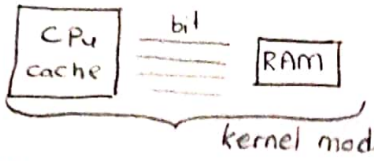
OPERATING SYSTEM

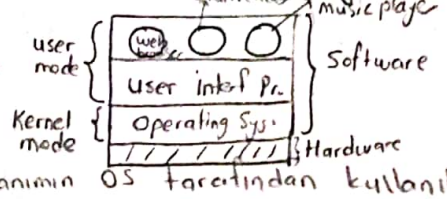
kernel mode }
user mode }

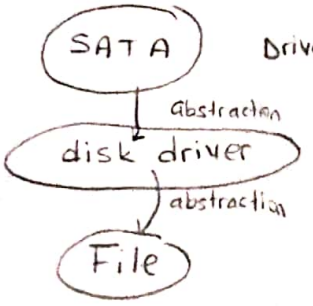


cache = internal memory (ön)
abstraction = soyutlama

ARASTIR
Disk, Gui
kernelmode
user mode
#disk = SSD
SATA
Spooling
time sharing
CTSS
MULTICS
Cloud computing
UNIX, POSIX
LSI
HDD / SSD

Uygulama sadece user mode'da yapılabilir.
Gui'den (Graphical User Interfaces) önce shell kullanılıyordu.
Gui user mode'da yapılır.
I/O kernel mode'da yapılır.
Trap (tuzak) " " " .





Driver: Herhangi bir donanımın OS tarafından kullanılmasını sağlayan arayüz.

multiplexing: Var olan kaynakları eş zamanlı çalışan uygulamalar arasında paylaştırmak

accomplish = başarmak, üstesinden gelmek

İşlemci mimarisi ⇒ ARM, X86 (AMD'de bu gruptal)

PSW (Program status word) // Register
- Comparison bit
- Condition bit
- Priority bit
- Kernel bit

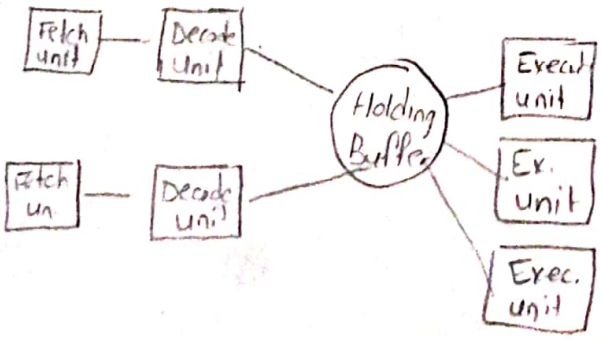
J1, J2, J3, J4 — J1 bilmeden J2'ye geçer ve J1'deki veriler hafızada saklanır. Bilmeye, jobbar hafızada saklanır.

pipeline

Fetch - Decode - Execute } A three stage pipeline

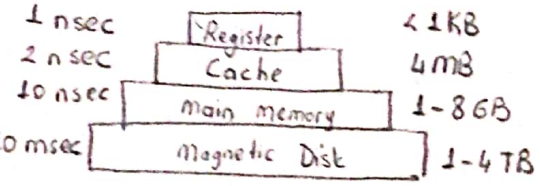
A superscalar CPU

Birden fazla execute unit var.

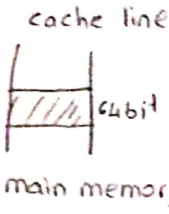


TRAP: → kernel
- system call
- user prog. // Kullanıcı programı I/O işlemi yapmak istediği zaman bunu direk yapamıyor protectiondan dolayı bu yüzden system call yapar.

System calls: user mode cannot enter kernel mode. Need to make system call which traps into the kernel and invokes OS. (yardım isteme)



09.10.2019



AMD'nin kullandığı yapıda L2 cache bütün corelarda farklı ve bağımsız olduğu için bütün corelardaki bilgiler barındırılmalı

Intel'in kullandığı yapıda L2 cache ortak olduğu için bir bilgi tek kopyalı (AMD'de aynı bilgi 4 defa bulunuyor)

trade-off (Bir yerden kazanırken bir yerden kaybetmek.)

cache miss: aranan bilgi registera bakıldı. Yok L1 cache bakıldı yok. L2 de yok. main memory'e bakılması en son

cache hit:

evicted:

magnetic disk = Hard disk ~ SSD

ROM: Ucuz, içindeki bilgi silinemiyor. Bilgisayarda çok kullanılan bilgiler ROM'da saklanabilir.

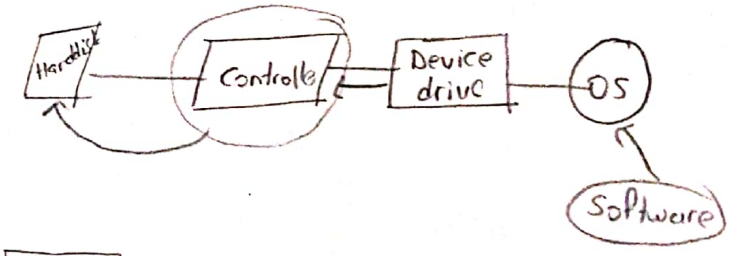
EEROM: içindeki bilgiler silinebiliyor

Flash memory: USB, hem yazılıp hem okunabilir
Çok sık silinip yazılırsa cihaz yıpranıyor. (2)

CMOS: Bir tür RAM
Çok az enerji harcıyor.
Bilgisayarda hangi hard diskten boot edilmesi gerektiğini, tarih-gün bilgisi tutuyor.

Disk  3mil
Her trackte ne kadar sector olduğu değişken.

MMUC memory management unit):
Cache'de nereye yazılacak, neresi silinecek, Ram'e mi gidilecek bunu belirliyor.
I/O: işlemci ve hafıza dışındaki her şey I/O cihazı



CPU → CPU bound: CPU erişimi gereken instructionlar.
→ I/O bound: I/O cihazlarına erişmesi gereken instructionlar.

shared / paralell bus → 32 bitlik bilgi aynı anda 1'er bit ile gidiyor. Bilgi senkronize gitmek zorunda ve bu da çok zor.

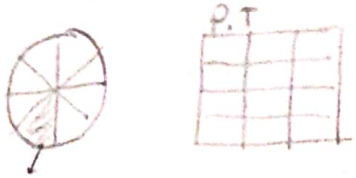
Yeni teknolojide serial bus → Bilgi paket gibi gidiyor.

USB 3.0 → 5.6 Gbps

SCSI bus → Çok yüksek I/O kapasitesine sahip. 640 Mb/s

☆ plug and play:

Boot: Bilgisayarın açılışı
Bios chip: low level I/O software
Partion table



Her sektörün bilgisi bulunuyor

Dıştaki partionlar daha hızlı, içtekiler daha yavaş

mainframe OS → Yüksek I/O kapasitesi
(1000 tane diski yönetebiliyo)

Server OS → mainframe göre daha düşük I/O kapasitesi (Yine de yüksek kapasiteli)
Printer paylaştırılması, dosya paylaşımı, büyük sitelerin üzerinde bulunduğu işletim sistemi.
Solaris, Free BSD, Linux

Multiprocessor OS → Resource management

Handheld Computer OS → Android, iOS

Embedded OS → Evdeki eşyalar, mikrodalgalar
Dışarıdan müdahale yok.
içerideki OS buglardan temizlenmişse ölene kadar kullanılır.

Sensor Node OS: IdT → sensörler
Çok az enerji harcayan tipte olmalı çünkü pille çalışıyo. TinyOS

Real-Time OS → Hard real time
Aşırı hassas olmalı.
(Örneğin bir araba fabrikasından robot kolu için yazılan OS)

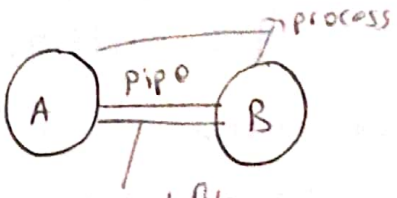
Soft real time
Aşırı hassas olmasına gerek yok
Elimizdeki cep telefonları.

Smart-card OS → Akbiller, öğrenci kartları
Enerjiyi okuttuğu zaman alıyor.
Jum

UNIX: 2 tip dosya sistemi

① Block special files:
Harddiskte data saklayan .dev klasörü altında

② character special files:
karakter işleme yapabilen cihazlar için, printer, modem ...



pseudofile

A'dan B'ye yazma yapılacaksa pipe'a dosya aktarılıyor.

(rwx)(rwx)(r-x) → protection
owner group bütün kullanıcılar

drum: bir tür hafıza

The operator = user

Normalde driverlar kernel mode'da çalışmak zorunda ama microkernel mimaride öyle değil. Kernel minimize ediliyor. (mikrokernel yapıda). İşletim sisteminde oluşan bugların temizlenme olasılığı az.

hypervisor → virtual machine

Type 1 → hypervisor'ın altında bulunan donanıma uygun olmayan işletim sistemi biri kullanılırken diğeri kullanılamıyor.

Type 2 → aynı anda birden fazla OS çalışabiliyor.

Type 3 → hypervisor ana işletim sisteminin kernel'ine kısıtlı biçimde ulaşabiliyor.

Jvm → bir tür OS mimarisi

PROCESSES AND THREADS 16.10.19

- The Process Model (1)

Her bir işlemcide bir process çalışır. Buna Pseudoparallelism denir paralellik var gibidir ama aslında yoktur.

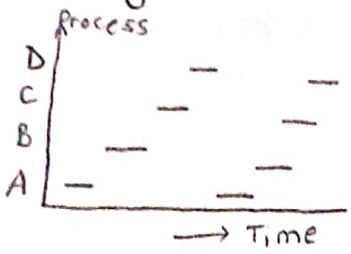
Program counter'a göre processlerde işleniyor.

Program, diskinizde saklı dosya bazlı yazılımdır bunu her çalıştırdığınızda process çalıştırmış olursunuz.

Processlerin kendilerine ait adres uzayı var ve başka bir process buna erişemiyor.

Bozulma ve sistem güvenliğini bu şekilde sağlıyoruz.

Çoklu işlemli program için bu mesela 8 çekirdeği varsa 8 tane çekirdek aynı anda çalışıyor



(syf 87)

Programla, process arasındaki fark pasta yapan bir programcının pasta yapması process sonra kızını arı soktu (interrupt) araya kesme girdi process durdu. Eline ilk yardım yapmak için kitap aldı okumaya başlayınca başka bir process yaratıldı. Kızını tedavi etti sonra pastaya devam etti.

Program çalıştırıldığında process olur. Aksi halde diskte yatan bir yazılımdır.

Process Creation

① Sistemin başlatılması ve birçok processin çalışmaya başlaması

/* Backgroundda kullanılan processlere daemon denir. */

UNIX → ps diye program ve
windows → Task managerdan

Arka planda çalışan daemonları görebilirsi.

② Çalışan bir process tarafından bir process yaratma sistem çağrısı yapılması.

child process: main process child processi yaratır. Processler arasında hiyerarşi var. Ana processin adres uzayı kopyalanarak yaratılıyor. (Bütün datalar ve register içerikleri adres uzayını oluşturuyor.) child process kendince ana processin adres uzayını değiştiriyor. Ana processin child'a erişimi engelleniyor.

③ Bir kullanıcının isteği amacıyla

④ Batch jobın başlatılması

Batch job: Birçok işin aynı anda yapılması → system call

UNIX → fork /* O anda çalışan main processin birebir klonu yaratılıyor. */

execve: Başka bir sistem çağrısı ile child process yeni memory image (adres uzayı) yaratıyor.

init: 1. durumda sistem başlatılması durumunda oluşan processlerin yaratılması için init sys. call kullanılıyor.

windowsta bunun için sistem çağrısı yok. Hiyerarşi yok.

token (handle): Bunu kurarak ilgili child processi kontrol edebiliyor. Bu handle'ı başka bir processe de aktarabiliyo-

Process Termination (Processlerin Bitirilmesi)

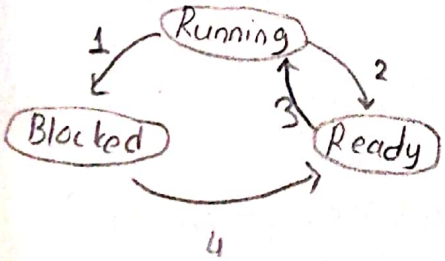
① Normal cikis (voluntary)
isteğe bağlı.
UNIX → exit sistem cagrisiyla
Windows → ExitProcess sistem cagrisiyla process yok edilebiliyor

② Bir hata olması durumu, mesela deneme.c 'yi derliyorsunuz fakat bu dosya diskte bulunamadı bu durumda c derleyicisi hata modunda cikacak.

③ Fatal error olması durumu
Sifira bölünme, sistem out olur. Var olmayan hafiza gözüne erisimyine fatal error verir.
Bu processin kullanmaması gereken bir instructiona erismeye calismasi da fatal error dur.

④ Baska bir process tarafından o processin öldürülmesi
UNIX → kill
Windows → TerminateProcess

Process States



1. durum: I/O bazlı bir instruction gelirse bloke duruma geciyor. Gelecek olan dataya ihtiyaci var.
2. durum: Process verdigimiz zamani kullandı ama isi bitmedi ready konumuna gecer hazir ama calisamiyor. Baska bir processe zaman verilir.

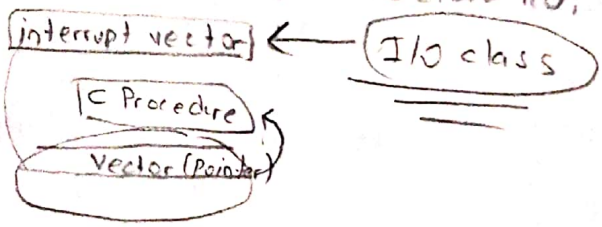
③ Süresi bitip ready olan process tekrar calisir moda gecti

④ I/O'dan data geldi hazir duruma gecer o sirada bitmemis olabilir sirasi geldiginde tekrar calisir moda gecer

Processler arasi gecisi yapan mekanizmaya process scheduler denir.

Process table: Her bir process (Process Control Blocks) entrysi için syf 95 2.b tablosundakiler vardır.

Her bir processin bağlı olduğu kullanabileceği I/O sınıfı var bu interruptların istenebilmesi için gerekli
Her bir I/O classın kendine ait interrupt vectoru var hafizanin ait kismında her bir I/O class için bir interrupt vectoru var.



interrupt service procedure ilgili interruptun olması durumunda calistirilacak yönelge.
mesela 3. no'lu process calisirken ibir disk interrupti olustu, 3. processin içeriği, registerları interrupt hardware tarafından stack'le saklaniyor. Interrupt vectorunde sakli adres neyse ona gidip ilgili alandaki C procedure calisiyor.

C procedürü // isletim sistemleri genelde C ile yaziliyor.

Multiprogramming

p → multiprogramming degree

Aynı anda kaç process yaptığını belirtir.

n=3 ise 3 process çalışıyor demektir.

Bir tasarım durumu var. Diyelim ki bir process ortalama toplam kullanım süresinin oranı kadar I/O veriyor. Buna p diyoruz. (1/3)

n adet process birbirinden bağımsız olmalı birbirinin child'ı olmamalı.

8 GB RAM'li bir bilgisayar var diyelim

OS + Table → 2 GB

→ User Progam → 2 GB

Bu durumda hafıza 3 process'i tutabilecek 2'si işletim sistemi

$= 1 - (0,8)^3 = \%49$

8 GB daha RAM eklersek bu durumda yine 3 process olursa

$= 1 - (0,8)^7 = \%79$ → 3+4 (8gb)

8 daha eklersek %91 oluyor.

P: input bekleme zamanı

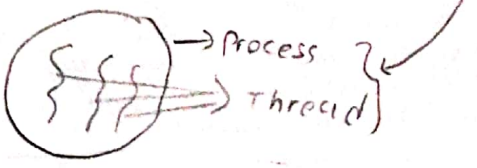
$= 1 - (0,8)^{(7+4)} = \%91$

THREAD USAGE

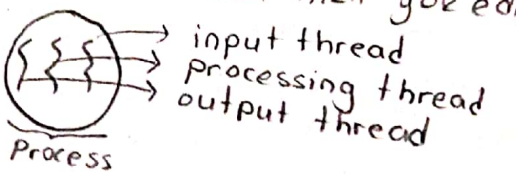
Hafif sıklet process olduğu için kullanıyoruz. Kullanabilmek için threadleri destekleyen işletim sistemi ve işlemci gerekir. Processleri threadlere bölmeye çalışıyoruz. Diyelim ki 600 sayfalık dökümanla uğraşıyorsunuz 200. sayfasını sildiniz tüm sayfaları yukarı kayacak eğer bu işlemlere process atasaydık diğer işlemler yapılamayacaktı. Thread yapısı ile bunları threadlere atıyoruz.

Bir processin içinde birden fazla thread kullanabiliyor (multithreading veya hyper threading denir.)

Tüm threadler processin uzayına erişebiliyor, veri iletişimini gerçekleştirmiş oluyorlar.



Threadler processlere göre 10 ile 100 kat daha hızlı yok edip yaratılır.



Threadler arasında hiyerarşi yok. Kendisine ait stacki var. Herbir thread birbirine erişiyor bu yüzden processler arasında olan güvenlik burada sağlanmıyor.

The Classical Thread model (2) syf 102...

Threadlerde ; Running, readingi blockeda ek olarak Terminated var. Process de terminate (isi bitmis processin hala saklanıyor olmasi) gerekli degil. Threadleri bitirseniz bile tekrar kullanabilirsiniz kendi adres uzaylari yok.

The Classical Thread model (3)

Her bir cagrilan fakat geri dönemeyen procedure

Procedure, function return olmadi variables & return adres saklamak için her bir threadin stacinde entry var.
multi thread yapilada ilk asamada tek threadli process yaratiliyor daha sonra ilgili kütüphane prosedürleri ile yeni bir thread yaratilir.
library procedure ← thread create
Thread isini bitirdiyse thread_exit ile threadi durdurabiliyorsunuz.
Ayni sekilde thread_join ile yeni yaratilacak threadin baska bir threadin cikisini beklemesi sağlaniyor.
Thread_yield üzerinde yaptirim olmayan bir thread yerini baska bir threade verebiliyor. Kendi kendine calismayi sonlandirabiliyor.