1906003052015

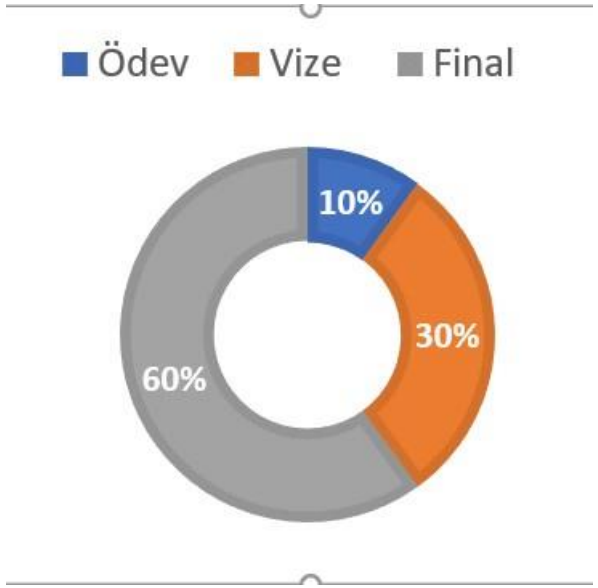
İşletim Sistemleri

Dr. Öğr. Üy. Önder EYECİOĞLU
Bilgisayar Mühendisliği

Giriş

Ders Günü ve Saati:
Çarşamba: 13:00-16:00

- Uygulama Unix (Linux) İşletim sistemi
- Devam zorunluluğu %70
- Uygulamalar C programlama dili üzerinde gerçekleştirilecektir. Öğrencilerden programlama bilgisi beklenmektedir.



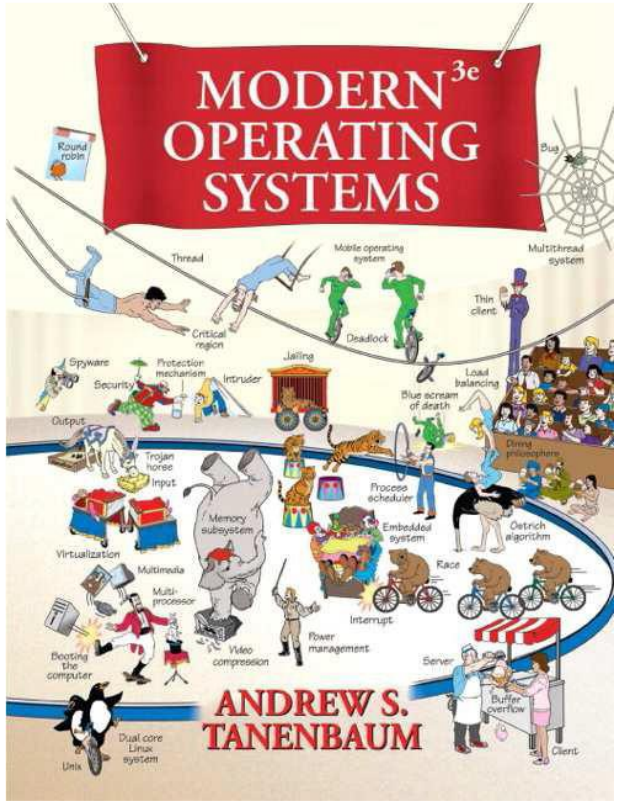
HAFTA	KONULAR
Hafta 1	: İşletim sistemlerine giriş, İşletim sistemi stratejileri
Hafta 2	: Sistem çağrıları
Hafta 3	: Görev, görev yönetimi
Hafta 4	: İplikler
Hafta 5	: İş sıralama algoritmaları
Hafta 6	: Görevler arası iletişim ve senkronizasyon
Hafta 7	: Semaforlar, Monitörler ve uygulamaları
Hafta 8	: Vize
Hafta 9	: Kritik Bölge Problemleri
Hafta 10	: Kilitlenme Problemleri
Hafta 11	: Bellek Yönetimi
Hafta 12	: Sayfalama, Segmentasyon
Hafta 13	: Sanal Bellek
Hafta 14	: Dosya sistemi, erişim ve koruma mekanizmaları, Disk planlaması ve Yönetimi
Hafta 15	: Final

Giriş

Kaynaklar:

– Modern Operating Systems, 3rd Edition by Andrew S. Tanenbaum, Prentice Hall, 2008.

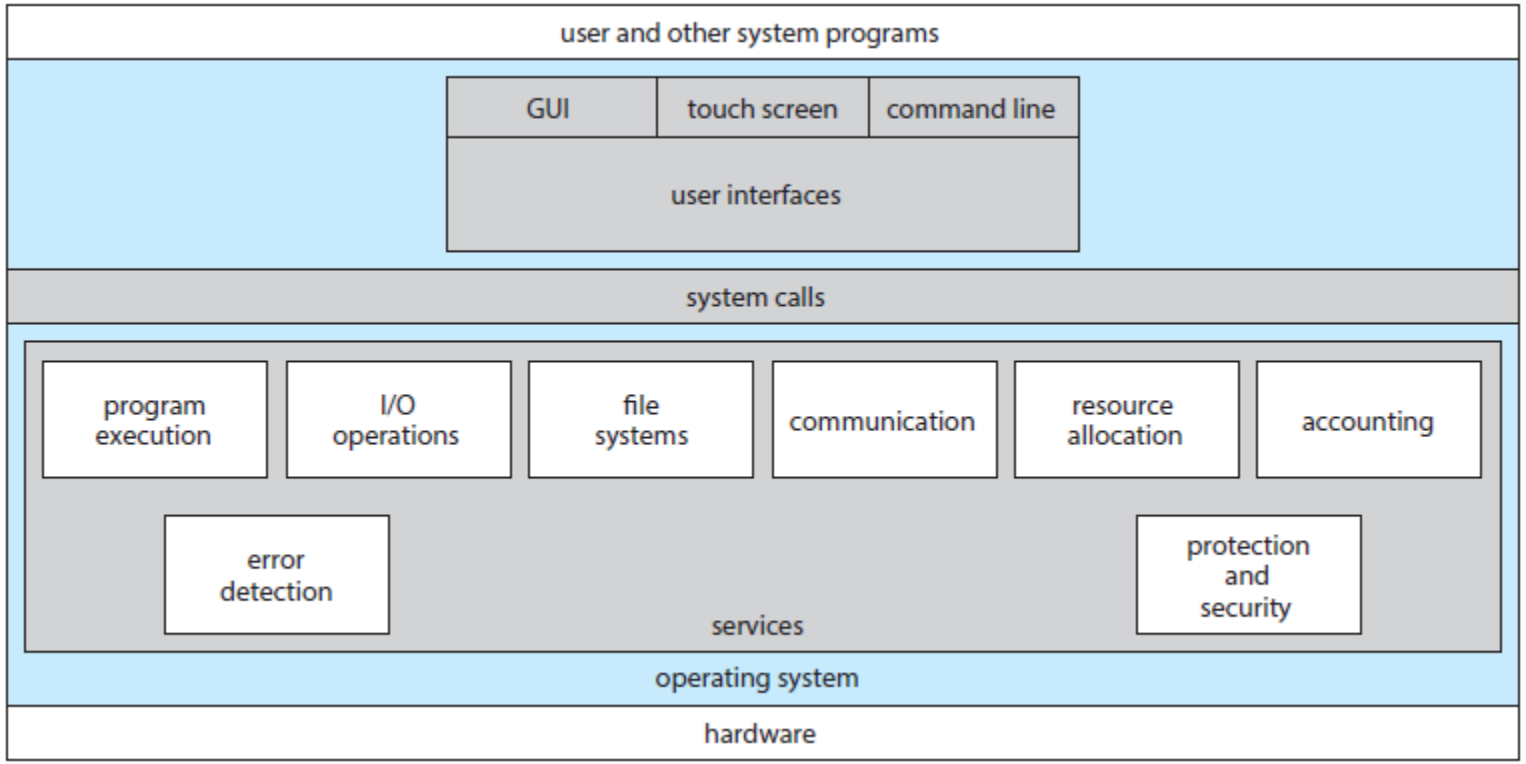
– Bilgisayar İşletim Sistemleri (BIS), Ali Saatçi, 2. Baskı, Bıçaklar Kitabevi.



DERS - 2

İŞLETİM SİSTEMİ HİZMETLERİ

Çoğu işletim sistemi, süreçler, adres alanları ve bunları anlamak için merkezi olan dosyalar gibi belirli temel kavramları ve soyutlamaları sağlar. Ayrıca Bir işletim sistemi, programların yürütülmesi için bir ortam sağlar. Belirli hizmetleri programlara ve bu programların kullanıcılarına sunar.



İŞLETİM SİSTEMİ HİZMETLERİ

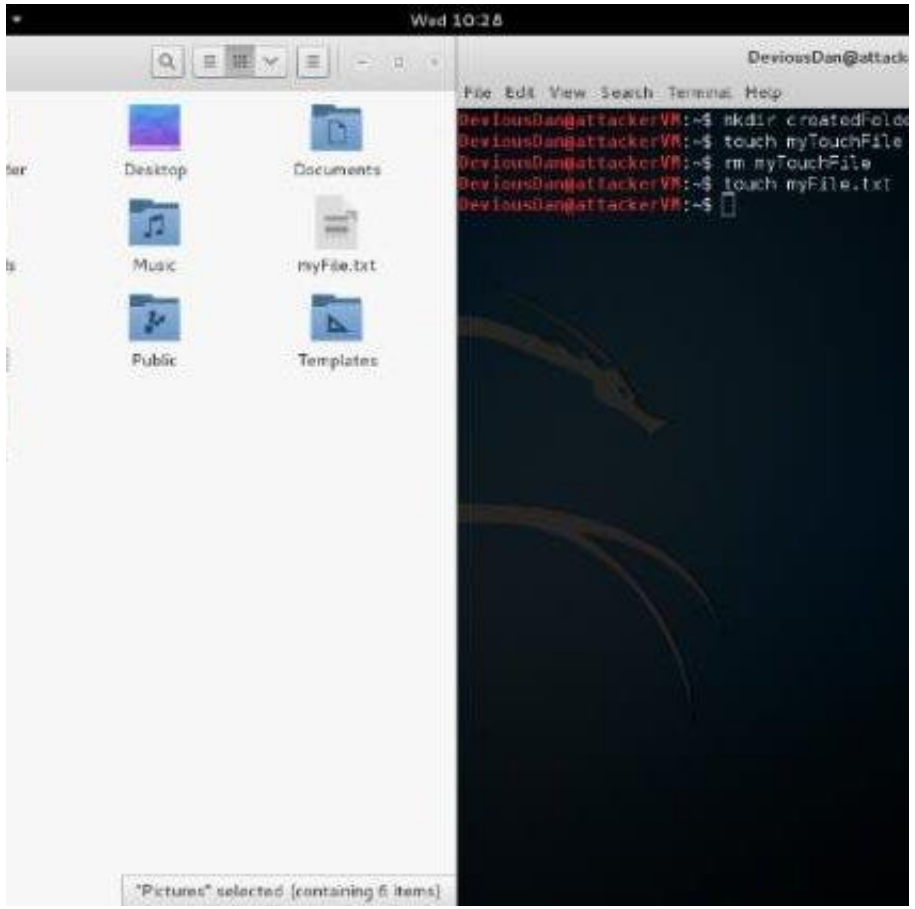
Kullanıcı arayüzü.

Hemen hemen tüm işletim sistemlerinde bir kullanıcı arayüzü (UI) bulunur. Bu arayüz çeşitli şekillerde olabilir.

- En yaygın olarak, bir grafik kullanıcı arabirimi (GUI) kullanılır. Burada arayüz, I/O'yu yönlendirmek, menülerden seçim yapmak ve seçimler yapmak için bir işaretleme aygıtı ve metin girmek için bir klavye işlevi gören fareli bir pencere sistemidir.

- Başka bir seçenek, metin komutlarını ve bunları girme yöntemini kullanan bir komut satırı arabirimidir (CLI), (örneğin, komutları belirli bir biçimde belirli seçeneklerle yazmak için bir klavye).

- Bazı sistemler bu varyasyonlardan ikisini veya üçünü sağlar.



İŞLETİM SİSTEMİ HİZMETLERİ

Programın yürütülmesi.

Sistem, bir programı belleğe yükleyebilmeli ve o programı çalıştırabilmelidir. Program, normal veya anormal bir şekilde (hata göstererek) yürütülmesini sonlandırabilmelidir.

G/Ç işlemleri.

Çalıştırma programı, bir dosya veya bir G/Ç aygıtı içerebilen G/Ç gerektirebilir.

Belirli cihazlar için özel işlevler istenebilir (bir ağ arayüzünden okuma veya bir dosya sistemine yazma gibi).

Verimlilik ve koruma için, kullanıcılar genellikle I/O cihazlarını doğrudan kontrol edemezler. Bu nedenle, işletim sistemi G/Ç yapmak için bir araç sağlamalıdır.

İŞLETİM SİSTEMİ HİZMETLERİ

Dosya sistemi manipülasyonu.

Programların dosya ve dizinleri okuması ve yazması gerekir.
Ayrıca, adlarına göre oluşturmaları ve silmeleri, belirli bir dosyayı aramaları ve dosya bilgilerini listelemeleri gerekir.
Son olarak, bazı işletim sistemleri, dosya sahipliğine bağlı olarak dosyalara veya dizinlere erişime izin vermek veya erişimi reddetmek için izin yönetimi içerir. Çoğu işletim sistemi, bazen kişisel seçime izin vermek ve bazen belirli özellikler veya performans özellikleri sağlamak için çeşitli dosya sistemleri sağlar.

İŞLETİM SİSTEMİ HİZMETLERİ

İletişim.

Bir sürecin başka bir süreçle bilgi alışverişi yapması gereken birçok durum vardır.

Bu tür bir iletişim, aynı bilgisayarda yürütülen işlemler arasında veya bir ağ ile birbirine bağlı farklı bilgisayar sistemlerinde yürütülen işlemler arasında gerçekleşebilir.

İletişim, iki veya daha fazla işlemin belleğin paylaşılan bir bölümüne okuyup yazdığı paylaşılan bellek veya önceden tanımlanmış formatlardaki bilgi paketlerinin işletim sistemi tarafından işlemler arasında taşındığı mesaj geçişi yoluyla gerçekleştirilebilir.

İŞLETİM SİSTEMİ HİZMETLERİ

Hata tespiti.

İşletim sisteminin hataları sürekli olarak algılaması ve düzeltmesi gerekir.

CPU ve bellek donanımında (bellek hatası veya elektrik kesintisi gibi), G/Ç aygıtlarında (diskte eşlik hatası, ağda bağlantı hatası veya yazıcıda kağıt eksikliği gibi) hatalar oluşabilir ve kullanıcı programında (aritmetik taşma veya geçersiz bir bellek konumuna erişme girişimi gibi).

Her bir hata türü için işletim sistemi, doğru ve tutarlı bilgi işlemi sağlamak için uygun eylemi gerçekleştirmelidir. Bazen, sistemi durdurmaktan başka seçeneği yoktur.

Diğer zamanlarda, hataya neden olan bir işlemi sonlandırabilir veya işlemin algılaması ve muhtemelen düzeltmesi için bir işleme bir hata kodu döndürebilir.

İŞLETİM SİSTEMİ HİZMETLERİ

Diğer bir dizi işletim sistemi işlevi, kullanıcıya yardımcı olmak için değil, sistemin kendisinin verimli çalışmasını sağlamak için mevcuttur. Birden çok işlemi olan sistemler, bilgisayar kaynaklarını farklı işlemler arasında paylaştırarak verimlilik kazanabilir.

Kaynak tahsisi.

- Aynı anda çalışan birden fazla işlem olduğunda, bunların her birine kaynaklar tahsis edilmelidir. İşletim sistemi birçok farklı kaynak türünü yönetir.
- Bazıları (CPU döngüleri, ana bellek ve dosya depolama gibi) özel tahsis koduna sahip olabilirken, diğerleri (I/O cihazları gibi) çok daha genel istek ve yayın koduna sahip olabilir. Örneğin, CPU'nun en iyi nasıl kullanılacağını belirlerken, işletim sistemlerinin CPU'nun hızını, yürütülmesi gereken işlemi, CPU'daki işlem çekirdeklerinin sayısını ve diğer faktörleri hesaba katan CPU zamanlama rutinleri vardır.
- Yazıcıları, USB depolama sürücülerini ve diğer çevresel aygıtları tahsis etmek için rutinler de olabilir.

İŞLETİM SİSTEMİ HİZMETLERİ

Kayıt tutma.

- Hangi programların ne kadar ve ne tür bilgisayar kaynaklarını kullandığını takip etmek istiyoruz.
- Bu kayıt tutma, muhasebe veya sadece kullanım istatistiklerini toplamak için kullanılabilir.
- Kullanım istatistikleri, bilgi işlem hizmetlerini iyileştirmek için sistemi yeniden yapılandırmak isteyen sistem yöneticileri için değerli bir araç olabilir.

Koruma ve güvenlik.

- Çok kullanıcılı veya ağ bağlantılı bir bilgisayar sisteminde saklanan bilgilerin sahipleri, bu bilgilerin kullanımını kontrol etmek isteyebilir.
- Birkaç ayrı işlem aynı anda yürütüldüğünde, bir işlemin diğerlerine veya işletim sisteminin kendisine müdahale etmesi mümkün olmamalıdır.
- Koruma, sistem kaynaklarına tüm erişimin kontrol edilmesini sağlamayı içerir. Sistemin dışarıdan gelenlere karşı güvenliği de önemlidir.
- Bu güvenlik, her kullanıcının kendi kimliğini doğrulamasını istemekle başlar.

İŞLETİM SİSTEMİ HİZMETLERİ

Koruma ve güvenlik.

- Bilgisayarlar, kullanıcıların genellikle korumak ve gizli tutmak istediği büyük miktarda bilgi içerir. Bu bilgiler e-posta, iş planları, vergi beyannameleri ve çok daha fazlasını içerebilir. Örneğin, dosyalara yalnızca yetkili kullanıcılar tarafından erişilebilmesi için sistem güvenliğini yönetmek işletim sistemine bağlıdır.
- UNIX'teki dosyalar, her birine 9 bitlik ikili koruma kodu atanarak korunur. Koruma kodu, biri sahip için, biri sahip grubunun diğer üyeleri için (kullanıcılar sistem yöneticisi tarafından gruplara ayrılır) ve biri diğer herkes için olmak üzere üç adet 3 bitlik alandan oluşur. Her alanın okuma erişimi için bir biti, yazma erişimi için bir biti ve yürütme erişimi için bir biti vardır. Bu 3 bit, rwx bitleri olarak bilinir. Örneğin, koruma kodu rwxr-x--x, sahibinin dosyayı okuyabileceği, yazabileceği veya yürütebileceği, diğer grup üyelerinin dosyayı okuyabileceği veya yürütebileceği (ancak yazamayacağı) ve diğer herkesin yürütülebileceği (ancak okunamayacağı) anlamına gelir. veya dosyayı yazın). Bir dizin için x, arama iznini belirtir. Kısa çizgi, ilgili iznin olmadığı anlamına gelir.
- Dosya korumasına ek olarak, başka birçok güvenlik sorunu vardır. Sistemi hem insan hem de insan olmayan (örneğin virüsler) istenmeyen davetsiz misafirlerden korumak bunlardan biridir.

İŞLETİM SİSTEMİ KAVRAMLARI

SÜREÇLER (PROCESS)

Tüm işletim sistemlerinde anahtar kavram, süreçtir. Bir süreç temelde yürütülmekte olan bir programdır. Her işlemle ilişkili, işlemin okuyabileceği ve yazabileceği 0'dan bazı maksimumlara kadar bellek konumlarının bir listesi olan adres alanıdır. Adres alanı yürütülebilir programı, programın verilerini ve yığınını içerir. Ayrıca her işlemle ilişkili olan, genellikle kayıtlar (program sayacı ve yığın işaretçisi dahil), açık dosyaların bir listesi, bekleyen alarmlar, ilgili işlemlerin listeleri ve programı çalıştırmak için gereken tüm diğer bilgileri içeren bir dizi kaynaktır. İşlem, temelde bir programı çalıştırmak için gereken tüm bilgileri içeren bir kapsayıcıdır.

İŞLETİM SİSTEMİ KAVRAMLARI

SÜREÇLER (PROCESS)

Anahtar süreç yönetim sistemi çağrıları, süreçlerin yaratılması ve sonlandırılmasıyla ilgilenenlerdir.

Bir süreç bir veya daha fazla başka süreç (alt süreçler olarak adlandırılır) oluşturabiliyorsa ve bu süreçler de alt süreçler oluşturabiliyorsa, süreç ağacı yapısı oluşur.

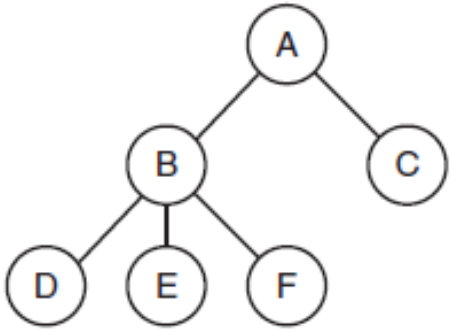


Figure 1-13. A process tree. Process *A* created two child processes, *B* and *C*. Process *B* created three child processes, *D*, *E*, and *F*.

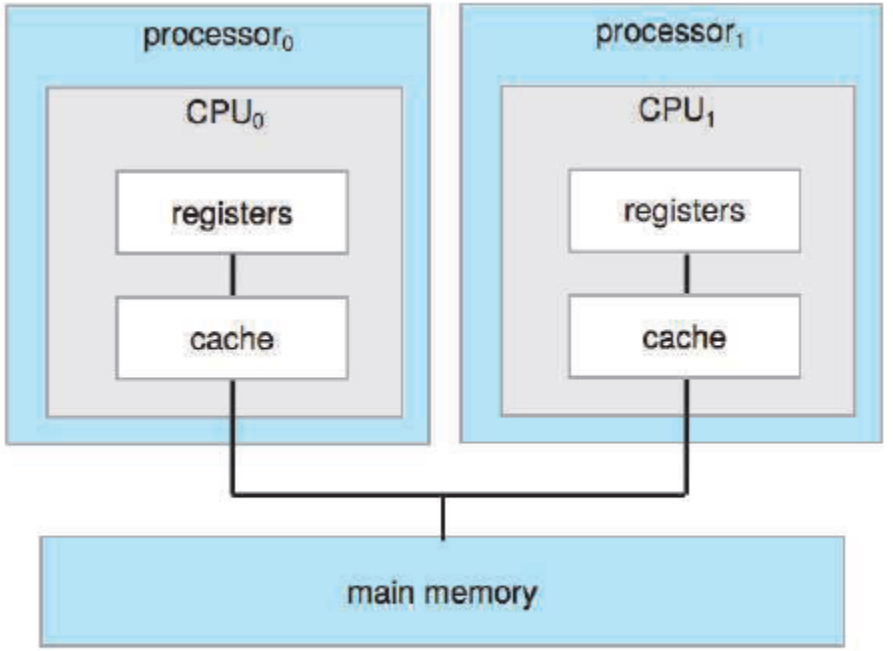


Figure 1.8 Symmetric multiprocessing architecture.

SÜREÇLER (PROCESS)

Bazı işleri yapmak için işbirliği yapan ilgili süreçlerin genellikle birbirleriyle iletişim kurması ve faaliyetlerini senkronize etmesi gerekir. Bu iletişime **süreçler arası iletişim** denir.

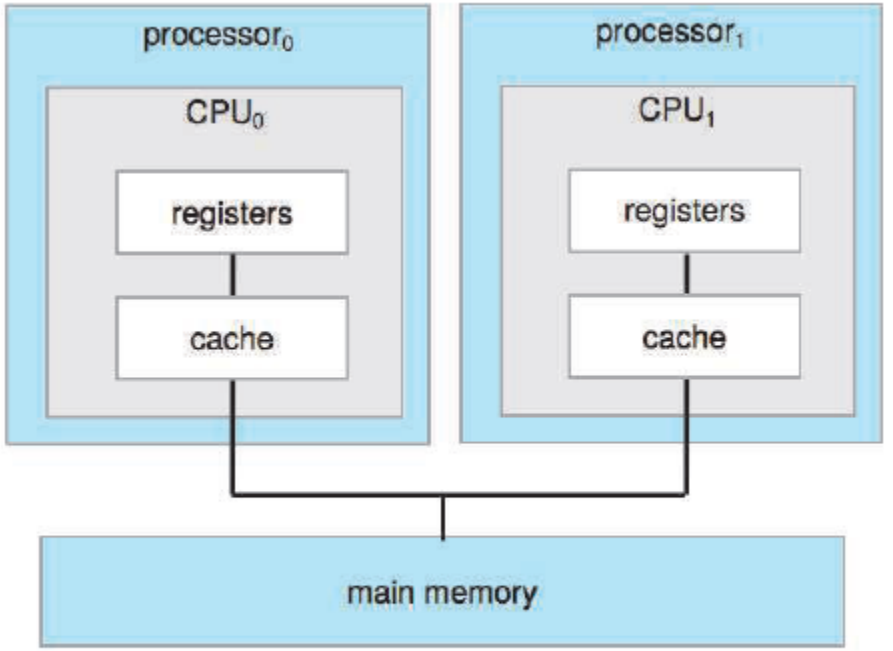


Figure 1.8 Symmetric multiprocessing architecture.

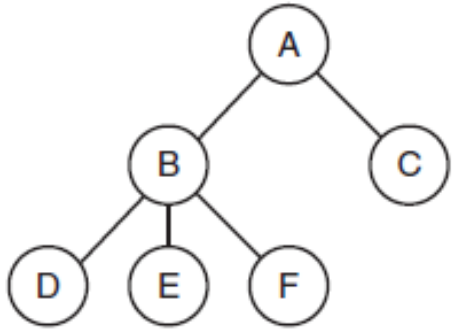


Figure 1-13. A process tree. Process *A* created two child processes, *B* and *C*. Process *B* created three child processes, *D*, *E*, and *F*.

İŞLETİM SİSTEMİ KAVRAMLARI

SÜREÇLER (PROCESS)

Bir sistemi kullanma yetkisi olan her kişiye, sistem yöneticisi tarafından bir UID (Kullanıcı Kimliği) atanır. Başlatılan her işlem, onu başlatan kişinin UID'sine sahiptir. Bir alt süreç, ebeveyniyle aynı UID'ye sahiptir. Kullanıcılar, her biri bir GID'ye (Grup Kimliği) sahip olan grupların üyeleri olabilir.

```
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           2  0.0  0.0      0     0 ?        S    16:55   0:00 [kthreadd]
root           3  0.0  0.0      0     0 ?        I<   16:55   0:00  \_ [rcu_gp]
root           4  0.0  0.0      0     0 ?        I<   16:55   0:00  \_ [rcu_par_gp]
root           6  0.0  0.0      0     0 ?        I<   16:55   0:00  \_ [kworker/0:0H-kblockd]
root           8  0.0  0.0      0     0 ?        I<   16:55   0:00  \_ [mm_percpu_wq]
```

İŞLETİM SİSTEMİ KAVRAMLARI

ADRES UZAYLARI

- Her bilgisayarın, çalışan programları tutmak için kullandığı bir ana belleği vardır. Çok basit bir işletim sisteminde, aynı anda yalnızca bir program bellekte bulunur.
- İkinci bir programı çalıştırmak için ilkinin kaldırılması ve ikincisinin belleğe yerleştirilmesi gerekir.
- Daha karmaşık işletim sistemleri, birden çok programın aynı anda bellekte olmasına izin verir.
- Birbirlerine (ve işletim sistemine) müdahale etmelerini önlemek için bir tür koruma mekanizmasına ihtiyaç vardır. Bu mekanizmanın donanımda olması gerekirken, işletim sistemi tarafından kontrol edilmektedir.
- Yukarıdaki bakış açısı, bilgisayarın ana belleğini yönetmek ve korumakla ilgilidir. Farklı, ancak aynı derecede önemli, bellekle ilgili bir konu, süreçlerin adres alanını yönetmektir. Normalde, her işlemin kullanabileceği, tipik olarak 0'dan maksimuma kadar çalışan bir dizi adres vardır.

İŞLETİM SİSTEMİ KAVRAMLARI

ADRES UZAYLARI

- En basit durumda, bir işlemin sahip olduğu maksimum adres alanı miktarı ana bellekten daha azdır. Bu şekilde, bir işlem adres alanını doldurabilir ve ana bellekte hepsini tutmak için yeterli alan olacaktır.
- Bununla birlikte, birçok bilgisayarda adresler 32 veya 64 bittir ve sırasıyla 2^{32} veya 2^{64} baytlık bir adres alanı verir. Bir işlemin bilgisayarın ana belleğinden daha fazla adres alanı varsa ve işlem hepsini kullanmak isterse ne olur? İlk bilgisayarlarda böyle bir süreç şanssızdı. Günümüzde, daha önce de belirtildiği gibi, sanal bellek adı verilen, işletim sisteminin adres alanının bir kısmını ana bellekte ve bir kısmını diskte tuttuğu ve gerektiğinde parçaları bunlar arasında ileri geri gönderen bir teknik var.
- Özünde, işletim sistemi, bir işlemin başvurabileceği adresler kümesi olarak bir adres alanının soyutlamasını yaratır. Adres alanı, makinenin fiziksel belleğinden ayrılmıştır ve fiziksel bellekten daha büyük veya daha küçük olabilir.
- Adres alanlarının ve fiziksel belleğin yönetimi, bir işletim sisteminin yaptığının önemli bir bölümünü oluşturur.

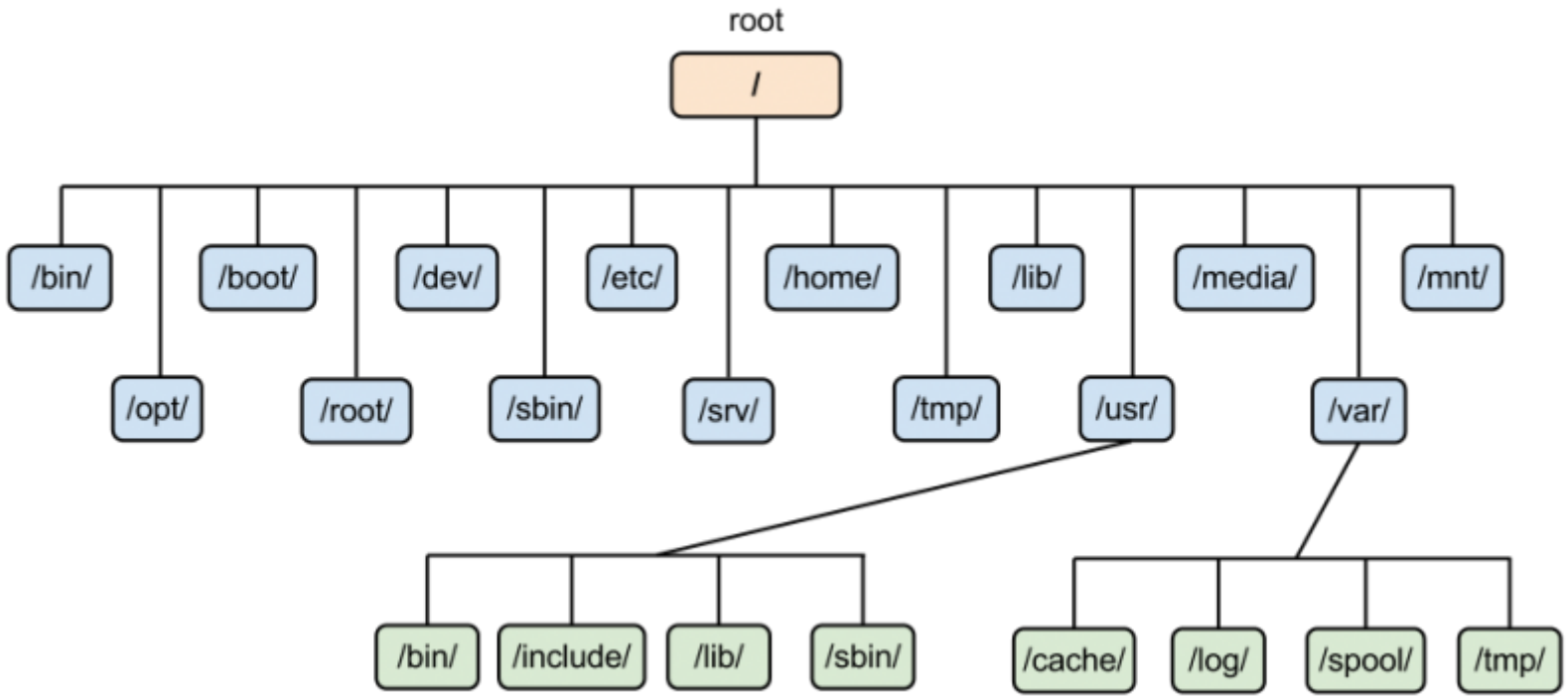
İŞLETİM SİSTEMİ KAVRAMLARI

DOSYA

- Hemen hemen tüm işletim sistemleri tarafından desteklenen bir diğer anahtar kavram ise dosya sistemidir.
- Daha önce belirtildiği gibi, işletim sisteminin önemli bir işlevi, disklerin ve diğer G/Ç aygıtlarının özelliklerini gizlemek ve programcıya aygıttan bağımsız dosyaların güzel, temiz, soyut bir modelini sunmaktır.
- Dosya oluşturmak, dosyaları kaldırmak, dosyaları okumak ve dosya yazmak için sistem çağrılarına açıkça ihtiyaç vardır. Bir dosyanın okunabilmesi için önce diskte bulunması ve açılması gerekir, okunduktan sonra kapatılması gerekir, bu nedenle bu işlemleri yapmak için çağrılar sağlanır.
- Dosyaları saklamak için bir yer sağlamak için çoğu PC işletim sistemi, dosyaları bir arada gruplamanın bir yolu olarak bir dizin kavramına sahiptir.
- Mevcut bir dosyayı bir dizine koymak ve bir dosyayı bir dizinden kaldırmak için çağrılar da sağlanır. Dizin girdileri, dosyalar veya başka dizinler olabilir.

İŞLETİM SİSTEMİ KAVRAMLARI

DOSYA



İŞLETİM SİSTEMİ KAVRAMLARI

DOSYA

- İşlem ve dosya hiyerarşilerinin her ikisi de ağaçlar olarak düzenlenir, ancak benzerlik buraya kadardır.
- Süreç hiyerarşileri genellikle çok derin değildir (üçten fazla düzey olağandışıdır), oysa dosya hiyerarşileri genellikle dört, beş, hatta daha fazla düzey derindir.
- Süreç hiyerarşileri tipik olarak kısa ömürlüdür, genellikle en fazla dakikalarca sürer, oysa dizin hiyerarşisi yıllarca var olabilir.

- Sahiplik ve koruma, işlemler ve dosyalar için de farklılık gösterir. Tipik olarak, yalnızca bir üst süreç bir alt süreci kontrol edebilir ve hatta bunlara erişebilir, ancak dosyaların ve dizinlerin yalnızca sahibinden daha geniş bir grup tarafından okunmasına izin veren mekanizmalar neredeyse her zaman mevcuttur.
- Dizin hiyerarşisindeki her dosya, dizin hiyerarşisinin en üstünden, kök dizin olan yol adını vererek belirtilebilir.

İŞLETİM SİSTEMİ KAVRAMLARI

DOSYA

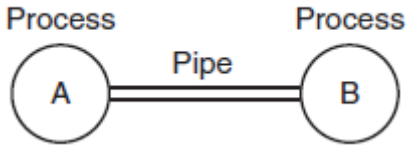
- Her an, her işlemin, eğik çizgi ile başlamayan yol adlarının arandığı geçerli bir çalışma dizini vardır. İşlemler, yeni çalışma dizinini belirten bir sistem çağrısı yayınlayarak çalışma dizinlerini değiştirebilir.
- Bir dosyanın okunabilmesi veya yazılabilmesi için önce açılması gerekir ve bu sırada izinler kontrol edilir. Erişime izin verilirse, sistem sonraki işlemlerde kullanmak üzere dosya tanımlayıcı adı verilen küçük bir tamsayı döndürür. Erişim engellenirse, bir hata kodu döndürülür.
- UNIX'teki bir diğer önemli kavram, bağlı (mounted) dosya sistemidir.

- UNIX'teki bir diğer önemli kavram da özel dosyadır. G/Ç cihazlarının dosyalara benzemesi için özel dosyalar sağlanmıştır.

İŞLETİM SİSTEMİ KAVRAMLARI

DOSYA

- Bu genel bakışta tartışacağımız son özellik, hem süreçler hem de dosyalar ile ilgilidir: borular. Bir boru, Şekil de gösterildiği gibi, iki işlemi birbirine bağlamak için kullanılabilen bir tür sahte dosyadır. A ve B süreçleri bir boru kullanarak konuşmak isterse, önceden ayarlamaları gerekir. A işlemi, B işlemine veri göndermek istediğinde, boruya bir çıktı dosyası gibi yazar. Aslında, bir borunun uygulanması bir dosyanınkine çok benzer.



- Boru, yalnızca bir komuttan başka bir komut tarafından okunana kadar verileri tutmak için var olan geçici bir dosya görevi görür. Bir Unix borusu, tek yönlü bir veri akışı sağlar. İlk komut dizisinin çıktısı veya sonucu, komut dizisinin girdisi olarak kullanılır. ikinci komut dizisi. Bir boru oluşturmak için, iki komut arasına komut satırına dikey bir çubuk (|) konulur.
- **who | wc -l**

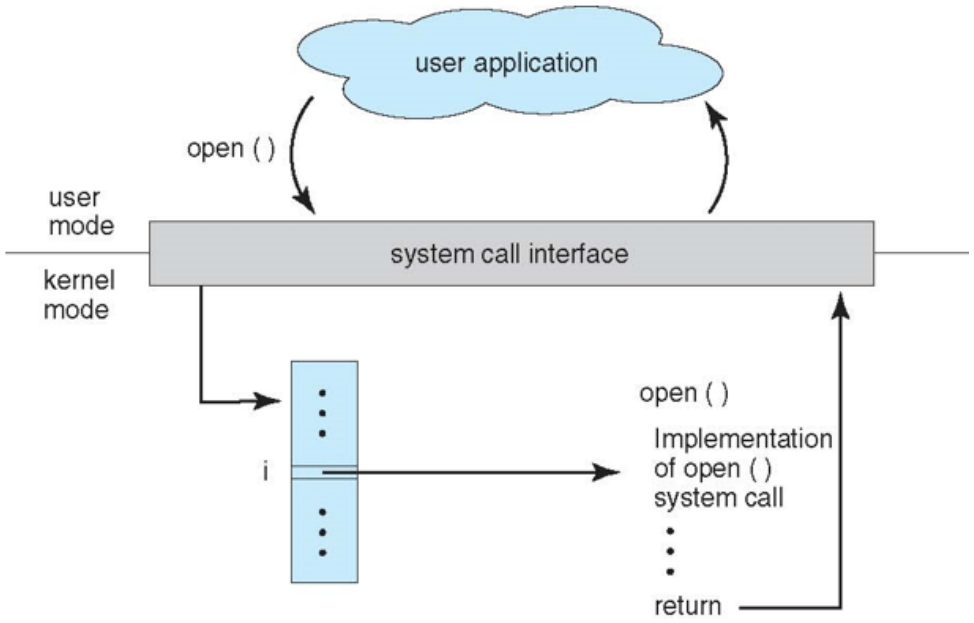
Sistem Çağrıları(System Calls)

- İşletim sistemlerinin iki ana işlevi olduğunu gördük: kullanıcı programlarına soyutlama sağlamak ve bilgisayar kaynaklarını yönetmek.
- Çoğunlukla, kullanıcı programları ve işletim sistemi arasındaki etkileşim, öncekilerle ilgilidir; örneğin, dosya oluşturma, yazma, okuma ve silme.
- Kaynak yönetimi bölümü, kullanıcılar için büyük ölçüde şeffaftır ve otomatik olarak yapılır. Bu nedenle, kullanıcı programları ve işletim sistemi arasındaki arayüz, öncelikle soyutlamalarla ilgilenmekle ilgilidir.
- İşletim sistemlerinin ne yaptığını gerçekten anlamak için bu arayüzü yakından incelememiz gerekiyor. Arayüzde bulunan sistem çağrıları bir işletim sisteminden diğerine farklılık gösterir (her ne kadar temeldeki kavramlar benzer olma eğiliminde olsa da).

Sistem Çağrıları(System Calls)

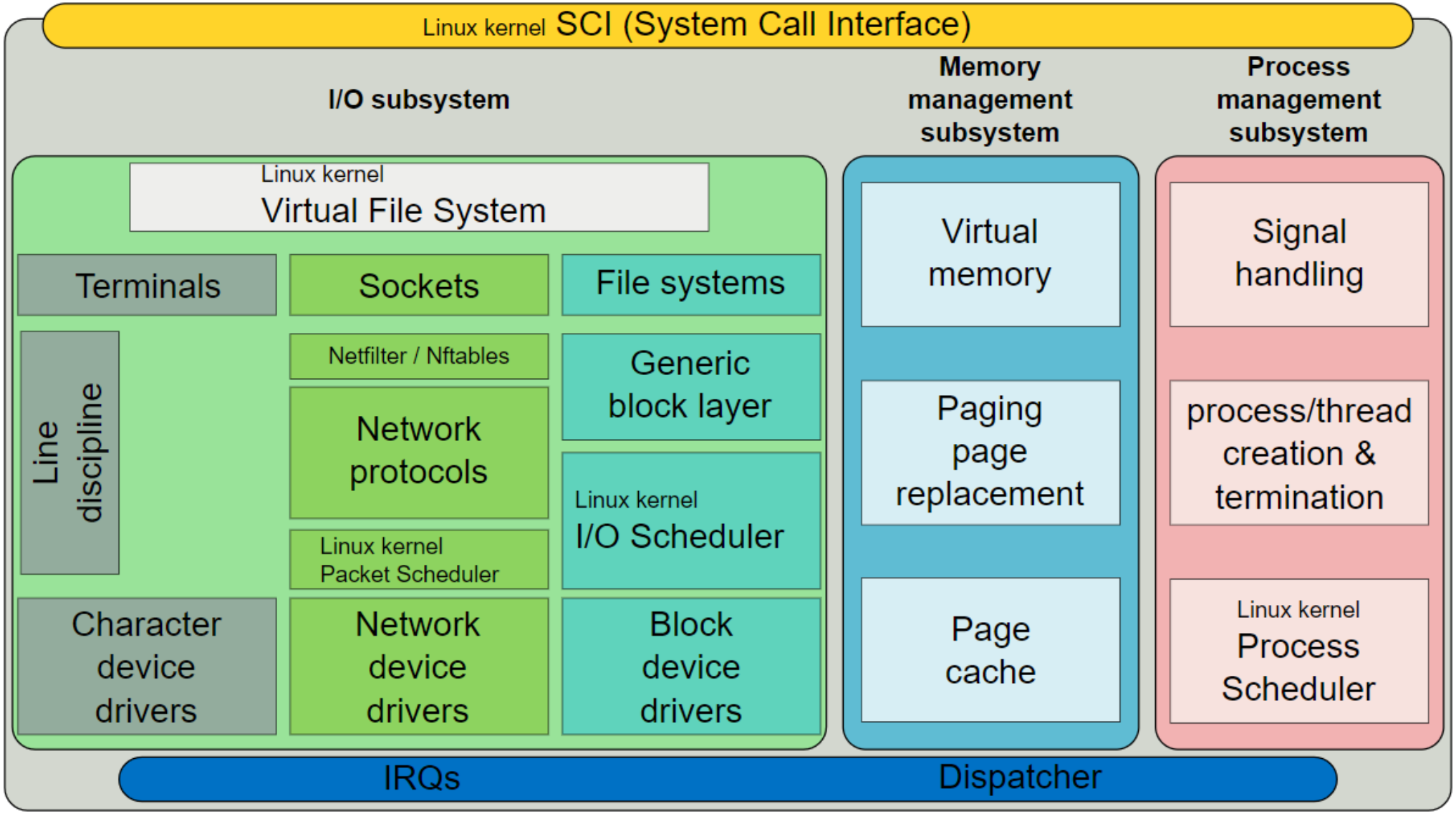
İşletim sistemi ile kullanıcı programları arasında tanımlı olan arayüz, işletim sistemi tarafından tanımlanan bir prosedürler kümesidir.
Sistem çağrıları çalışan program ile işletim sistemi arasındaki ara yüzü sağlar.
- Genellikle assembly dili komutlarıyla erişilebilir.
- Bazı yüksek seviyeli dillerde sistem çağrılarına doğrudan erişimi sağlar

İşletim sistemi tarafından tanımlanan bu prosedürlere sistem çağrıları(system calls) denilir.
Tanımlı olan işletim sisteminin sistem çağrıları kümesi işletim sistemlerinde farklı olabilir.
İsim olarak farklı olmasına rağmen arka planda gerçekleştirilen işlemler benzerdir.



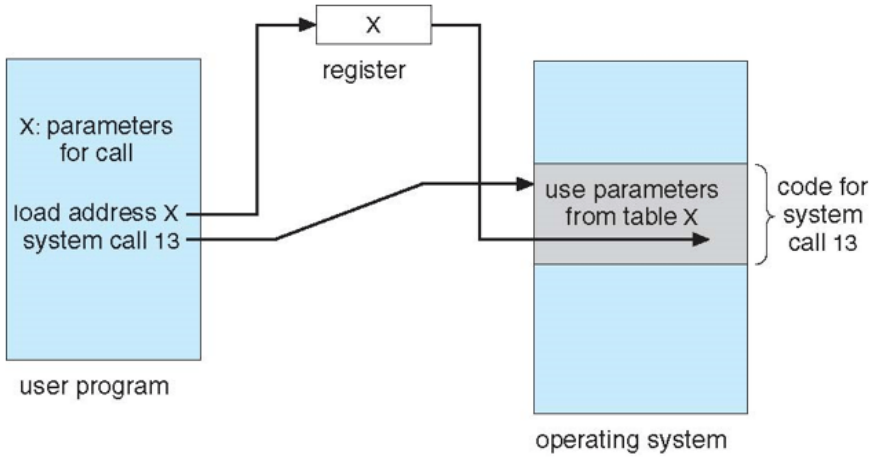
System Call ile İşletim sistemi arasındaki ilişki

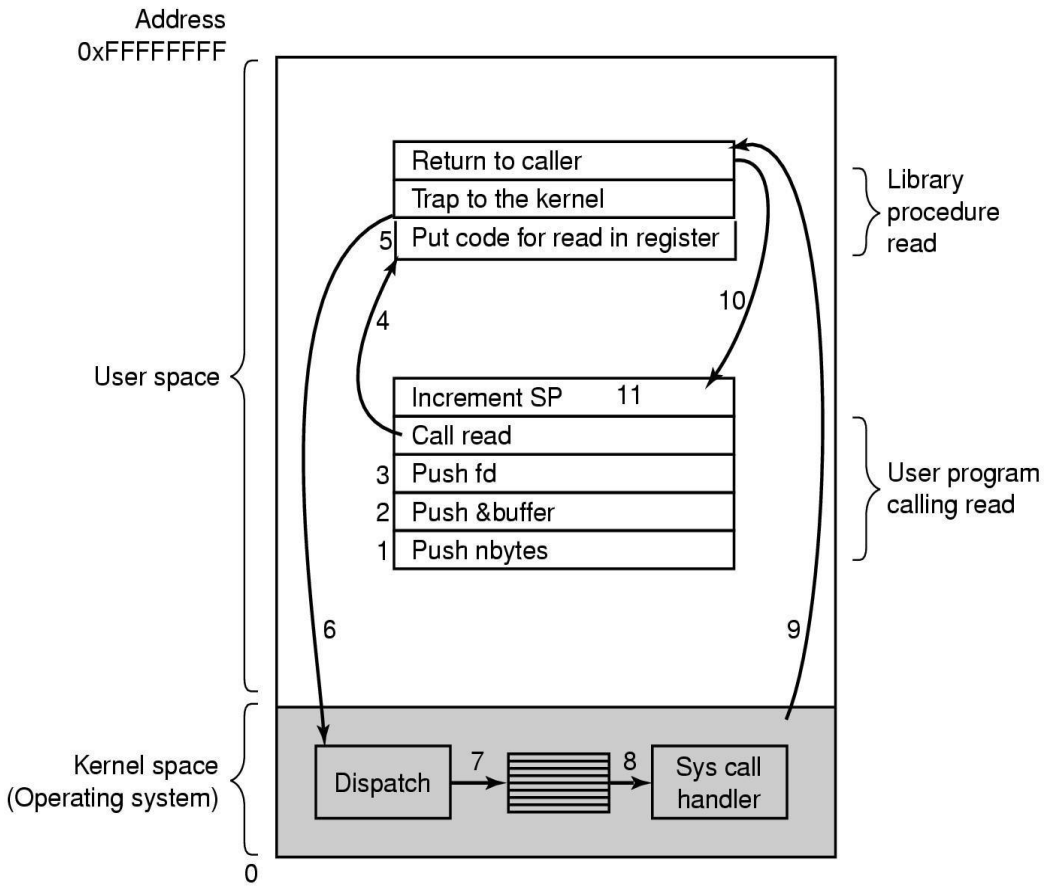
system call arayüzü user mode ile kernel mod arasında yer almakta, geçiş işlemini programcıya bırakmadan kendisi halletmektedir.


Linux kernel SCI (System Call Interface)
I/O subsystem
Memory management subsystem
Process management subsystem
Linux kernel
Virtual File System
Terminals
Sockets
File systems
Line discipline
Netfilter / Nftables
Network protocols
Linux kernel
Packet Scheduler
Generic block layer
Linux kernel
I/O Scheduler
Character device drivers
Network device drivers
Block device drivers
Virtual memory
Paging page replacement
Page cache
Signal handling
process/thread creation & termination
Linux kernel
Process Scheduler
IRQs
Dispatcher

Sistem Çağrısına Parametre Göndermek

Çalışan program ile işletim sistemi arasında parametrelerin gönderilmesinin üç temelyöntemi vardır:
-Parametrelerin yazmaçlarda gönderilmesi
-Parametrelerin bellekte, tabloda saklanması ve tablonun adresinin parametre gibi yazmaca gönderilmesi
-Parametrelerin programla yığına yazılması ve işletim sistemi tarafından yığından alınması.



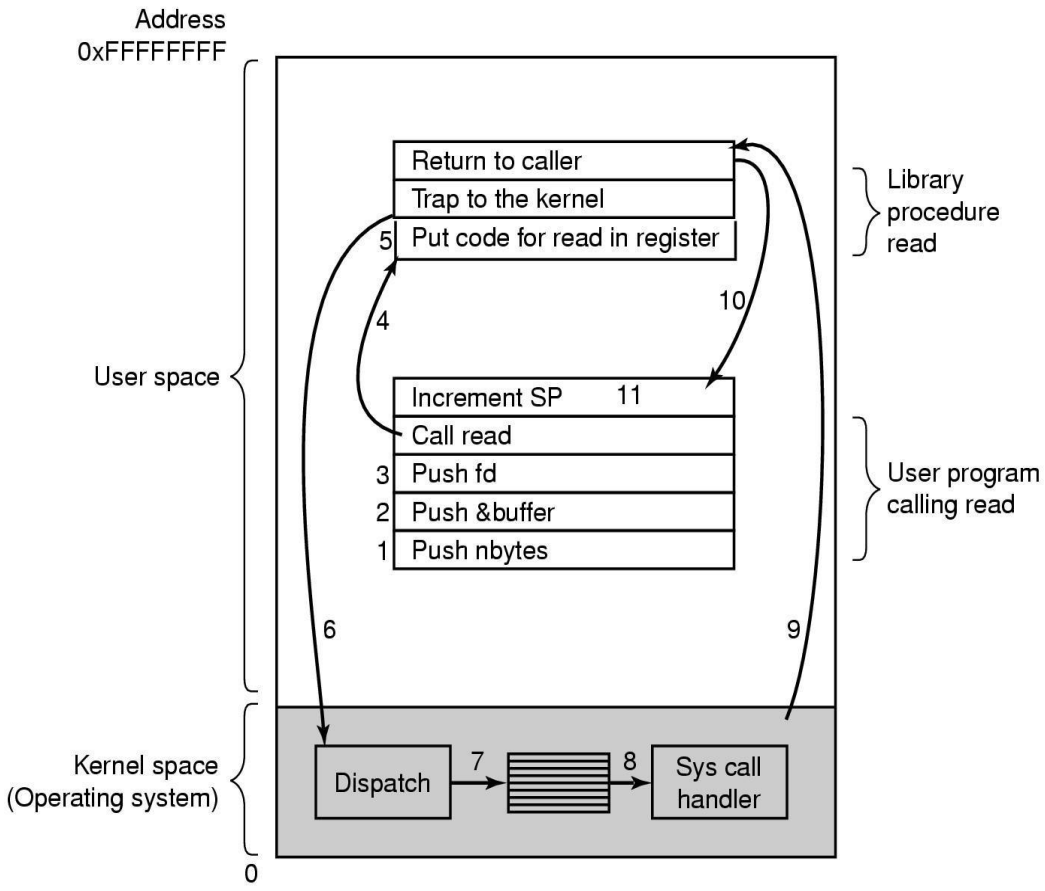


Sistem Çağrısına Parametre Göndermek

Sistem çağrısı mekanizmasını daha açık hale getirmek için, okuma sistem çağrısına hızlıca bir göz atalım. Bu çağrının üç parametresi vardır: ilki dosyayı belirtir, ikincisi arabelleği gösterir ve üçüncüsü okunacak bayt sayısını verir.

count = read(fd, buffer, nbytes);

Sistem çağrısı (ve kitaplık prosedürü), gerçekte okunan bayt sayısını sayı olarak döndürür. Bu değer normalde nbayt ile aynıdır.
Geçersiz bir parametre veya bir disk hatası nedeniyle sistem çağrısı gerçekleştirilemezse, count -1'e ayarlanır ve hata numarası global bir değişken olan errno'ya konur.
Programlar, bir hata oluşup oluşmadığını görmek için her zaman bir sistem çağrısının sonuçlarını kontrol etmelidir

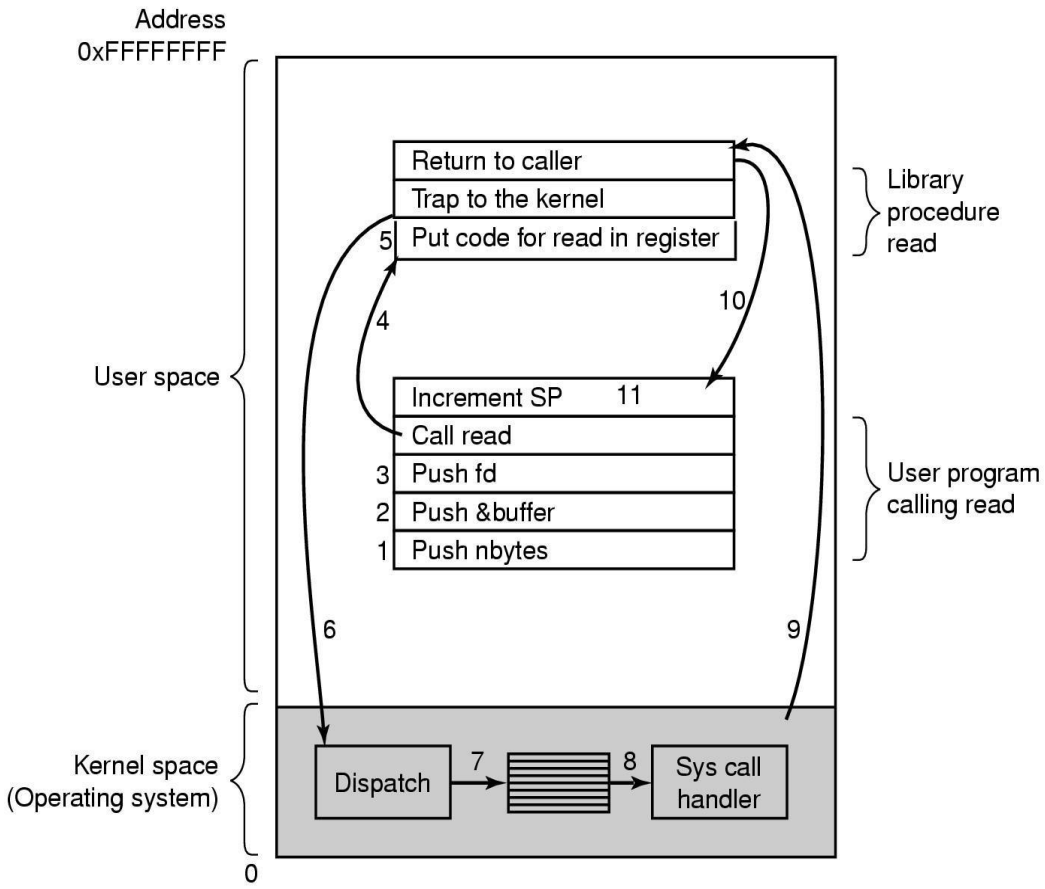


Sistem Çağrısına Parametre Göndermek

Sistem çağrıları bir dizi adımda gerçekleştirilir.

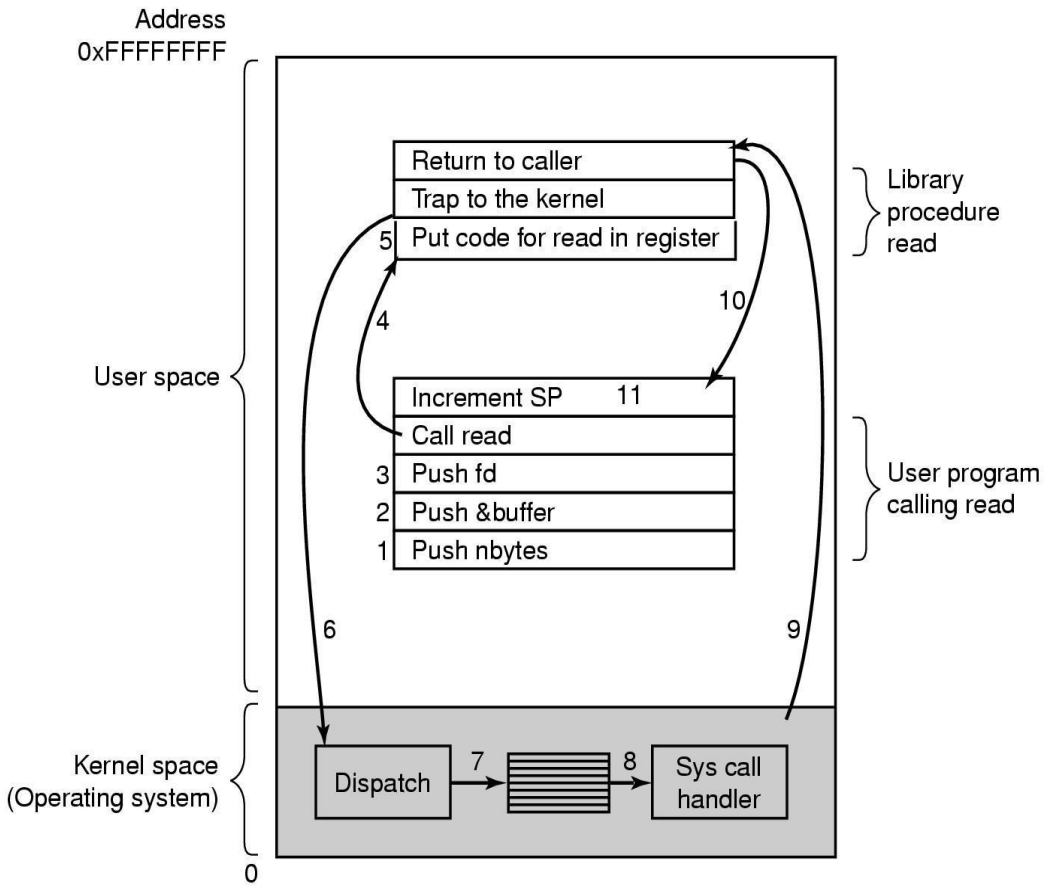
Gerçekten okuma sistemi çağrısını yapan okuma kitaplığı prosedürünü çağırmaya hazırlanırken, çağıran program ilk olarak Şekil 'deki 1-3. adımlarda gösterildiği gibi parametreleri yığına iter. C ve C++ derleyicileri, tarihsel nedenlerle parametreleri yığının (stack) üzerine ters sırayla ekler (push).

Birinci ve üçüncü parametreler değere göre çağrılır, ancak ikinci parametre referansla iletilir, yani tamponun adresi (& ile gösterilir) iletilir, tamponun içeriği değil. Ardından, kütüphane prosedürüne gerçek çağrı gelir (adım 4). Bu talimat, tüm prosedürleri çağırmak için kullanılan normal prosedür çağırma talimatıdır.



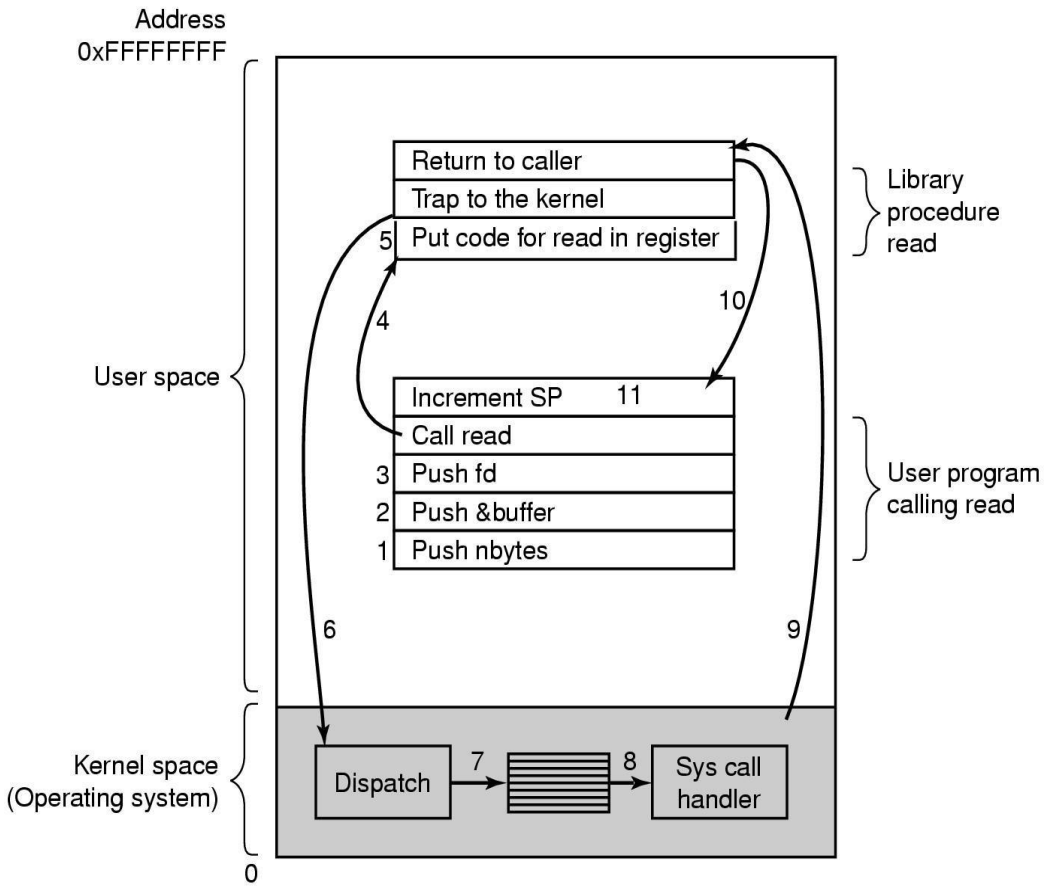
Sistem Çağrısına Parametre Göndermek

Muhtemelen assembly dilinde yazılmış olan kitaplık prosedürü, sistem çağrı numarasını tipik olarak işletim sisteminin beklediği bir yere, örneğin bir kayıt defterine koyar (adım 5). Ardından, kullanıcı modundan çekirdek moduna geçmek ve çekirdek içinde sabit bir adreste yürütmeye başlamak için bir TRAP talimatı yürütür (6. adım). TRAP talimatı, onu takip eden talimatın uzak bir yerden alınması ve dönüş adresinin daha sonra kullanılmak üzere yığına kaydedilmesi anlamında prosedür çağrısı talimatına oldukça benzer.



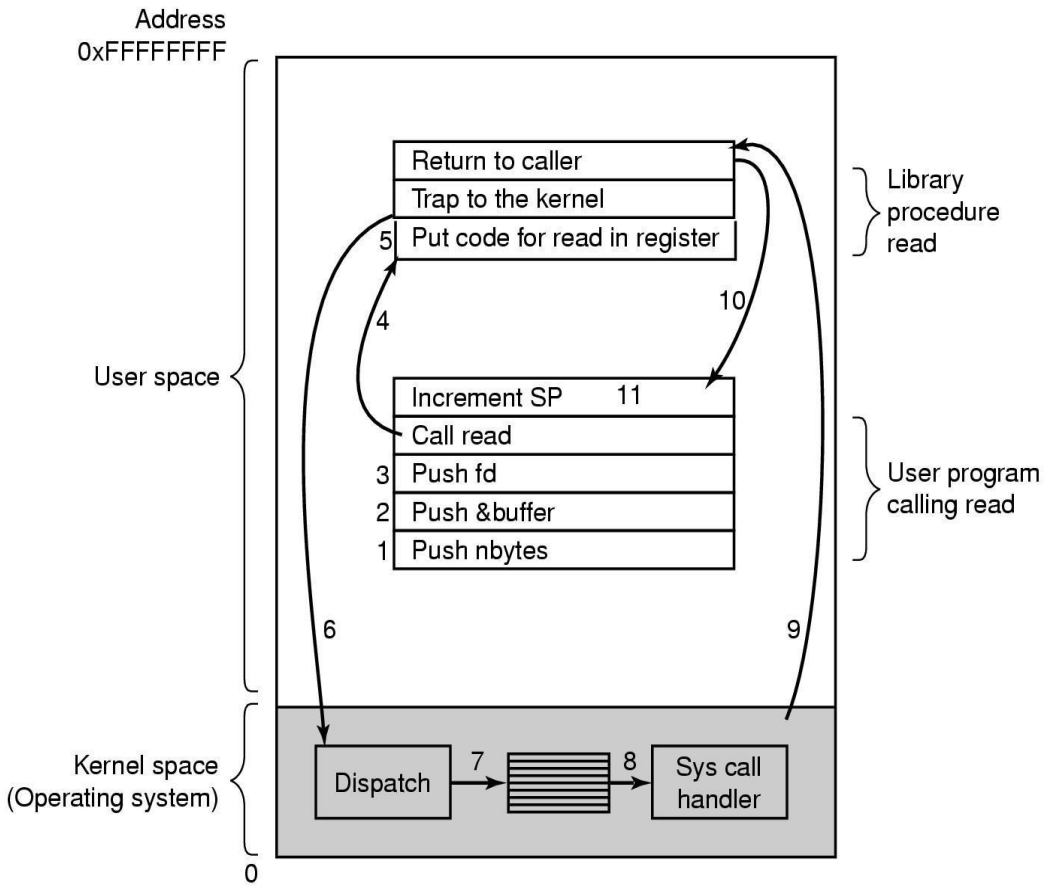
Sistem Çağrısına Parametre Göndermek

Bununla birlikte, TRAP talimatı, prosedür çağrı talimatından iki temel yönden farklıdır. İlk olarak, yan etki olarak çekirdek moduna geçer. Prosedür çağrısı talimatı modu değiştirmez. İkincisi, prosedürün bulunduğu yere göreli veya mutlak bir adres vermek yerine, TRAP komutu rastgele bir adrese atlayamaz. Mimariye bağlı olarak, ya tek bir sabit konuma atlar ya da komutta 8-bitlik bir alan vardır ve indeksi bellekte atlama adreslerini veya eşdeğerini içeren bir tabloya verir.



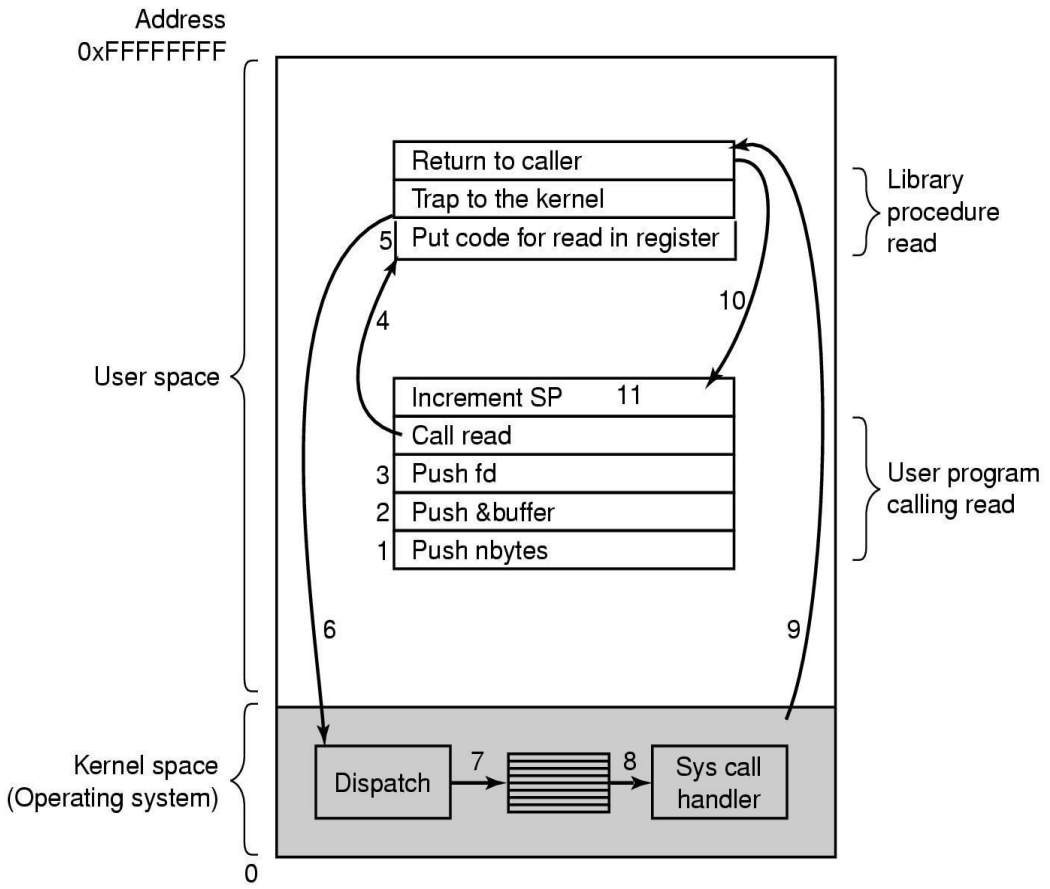
Sistem Çağrısına Parametre Göndermek

TRAP'den sonra başlayan çekirdek kodu, sistem çağrı numarasını inceler ve ardından doğru sistem çağrısı işleyicisine, genellikle bir işaretçi tablosu aracılığıyla sistem çağrı numarasında indekslenen sistem çağrısı işleyicilerine gönderir (adım 7). Bu noktada sistem çağrısı işleyicisi çalışır (adım 8). Çalışmasını tamamladıktan sonra, TRAP talimatını izleyen talimatta (adım 9) kontrol, kullanıcı alanı kitaplığı prosedürüne geri döndürülebilir. Bu prosedür daha sonra, prosedürün geri çağırma çağrılarının olağan şekilde (adım 10) kullanıcı programına geri döner.



Sistem Çağrısına Parametre Göndermek

İşi bitirmek için, kullanıcı programı, herhangi bir prosedür çağrısından sonra yaptığı gibi yığını temizlemelidir (adım 11). Çoğu zaman olduğu gibi yığının aşağı doğru büyüdüğünü varsayarsak, derlenmiş kod yığın işaretçisini, okuma çağrısından önce gönderilen parametreleri kaldıracak kadar tam olarak artırır. Program artık daha sonra yapmak istediğini yapmakta özgürdür.



Sistem Çağrıları(System Calls)

Sistem Çağrılarının Türleri

- İşlem Kontrolü
- Dosya Yönetimi
- Cihaz Yönetimi
- Bilgi Sağlama
- İletişim
- Koruma

POSIX tarafından tanımlanan, birkaç sistem çağrısını inceleyelim.

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

Directory and file system management

Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

Miscellaneous

Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since Jan. 1, 1970

Süreç Yönetimi için Sistem Çağrıları

fork() sistem çağrısından başlayabiliriz. fork() sistem çağrısı UNIX sistemlerde yeni bir süreç oluşturmak için tek yoldur.

Bu fonksiyon asıl sürecin bire bir kopyasını oluşturur (dosya tanınlayıcıları, yazmaçlar,... herşey).

Kopyalama işleminden sonra iki süreç (ana ve çocuk) birbirlerinden tamamen ayrılırlar.

Kullandıkları veriler kendilerine özgü olur. (Fakat programın text kısmı aynı olduğu için iki süreç tarafından paylaşılır)

Örnek (Linux)

print-pid.c

```
#include <stdio.h>
#include <unistd.h>

int main(){
    printf("süreç numarası : %d\n", (int)getpid() );
    printf("Ana sürecin süreç numarası:%d\n", (int)getppid() );
}
```

-getpid(): geriye sürecin süreç numarasını çevirir.
*-getppid():*çalışan sürecin ana sürecinin süreç numarasını geri çevirir. (get_parent_program_id)

Linux'da çalışan süreçleri ***ps*** komutu ile öğrenebilirsiniz.
$ps -e -o pid,ppid,command

fork() ve exec() in kullanımı

fork() : mevcut sürecin birebir kopyasını oluşturur, iki süreçde fork() fonksiyonundan sonraki satırdan itibaren kendi başlarına çalışmaya devam eder.

exec(): fonksiyonları kümesi, mevcut sürecin çalıştırmak istenilen başka bir programın sürecine dönüşmesini sağlar.

fork() ve exec() in kullanımı

fork.c

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

int main(){
    pid_t cocuk_pid;

    printf("Ana sürecin pid = %d\n", (int)getpid() );

    cocuk_pid=fork(); if (cocuk_pid!=0){
        printf("burası ana sürectir, süreç id
        pid=%d\n",(int)getpid());
        printf("çocuk sürecin idsi pid =
        %d\n",(int)cocuk_pid);
    }else{
        printf("burası çocuk süreçtir, pid=%d\n",
        (int)getpid());
    }
    return 0;
}
```

fork-exec.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
//fonksiyon yeni bir programı yumurtlar(spawn)
//yani calistirir, mevcut süreci bu programa cevirir
int spawn(char *program,char** arg_list){
    pid_t cocuk_pid;

    cocuk_pid =fork();
    if (cocuk_pid!=0){
          return cocuk_pid;
    }else{
          execvp(program,arg_list);
          //eger hata olmus ise alt kisim calisir
          fprintf(stderr,"execvp de hata olustu\n")
          abort();
    }
}
int main(){
    char * arg_list[]={"ls", "-l","/", NULL};
    spawn("ls",arg_list);
    printf("basarili olarak ana program bitti\n");
    return 0;
}
```

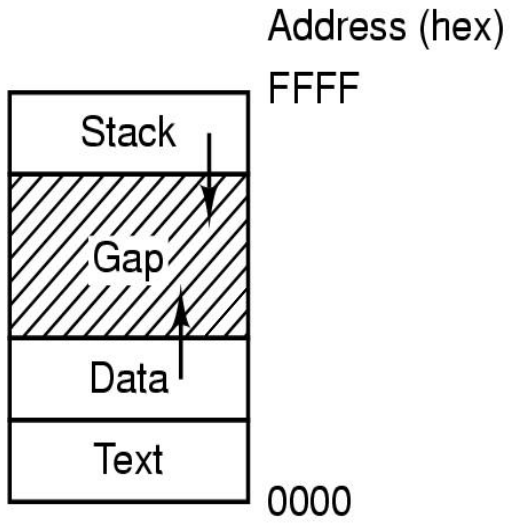
Süreçlerin Bellek Görünümü

UNIX temelli sistemlerde bellek üç kesime(segment) parçalanmıştır.

Veri Kesimi(Data Segment) : değişkenler

Stack Kesimi(Stack Segment): malloc() ile ayrılan yerler

Yazı Kesimi(Text Segment): Program kodunun olduğu bellek bölümü



Windows Win32 API

Programlama modelleri Windows ve Unix türevi işlerim sistemlerinin farklıdır.

Bir UNIX programı belirli bir görevi yerine getiren ve bu görevi yerine getirirken sistem çağrılarını kullanan bir programdır.

Bir Windows programı ise olay temellidir.Ana program belirli olayların gerçekleşmesini bekler. Bu olay gerçekleştiğinde de, olayı işleyecek(handle) olan prosedürü çalıştırır. Windowsunda sistem çağrıları bulunmaktadır. Unix sistemlerde genellikle sistem çağrısı ile çağrılacak olan kütüphane fonksiyonunun ismi aynıdır. Bu isimler POSIX tarafından tanımlanmıştır.

Windowsda durum bu şekilde değildir. Microsoft Win32 API (Application Programin Interface) adını verdiği bir prosedür kümesi tanımlamıştır. Programcılar işletim sisteminin servislerini kullanmak için bu prosedürleri kullanırlar. Bu arayüz tüm Windows işletim sistemleri tarafından kısmi olarak desteklenmektedir.

Yeni windows sistemlerinde bu prosedürler ve kullanımları farklılaştırılmaktadır.

UNIX	Win32	Description
fork	CreateProcess	Create a new process
waitpid	WaitForSingleObject	Can wait for a process to exit
execve	(none)	CreateProcess = fork + execve
exit	ExitProcess	Terminate execution
open	CreateFile	Create a file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	Read data from a file
write	WriteFile	Write data to a file
lseek	SetFilePointer	Move the file pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory
rmdir	RemoveDirectory	Remove an empty directory
link	(none)	Win32 does not support links
unlink	DeleteFile	Destroy an existing file
mount	(none)	Win32 does not support mount
umount	(none)	Win32 does not support mount
chdir	SetCurrentDirectory	Change the current working directory
chmod	(none)	Win32 does not support security (although NT does)
kill	(none)	Win32 does not support signals
time	GetLocalTime	Get the current time