı çalıştırma izni.

Her izin için bir rakam kullanılır ve bu rakamların toplamı toplam 3 rakama ulaşır. Bu rakamlar şunlardır:

- **0**: İzin yok.
- **1**: Yalnızca çalıştırma (execute) izni.
- **2**: Yalnızca yazma (write) izni.
- **3**: Yazma (write) ve çalıştırma (execute) izinleri.
- **4**: Yalnızca okuma (read) izni.
- **5**: Okuma (read) ve çalıştırma (execute) izinleri.
- **6**: Okuma (read) ve yazma (write) izinleri.
- **7**: Okuma (read), yazma (write) ve çalıştırma (execute) izinleri.

OS-codes

Bu izinler, octal sayı sistemi kullanılarak birleştirilir. Örneğin:

- Dosya sahibi için okuma, yazma ve çalıştırma izinlerine sahip olmak için 7 kullanılır (rwx).

- Dosya sahibinin grubu için yalnızca okuma ve yazma izinlerine sahip olmak için 6 kullanılır (rw-).

- Diğer kullanıcılar için yalnızca okuma iznine sahip olmak için 4 kullanılır (r--).

Bu nedenle, bir dosyanın izinlerini temsil eden sayısal bir ifade üç haneli bir rakam olacaktır. Örneğin, 764 sayısal izni, dosya sahibinin izinlerini `rwx`, grup izinlerini `rw-` ve diğer kullanıcıların izinlerini `r--` şeklinde temsil eder.

*/

// შევამოწმოთ fd ფაილური დესკრიპტორის მნიშვნელობა

if (fd < 0){ // თუ ფაილი არ შეიქმნა, მასში მონაცემებს ვერ ჩავწერთ

printf("Can\'t open file\n");

exit(EXIT_FAILURE);

}

// ინფორმაციის ჩაწერა კავშირის არხში

s1 = write(fd, mass, s2);

/*write() and read()

`write` ve `read` işlevleri dosya giriş/çıkış işlemlerinde kullanılır.

- `write`: Belirtilen dosyaya veri yazmayı sağlar. İşlevin imzası şu şekildedir:

  ssize_t write(int fd, const void *buf, size_t count);

  - `fd`: Yazma işleminin yapılacağı dosyanın dosya tanımlayıcısı.

  - `buf`: Yazılacak verinin bellek adresini gösteren işaretçi.

  - `count`: Yazılacak verinin boyutu (bayt cinsinden).

  - Geri dönüş değeri, yazılan verinin boyutudur. Hata durumunda `-1` döner.

- `read`: Belirtilen dosyadan veri okumayı sağlar. İşlevin imzası şu şekildedir:

  ```c
  ```

OS-codes

ssize_t read(int fd, void *buf, size_t count);

```

- `fd`: Okuma işleminin yapılacağı dosyanın dosya tanımlayıcısı.

- `buf`: Okunan verinin yazılacağı bellek alanının adresini gösteren işaretçi.

- `count`: Okunacak verinin maksimum boyutu (bayt cinsinden).

- Geri dönüş değeri, okunan verinin boyutudur. Hata durumunda `-1` döner.

Bu işlevler dosya giriş/çıkış işlemlerinde oldukça yaygın olarak kullanılır ve genellikle sistem çağrıları ile birlikte kullanılırlar.*/

// თუ მონაცემების ჩაწერისას სისტემური გამოძახება წარუმატებლად დასრულდა ან

// მონაცემები სრულად არ იქნა ჩაწერილი, მაშინ ვასრულებთ პროგრამას

if (s1 != s2) {

printf("Can\'t write all massing\n");

exit(EXIT_FAILURE);

}

/*`close()`

işlevi, belirtilen dosya tanımlayıcısını kapatır ve ilişkili dosya kaynaklarını serbest bırakır. Bu işlev, dosya giriş/çıkış işlemlerinden sonra dosya tanımlayıcısını serbest bırakmak için kullanılır.

İşlevin imzası aşağıdaki gibidir:

int close(int fd);

- `fd`: Kapatılacak dosyanın dosya tanımlayıcısı.

İşlev, başarı durumunda `0` değerini döndürürken, hata durumunda `-1` değerini döndürür.*/

close(fd); //დავხუროთ კავშირის არხი

return 0; }

**********Ornek kod anlatimi2**********/

#include<stdio.h>

#include<stdlib.h>
```

OS-codes

```
#include<unistd.h>

#include<sys/types.h>

int main(){

int fd[2]; // ორელემენტიანი მასივი ფაილური დესკრიპტორისთვის


size_t s1, s2; // ცვლადები მონაცემთა ზუსტად გადაცემის შესამოწმებლად

char mass[] = "Operating System!";

s2 = sizeof(mass);

char re_mass[s2];

// შევქმნათ კავშირის არხი pipe-ი

/*pipe?

Bir "pipe" (boru), Unix ve Unix benzeri işletim sistemlerinde iki süreç arasında iletişim kurmak için kullanılan bir yöntemdir.

Bir süreçten diğerine veri aktarmak için bir kanal sağlar. Pipe'lar genellikle bir süreçten diğerine veri akışı sağlamak için kullanılır.

Pipe'lar, çoğu Unix benzeri işletim sistemlerinde dosya işlemleri aracılığıyla oluşturulur ve yönetilir. İki süreç arasında tek yönlü veri akışını desteklerler: veri bir süreçten diğerine aktarılır, ancak ters yönde veri akışı mümkün değildir.

Pipe oluşturmak için pipe() sistem çağrısı kullanılır. Bu çağrı, bir dizi dosya tanımlayıcısı (file descriptor) döndürür: bir tanesi okuma, diğeri yazma için kullanılır. Bu dosya tanımlayıcıları, read() ve write() sistem çağrılarıyla veri alışverişinde bulunmak için kullanılır.*/

if (pipe(fd) < 0) {

  // pipe(fd) işlevi başarılı olduğunda, iki dosya tanımlayıcısı (file descriptor) atanır ve bunlar fd dizisine yerleştirilir. Bu dizinin ilk elemanı fd[0] okuma için kullanılırken, ikinci elemanı fd[1] yazma için kullanılır.

  //Dolayısıyla, bu dosya tanımlayıcıları, borunun okuma ve yazma uçları arasında veri iletişimi sağlamak için kullanılır.

printf("Can\'t create pipe\n");

exit(EXIT_FAILURE);

}

// pipe არხში შევიტანოთ mass მასივის მონაცემები

s1 = write(fd[1], mass, s2);

// შევამოწმოთ რამდენად სწორად მოხდა მონაცემების ჩაწერა
```

OS-codes

```
if (s1 != s2){
printf("Can\'t write all massing\n");
exit(EXIT_FAILURE);
}
// pipe არხიდან re_mass მასივში წავიკითხოთ მონაცემები
s1 = read(fd[0], re_mass, s2);
// შევამოწმოთ რამდენად სწორად მოხდა მონაცემების ჩაწერა
if (s1 != s2) {
printf("Can\'tread massing\n");
exit(EXIT_FAILURE);
}
printf("%s\n", re_mass); // დავბეჭდოთ re_mass მასივი
close(fd[0]); // დავხუროთ გახსნილი კავშირის არხები
close(fd[1]);
return 0;
}
```

**********Ornek kod anlatimi3**********/

//განვიხილოთ მაგალითი, რომელშიც ”მემკვიდრე“ პროცესებს შორის pipe არხის მეშვეობით

//მონაცემების გაცვლის მიზნით დემონსტრირებულია ერთმიმართულებიანი კავშირის ორგანიზება.

```
#include<sys/types.h>
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>
int main(){
```

OS-codes

```
int fd[2], pr;
size_t s1, s2;
char mass[] = "Operating System!";
s2 = sizeof(mass); char re_mass[s2];
 // შევქმნათ კავშირის არხი pipe-ი
if(pipe(fd) < 0){
printf("Can\'t create pipe\n");
exit(EXIT_FAILURE);
}
// შევქმნათ ახალი პროცესი
if((pr = fork()) < 0){
printf("Can\'t fork child\n");
exit(EXIT_FAILURE);
}
else {
if (pr != 0) { //მშობელ-პროცესი
close(fd[0]); // დავხუროთ მონაცემების კითხვის არხი
s1 = write(fd[1], mass, s2); // pipe არხში ჩავწეროთ მონაცემები
if(s1 != s2){
printf("Can\'t write all massing\n");
exit(EXIT_FAILURE);
}
close(fd[1]); // დავხუროთ მონაცემების ჩაწერის არხი
printf("Parent exit\n");
}
else { // შვილი-პროცესი
close(fd[1]); // დავხუროთ მონაცემების ჩაწერის არხი
s1 = read(fd[0], re_mass, s2); // კავშირის არხიდან წავიკითხოთ მონაცემები
if(s1 != s2){
```

OS-codes

```
printf("Can\'tread massing\n");
exit(EXIT_FAILURE);
}
printf("%s\n",re_mass); // დავბეჭდოთ წაკითხული სტრიქონი
close(fd[0]); // დავხუროთ მონაცემების კითხვის არხი
 }}
return 0;
}
```

```
**********Ornek kod anlatimi 4: FIFO **********/
#include<fcntl.h>
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<sys/stat.h>
#include<sys/wait.h>
#include<sys/types.h>
int main(){
/*FIFO
```

UNIX-მსგავს ოპერაციულ სისტემაში ნაკადებით ნებისმიერი პროცესების ურთიერთქმედებისთვის გამოიყენება კავშირის არხი, რომელსაც FIFO ეწოდება. FIFO ყველაფრით წააგავს pipe-ს გარდა ერთი გამონაკლისისა: ბირთვის მისამართების სივრცეში FIFO-ს განთავსებაზე და მის მდგომარეობაზე მონაცემების მიღება პროცესებს შეუძლიათ არა მემკვიდრე კავშირებით არამედ ფაილური სისტემიდან. ამ მიზნით FIFO-ს შექმნისას დისკზე იქმნება სპეციალური ტიპის ფაილი (.fifo), რომლის გამოყენებითაც პროცესები ცვლიან მონაცემებს ერთმანეთთან. პროცესების ურთიერთქმედების დასრულების შემდეგ FIFO არხი წყვეტს ფუნქციონირებას, ისევე როგორც pipe -ის შემთხვევაში, სპეციალური ტიპის ფაილი კი

OS-codes

სისტემაში რჩება. აღნიშნული ფაილის გამოყენება მომავალში პროცესებს შორის ურთიერთქმედებისთვის შესაძლებელია და პროცესების ურთიერთქმედების საჭიროებისას არ არსებობს აუცილებლობა შეიქმნას ახალი fifo გაფართოების ფაილი. ასეთ შემთხვევაში პროცესების ურთიერთქმედებამდე საჭიროა მათ იცოდნენ (სპეციალურად ამ მიზნისთვის შექმნილ) fifo გაფართოების ფაილამდე სრული ან მიმართებითი სახელი*/

```
int fd, pr, st, FIFO;

size_t s1, s2;

char mass[] = "Operating System!"; // გადასაცემი მასივი

s2 = sizeof(mass);

char re_mass[s2];

char name[]="file.fifo"; // კავშირის ფაილის სახელი

(void) umask(0); // გავანულოთ დაშვების საწყისი უფლებები

/*default izinler

Unix tabanlı işletim sistemlerinde, varsayılan dosya izinleri genellikle 666 ve varsayılan dizin izinleri ise 777'dir.

Bu, dosyaların okuma ve yazma yetkilerinin herkese açık olduğu (666) ve dizinlerin de tüm yetkilere (okuma, yazma, çalıştırma) açık olduğu (777) anlamına gelir.

 Bu varsayılanlar, güvenlik açısından istenmeyen sonuçlara yol açabilir, bu yüzden bazı uygulamalarda, özellikle güvenlik gereksinimleri olan sistemlerde,

 umask(0) çağrısı yapılabilir. Bu, dosya oluşturulduğunda belirli izinlerin devre dışı bırakılmasını sağlar ve kullanıcı, daha güvenli varsayılanlarla işlem yapabilir.

*/

FIFO = mknod(name, S_IFIFO | 0754, 0); //შევქმნათ კავშის არხი FIFO

/*mknod?

mknod() fonksiyonunu kullanarak FIFO dosyası oluşturulur: int mknod(char *path, int mode, int dev);

path: Oluşturulacak dosya düğümünün yolu.

mode: Dosya düğümünün izinlerini belirten bir tamsayı değeri.

dev: Dosya düğümünün türünü ve diğer özelliklerini belirten bir aygıt numarası.

FIFO-ს წარმატებით შექმნისას ფუნქცია აბრუნებს მნიშვნელობას 0, ხოლო წარუმატებელ შემთხვევაში კი უარყოფით მნიშვნელობას.
```

OS-codes

*/

/*S_IFIFO?

S_IFIFO, bir dosya tipi belirleyicisidir ve FIFO (First In, First Out) dosyalarını tanımlar. Bu sembolik sabit, dosya oluşturma işlevlerinde dosya tipini belirtmek için kullanılır.

FIFO dosyaları, çeşitli işlemler arasında veri iletişimi için kullanılabilir.

Örneğin, mknod() fonksiyonunu kullanarak FIFO dosyası oluştururken mode parametresine S_IFIFO sembolik sabitini verebilirsiniz:

mknod("myfifo", S_IFIFO | 0666, 0);

0666 bir modedir yukarida var izahi 1, 2,3 ,4 ,5, 6, 7

*/

if(FIFO < 0){

printf("შეუძლებელია FIFO-ს შექმნა\n");

exit(EXIT_FAILURE);

}

if((pr = (int) fork()) == -1){ // პროცესის შექმნა

printf("შეუძლებელია პროცესის წარმოქმნა\n");

exit(EXIT_FAILURE);

} else {

if (pr != 0){

// გავხსნათ FIFO არხი მონაცემების ჩასაწერად

fd = open(name, O_WRONLY); //dosya acilir, basarili olduysa dosya tanimlayiciyi dondurur degilse -1

if(fd < 0){

printf("შეუძლებელია FIFO-ს გახსნა\n");

exit(EXIT_FAILURE);

}

s1 = write(fd, mass, s2); // FIFO არხში ჩავწეროთ მონაცემები

if(s1 != s2){

printf("სრულად არ ჩაიწერა მონაცემები\n");

exit(EXIT_FAILURE);

}

close(fd); // დავხურო კავშირის არხი

wait(&st); // მშობელი ელოდება შვილის დასრულებას

/*wait()?

wait() işlevi, bir ebeveyn işlemin, bir veya daha fazla çocuk işleminin tamamlanmasını beklemesini sağlar.

Bu işlev, ebeveyn işlemin bir çocuk işleminin sonlanmasını beklerken askıya alınmasını ve diğer işlemlerle ilgilenmesini sağlar.

Bir çocuk işlem sonlandığında, wait() işlevi, çocuğun çıkış durumunu alır ve ebeveyn işlem için uygun bir şekilde işler.

Bu durum, çocuğun normal bir şekilde sonlandığı, bir sinyal tarafından sonlandığı veya bir hata durumunda sonlandığı gibi farklı olabilir.

wait() işlevi, ebeveyn işlemdeki çocuk işlemlerin bitiş sırasını kontrol etmek ve işlemlerin başarılı bir şekilde sonlanıp sonlanmadığını belirlemek için yaygın olarak kullanılır.

wait() fonksiyonunun kullanım prototipi şu şekildedir:

#include <sys/wait.h>

pid_t wait(int *status);

Bu prototip, wait() fonksiyonunu çağıran işlemin durumunu bekler. Eğer işlem başarıyla bekleniyorsa, geri dönüş değeri çocuğun PID'sidir. Başarısız olursa -1 döner ve errno değişkeni uygun bir hata koduyla ayarlanır.

status parametresi, çocuğun bitiş durumunu saklamak için bir işaretçidir. Eğer bu parametre NULL ise, bitiş durumu bilgisine ihtiyaç duyulmaz. Aksi takdirde, işlem tamamlandığında bitiş durumu bilgisi bu işaretçiye yazılır.

İşlevin dönüş değeri, sona eren işlemin PID'sidir veya bir hata durumunda -1 olabilir.

*/

} else {

// გავხსნათ FIFO არხი მონაცემების წასაკითხად

fd = open(name, O_RDONLY);

if (fd < 0){

printf("შეუძლებელი FIFO-ს გახსნა\n");

exit(EXIT_FAILURE);

}

// FIFO არხიდან წავიკითხოთ მონაცემები

OS-codes

```
s1 = read(fd, re_mass, s2);
if (s1 != s2){
printf("ინფორმაცია სრულადვერ იქნა წაკითხული \n");
exit(EXIT_FAILURE);
}
printf("\n\t%s\n",re_mass); // გადაცემული მონაცემების ბეჭდვა
close(fd); // დავხურო კავშირის არხი
}
}
exit(EXIT_SUCCESS);
/*შევნიშნოთ, რომ ამ პროგრამის ყოველი შემდეგი შესრულება მიგვიყვანს შეცდომამდე, რაც გამოწვეული იქნება პროგრამის შესრულების ყოველ ჯერზე სამუშაო დირექტორიაში fifo გაფართოების სპეციალური ფაილის არსებობით. პრობლემის გადაწყვეტა მოცემულ შემთხვევაში წარმოადგენს მისი შესრულების წინ ამ ფაილის წაშლა დირექტორიიდან, ან პირველი შესრულების შემდეგ განხორციელდებს პროგრამის ტექსტის რედაქტირება: იქიდან წაიშალოს ის ნაწილი, რომელიც დაკავშირებულია FIFO არხის წარმოქმნასთან ან მშობელ პროცესს მოვთხოვოთ, რომ სანამ დასრულდებოდეს წაშალოს FIFO არხი.
*/
}
```

****

***** lek1-5: *****

****

5.2

არაპრიორიტეტული გაძევებით

დაგეგმვის პოლიტიკის გამოყენებისას პროცესი, რომელიც იკავებს პროცესორს, ნებაყოფლობით

აბრუნებს მას დასრულების ან დროითი კვანტის ამოწურვის შემდეგ. პრიორიტეტული გაძევებით

დაგეგმვის პოლიტიკის გამოყენებისას კი პირიქით, სისტემაში მაღალპრიორიტეტული პროცესის

გამოჩენის შემთხვევაში მიმდინარე პროცესს (იძულებით) ჩამოერთმევა პროცესორი და ის

გადაეცემა მაღალპრიორიტეტულ პროცესს.

5.3

სტატიკური პრიორიტეტი (static priorities) არ შეიძლება შეიცვალოს. მისი დანიშვნის

მექანიზმის რეალიზება სისტემაში ადვილია და იწვევს სისტემის შედარებით ნაკლებ დაყოვნებას.

თუმცა ისინი არ რეაგირებენ გარშემო სიტუაციების ცვლილებაზე, რომლებმაც შესაძლებელია

განახორციელონ პრიორიტეტების კორექტირება და გამოიწვიონ დაყოვნების შემცირება.

დინამიური პრიორიტეტები (dynamic priorities) რეაგირებენ სიტუაციის ცვალებადობაზე.

დინამიური პრიორიტეტების სქემის რეალიზება, სტატიკურ სქემებთან მიმართებაში, რთულია და იძლევა მეტ დაყოვნებას.

5.4

მაგალითად, შეიძლება წარმოიშვას

სიტუაცია, რომლის დროსაც დაბალპრიორიტეტულ პროცესს დაკავებული აქვს რესურსი,

რომელსაც საჭიროებს მაღალპრიორიტეტული პროცესი. თუ ეს რესურსი არ არის განაწილებადი,

მაშინ დამგეგმავმა უნდა შესთავაზოს დაბალპრიორიტეტულ პროცესს მის დასასრულებლად

საჭირო რესურსები, რათა მან გამოათავისუფლოს მაღალპრიორიტეტული პროცესისთვის საჭირო

რესურსი. ამ მიდგომას ეწოდება პრიორიტეტების ინვერსია (priority inversion), ვინაიდან პროცესების

მიმართებითი პრიორიტეტები ცვლიან ადგილებს, რომლის მეშვეობითაც მაღალპრიორიტეტული

პროცესი ახერხებს მისთვის საჭირო რესურსის მიღებას.

 სამართლიანობა (fairness). დაგეგმვის პოლიტიკა სამართლიანია, თუ ყველა პროცესს ის

ექცევა ერთნაირად და არ ხდება პროცესის შესრულების უსასრულოდ გადადება დამგეგმავის

მიზეზით;

OS-codes

 წინასწარმეტყველებადობა (predictability). სისტემის თანაბარი დატვირთულობის შემთხვევაში პროცესის შესრულება არ უნდა საჭიროებდეს განსხვავებულ დროს სხვადასხვა დროს შესრულებისას;

 მასშტაბირებადობა (scalability). სისტემის დატვირთულობის შემთხვევაში დაგეგმვამ არ უნდა დაკარგოს ქმედუნარიანობა.

5.6

****

***** prak-6: *****

****

SystemVIPC (IPC - inter process communications). SystemVIPC ჯგუფში შედის: შეტყობინებათა მიმდევრობები,

განაწილებადი მეხსიერება და სემაფორები.

რაიმე სახის ობიექტების შესაძლო სახელების სიმრავლეს ეწოდება შესაბამისი ტიპის ობიექტების სახელების სივრცე.

SystemVIPC-სთვის სახელების ასეთ

სივრცეს წარმოადგენს მონაცემთა გარკვეული მთელმნიშვნელობიანი რიცხვების (key_t გასაღებების) სიმრავლე. ამასთან, პროგრამისტს არ შეუძლია პირდაპირ მიანიჭოს გასაღების მნიშვნელობა, ეს მნიშვნელობა მოიცემა გასაშუალოებით: ფაილურ სისტემაში არსებული გარკვეული ფაილის სახელის და პატარა მთელი რიცხვის კომბინაციით.

ორი კომპონენტისგან გასაღების მნიშვნელობის მიღება ხორციელდება ფუნქციით ftok()

ftok() ფუნქციის პროტოტიპი

```
#include <sys/types.h>
#include <sys/ipc.h>
key_t ftok(char *pathname, char proj_id);
```

ფუნქციის აღწერა

OS-codes

ftok ფუნქცია გამოიყენება არსებული ფაილის სახელის პატარა მთელ რიცხვში გარდაქმნისთვის,

მაგალითად, კავშირის საშუალების ეგზემპლარის რიგითი ნომრის SystemVIPC გასაღებში.

pathname პარამეტრი უნდა იყოს მიმთითებელი არსებულ ფაილზე, რომელზეც წვდომა გააჩნია

ftok ფუნქციის გამომძახებელ პროცესს.

proj_id - ეს არის პატარა მთელი რიცხვი, რომელიც ახასიათებს კავშირის საშუალების ასლს.

გასაღების გენერაციის შეუძლებლობის შემთხვევაში ფუნქცია აბრუნებს უარყოფით მნიშვნელობას წინააღმდეგ შემთხვევაში აბრუნებს გენერირებული გასაღების მნიშვნელობას.

key_t მონაცემთა ტიპი წარმოადგენს 32-თანრიგა მთელ რიცხვს.

System V IPC'de bir IPC kaynağına erişmek için kullanılan bir tür tanımlayıcıdır. Bu tanımlayıcılar, IPC kaynaklarını işaret etmek için kullanılır

ve System V IPC mekanizmaları arasında paylaşılır. Her bir System V IPC mekanizması, kendi türüne özgü bir tanımlayıcı kullanır.

Shared Memory, bilgisayar biliminde ve işletim sistemlerinde kullanılan bir IPC (Inter-Process Communication - Süreçler Arası İletişim) mekanizmasıdır.

Paylaşılan bellek, birden fazla süreç arasında veri paylaşımını mümkün kılan bir iletişim mekanizmasıdır.

İşletim sistemi, paylaşılan bellek bölgesini oluşturur ve bu bölgeye birden fazla sürecin erişimini sağlar. Süreçler, bu paylaşılan bellek bölgesine yazabilirler

veya ondan okuyabilirler. Bu, süreçler arasında veri aktarımını hızlandırır ve veri paylaşımını kolaylaştırır.

გარკვეული გასაღებით განაწილებადი მეხსიერების მიდამოს შექმნისთვის ან უკვე არსებულ განაწილებად მიდამოზე დაშვებისთვის გამოიყენება სისტემური გამოძახება shmget().

shmget() სისტემური გამოძახების პროტოტიპი

#include <sys/ipc.h>

#include <sys/shm.h>

int shmget(key_t key, int size, int shmflg);

სისტემური გამოძახების აღწერა

shmget სისტემური გამოძახება გამოიყენება განაწილებადი მეხსიერების სეგმენტზე დაშვებისთვის და მისი წარმატებით დასრულების შემთხვევაში ის აბრუნებს ამ სეგმენტისთვის

SystemVIPC დესკრიპტორს1

.

key პარამეტრი წარმოადგენს SystemVIPC გასაღებს სეგმენტისთვის, ანუ ფაქტიურად მის სახელს

OS-codes

SystemVIPC სახელების სივრციდან. ამ პარამეტრის მნიშვნელობის როლში შეიძლება გამოყენებული იყოს ftok() ფუნქციის მეშვეობით მიღებული გასაღების მნიშვნელობა ან სპეციალური მნიშვნელობა IPC_PRIVATE. IPC_PRIVATE მნიშვნელობის გამოყენებას ყოველთვის მივყავართ განაწილებადი მეხსიერების ახალი სეგმენტის შექმნის მცდელობამდე გასაღებით, რომელიც არ ემთხვევა არსებული სეგმენტების გასაღების მნიშვნელობებს და არ შეიძლება მიღებული იყოს ftok() -ის მეშვეობით.

size პარამეტრი განსაზღვრავს არსებული ან შესაქმნელი სეგმენტის მოცულობას ბაიტებში. თუ სეგმენტი მითითებული გასაღებით უკვე არსებობს, მაგრამ მისი მოცულობა არ ემთხვევა size პარამეტრით მითითებულ მნიშვნელობას წარმოიშობა შეცდომა.

shmflg პარამეტრი გამოიყენება მხოლოდ განაწილებადი მეხსიერების ახალი სეგმენტის შექმნისას

და განსაზღვრავს მომხმარებლების სეგმენტზე დაშვების უფლებებს, ასევე ახალი სეგმენტის შექმნის აუცილებლობას და სისტემური გამოძახების ყოფაქცევას შექმნის მცდელობისას. ის წარმოადგენს შემდეგი მნიშვნელობების გარკვეულ კომბინაციას (ბიტური ოპერაციით ან (‘|’)):

 IPC_CREAT – თუ სეგმენტი მითითებული გასაღებისთვის არ არსებობს, მაშინ ის უნდა შეიქმნას;

 IPC_EXCL – გამოიყენება IPC_CREAT flag-თან ერთად. მათი ერთობლივი გამოყენების და მითითებული გასაღებით სეგმენტის არსებობისას, არ ხორციელდება სეგმენტზე დაშვება და წარმოიშობა შეცდომის შემცველი სიტუაცია, ამასთან <errno.h> ფაილში აღწერილი errno ცვლადი ღებულობს მნიშვნელობას EEXIST;

 0400 – სეგმენტის შემქმნელი მომხმარებლისთვის ნებადართულია კითხვა;

 0200 – სეგმენტის შემქმნელი მომხმარებლისთვის ნებადართულია ჩაწერა;

 0040 – სეგმენტის შემქმნელი მომხმარებლის ჯგუფისთვის ნებადართულია კითხვა;

 0020 – სეგმენტის შემქმნელი მომხმარებლის ჯგუფისთვის ნებადართულია ჩაწერა;

 0004 – სხვა მომხმარებლისთვის ნებადართულია კითხვა;

 0002 – სხვა მომხმარებლისთვის ნებადართულია ჩაწერა.

დასაბრუნებელი მნიშნელობა

სისტემური გამოძახება წარმატებით დასრულებისას განაწილებადი მეხსიერების სეგმენტისთვის

OS-codes

აბრუნებს SystemVIPC დესკრიპტორის მნიშვნელობას და -1-ს შეცდომის წარმოქმნის

შემთხვევაში.

***descriptor, bir kaynağa erişmek için kullanılan bir tanımlayıcıdır, ancak key, bu kaynağı tanımlamak için kullanılan bir anahtardır.

İki kavram arasında dolaylı bir ilişki vardır çünkü bir descriptor genellikle bir key'e dayanır.

Süreç adres uzayı, her süreç için benzersiz bir şekilde oluşturulur ve yönetilir. Bu, her bir sürecin kendi bellek alanını ve

verilerini izole etmesini ve diğer süreçlerle etkileşimini kontrol altında tutmasını sağlar.

shmat() სისტემური გამოძახების პროტოტიპი

#include<sys/types.h>

#include <sys/shm.h>

char *shmat(int shmid, char *shmaddr, int shmflg);

სისტემური გამოძახების აღწერა

shmat სისტემური გამოძახება გამოიყენება პროცესის მისამრთების სივრცეში განაწილებადი

მეხსიერების სეგმენტის განსათავსებლად.

shmid პარამეტრი წარმოადგენს განაწილებადი მეხსიერების სეგმენტისთვის SystemVIPC

დესკრიპტორს, ანუ shmget() სისტემური გამოძახების მიერ დაბრუნებულ მნიშვნელობას.

shmaddr პარამეტრის როლში ვიყენებთ მნიშვნელობას NULL, რითაც ოპერაციულ სისტემას ეძლევა

შესაძლებლობა განათავსოს განაწილებადი მეხსიერება პროცესის მისამართების სივრცეში.

shmflg პარამეტრისთვის გამოვიყენებთ ორ მნიშვნელობას: 0 - კითხვის და ჩაწერის ოპერაციების

განსახორციელებლად ან SHM_RDONLY - თუ გვინდა მისგან მხოლოდ კითხვა. ამასთან

პროცესს უნდა გააჩნდეს სეგმენტზე დაშვების შესაბამისი უფლება.

დასაბრუნებელი მნიშნელობა

სისტემური გამოძახება წარმატებით დასრულების შემთხვევაში აბრუნებს განაწილებადი

მეხსიერების მისამართს პროცესის მისამართების სივრცეში და -1-ს შეცდომის წარმოქმნის

შემთხვევაში.

OS-codes

shmdt() სისტემური გამოძახების პროტოტიპი

#include<sys/types.h>

#include <sys/shm.h>

int shmdt(char *shmaddr);

სისტემური გამოძახების აღწერა

shmdt სისტემური გამოძახება გამოიყენება მიმდინარე პროცესის მისამართების სივრციდან

განაწილებადი მეხსიერების სეგმენტის ამოსაშლელად.

shmaddr პარამეტრი წარმოადგენს განაწილებადი მეხსიერების სეგმენტის მისამართს ანუ,

მნიშვნელობას რომელიც დააბრუნა სისტემურმა გამოძახებამ shmat().

დასაბრუნებელი მნიშნელობა

სისტემური გამოძახება აბრუნებს 0-ს ნორმალურად დასრულებისას და -1-ს შეცდომის წარმოქმნის

შემთხვევაში.