Erciyes Üniversitesi
Bilgisayar Mühendisliği Bölümü

BZ 313 Yazılım Mühendisliği
15. Performans

Performans derken neyi kastediyoruz?

- Sözlüğe baktığımızda…
  - *Cambridge*: Bir makinanın veya insanın bir işi ne kadar iyi yaptığı
  - *Macmillan*: Bir makinanın veya insanın hızı veya etkinliği

- Yazılım QA için: fonksiyonel olmayan gereksinimdir (**kalite özelliği**)
  - Dar açı: bir programın **hızı**
  - Geniş açı: bir programın **efektifliği**
  - Bu bölümde, performansa daha geniş bir açıyla bakacağız

Ancak «hız» tanımı bile oldukça zordur

- Dar bir açıyla bile performansı (hız) tanımlamak zordur
- *Web tarayıcı* için hız
  - Bir web sayfasının kullanıcı etkileşimlerine ne kadar hızlı yanıt verebildiği (sayfa yükleme, buton tıklama, sürükleme, yazma vs.)
  - Cevaplanabilirlik ortalama cevap süresi (response time) ile ölçülür
- *Web sunucusu* için hız
  - Bir sunucunun bir sayfa isteğine ne kadar sürede cevap verdiği
  - Daha da önemlisi, saniye başına servis edilen sayfa sayısı (a.k.a. **throughput**)
  - Tepki süresi bir eşikten az olduğu sürece (<100ms gibi) web sunucu performansı response time yerine throughput ile ölçülür.
- Performansı ölçmek için birden fazla metriğe ihtiyacımız bulunmaktadır.

Performans Göstergeleri

- Bir system under test'e (SUT) ait nicel ölçümlerdir
- Örnek (dar açıdan, hız):
  - Bir butona basıldığında yanıt verme süresi nedir? (*response time*)
  - Aynı anda kaç kullanıcıya hizmet verebilir? (*throughput*)
- Örnek (geniş açıdan)
  - Sistem hata almadan ne kadar çalışabilir? (*kullanılabilirlik*, *availibility*)
  - Veritabanındaki standart bir sorgu ne kadar CPU kullanır? (*kullanım*, *utilization*)
  - Program MB türünde ne kadar hafıza kullanır? (*kullanım*, *utilization*)
  - Program saniye başına kaç watt enerji harcar? (*kullanım*, *utilization*)

Anahtar Performans Göstergeleri
Key Performance Indicators (KPIs)

- **KPI**: bir performans göstergesi kullanıcılar için önemlidir
- Yalnızca gerçekten önemli olan birkaç KPI seçin
  - Bunlar yazılımın başarılı veya başarısız olduğunu göstermelidir
  - Ör. Hibrit araçlar için km başına düşen litre KPI olmalıdır
  - ÖR. Formula 1 araçları için km başına düşen litre KPI olmamalıdır
  - Ayrım gözetmemek, önemli performans hedeflerinin zarar göreceği anlamına gelir
- **Performans hedefi**: KPI'ın ideal olarak ulaşması gereken nicel ölçü
- **Performans eşiği**: bir KPI'ın ulaşması gereken minimum değer
  - Minimum değere ulaşıldığında üretime hazır demektir
  - Performans hedefine kıyasla daha gevşek bir hedeftir

KPI / Performans Hedefi / Performans Eşiği

- Bir web uygulamasına ait gereksinimleri geliştiriyor olduğunuzu düşünelim
- Örnek KPI / Performans hedefi / Performans eşiği değerleri:
  - KPI: response time
  - Performans hedefi: 100 ms
  - Performans eşiği: 500 ms
- Bir başka örnek: KPI / Performans hedefi / Performans eşiği değerleri:
  - KPI: throughput
  - Performans hedefi: 100 kullanıcı isteği/saniye
  - Performans eşiği: 10 kullanıcı isteği/saniye

Performans Göstergeleri: Kategoriler

- Performans göstergelerinin temel olarak iki kategorisi bulunmaktadır

- Servis Odaklı (Service-Oriented)

- Verimlilik Odaklı (Efficiency-Oriented)

Servis Odaklı Performans Metrikleri

- Sistemin kullanıcılara ne kadar iyi hizmet sağladığını ölçer
  - Son kullanıcı deneyimini ölçer
  - İçeride gerçekleşenlerden bağımsızdır (blackbox test gibi)
- İki Alt Kategori:
  - Cevap Zamanı (Response Time)
    - Kullanıcı isteğine ne kadar hızlı cevap verebilir?
  - Kullanılabilirlik (Availability)
    - Kullanıcı zamanın yüzde kaçında sisteme erişim sağlayabilir?

Verimlilik Odaklı Performans Göstergeleri

- Bir sistemin hesaplama kaynaklarını ne kadar iyi kullandığını ölçer
  - Sistemin iç işleyişinin ölçümüdür
  - Son kullanıcının neden iyi/kötü deneyim sahibi olduğunu gösterebilir
- İki alt kategori:
  - Kullanım (Utilization)
    - Sistem ne kadar kaynak kullanıyor?
  - Çıktı (Throughput)
    - Verilen zaman diliminde ne kadar istek işlenebiliyor?

Servis Odaklı vs. Verimlilik Odaklı

- Günün sonunda, önemli olan kullanıcı deneyimidir (servis-odaklı)
- Ancak verimlilik odaklı göstergeler doğrudan kullanıcı deneyimini etkiler
  - Uygulama çok fazla CPU zamanı kullanıyor→ *cevap zamanı problemi*
  - Uygulama çok fazla hafıza tüketiyor→ *kullanılabilirlik problemi*
  - Uygulama talep artışını karşılayamıyor → *cevap zamanı problemi*
  - Uygulama verimi kronik olarak düşük → *kullanılabilirlik problemi*
- Servis tabanlı göstergeler kullanıcının memnuniyetsizliğini gösterebilir
- Verimlilik tabanlı göstergeler kullanıcıların neden memnuniyetsiz olduğunu gösterebilir
  - Ve problemin çözüm yolunu da (ör. Ekstra hafıza ekle)

Servis Tabanlı Performans Göstergelerini Test Etme

Response Time / Availability
Tepki Süresi/Kullanılabilirlik

Response Time Testi

- Gerçekleştirmesi oldukça kolaydır!
  - Herhangi bir şey yap
  - Kronometre başlat
  - Cevabı bekle
  - Kronometreyi durdur
  - Kronometredeki sayıyı yaz!
- Peki bu yaklaşımla ilgili problem bulunuyor mu?

Tepki süresinde manuel test problemleri

1. Saniye altındaki cevap zamanlarının ölçümü imkansızdır
2. Son kullanıcıya gösterilmeyen cevapların tepki süresi ölçümü imkansızdır
3. İnsana bağlı hatalar meydana gelebilir
4. Zaman alıcıdır
5. Ve muhtemelen bunu yapan insan için oldukça sıkıcıdır

Performans testi otomasyona ve istatistiklere sıkı sıkıya bağlıdır

Neden istatistik?

- Performans testinde asla tek bir sonuca güvenme
  - Her zaman birden fazla sonuç al ve ortalamasını raporla
  - Ayrıca min/max değerleri arasındaki varyansı da incele

- Peki neden? Çünkü tek bir test çalıştırmasıyla çok fazla şey yanlış olabilir:
  - Bir başka işlem CPU zamanını artırabilir
  - Hard diskte bellek sayfaları değiştiriliyor olabilir
  - Network başka makinalar tarafından kullanılıyor olabilir

- Tek bir test çalıştırmak neredeyse değersizdir

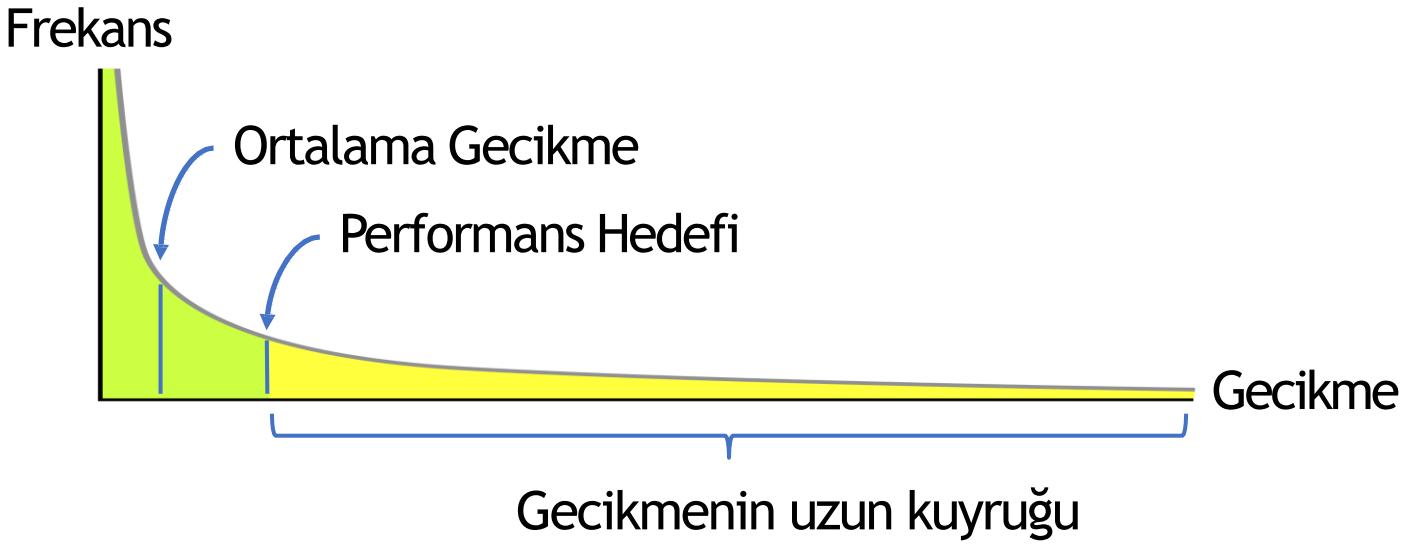
Performans testi bir bilimdir

- Test altındaki kod hariç diğer bütün değişkenleri elemelisiniz
  - Aynı donanım ayarları ile çalıştığınızdan emin olun
  - Birebir aynı kütüphane/OS/driver versiyonları ile çalışın
  - Test ettiğiniz program hariç bütün diğer processleri kapatın
  - Bütün zamanlanmış görevleri kaldırın (2 saatte bir çalışan anti-virüs gibi)
  - Çeşitli ısınma uygulamaları çalıştırarak hafızayı/cache'i doldurun

- Bütün bunları yaptıktan sonra bile değişkenlik var olmaya devam ediyor olabilir
  - İstatistiksel olarak anlamlı sonuçlar almak için çok fazla çalıştırıp sonuç alın

Uzun Gecikme Süresi

- Genellikle gecikme dağılımınız aşağıdaki gibi olacaktır



- Genelde «uzun kuyruk» kısmı ortalama gecikmeden daha önemlidir
  - Bunlar performans hedefinden şaşmış tepki süreleridir
- Sadece ortalamayı doğru bir şekilde ölçmek için değil, aynı zamanda "uzun kuyruğun" uzunluğunu ve yüksekliğini tespit etmek için birçok kez test etmek gereklidir.

Tepki süresini test etmek için olay çeşitleri

- Hesaplama zamanı
- Karakterin ekranda görünme zamanı
- Görüntünün görünme zamanı
- İndirme zamanı
- Sunucu cevap zamanı
- Sayfa yüklenme zamanı

Kaba tepki zamanı performans hedefleri

< 0.1 S : Sistemin anlık olduğuun hissetmek için gerekli tepki süresi

< 1 S : Akışın kesintiye uğramaması için gerekli tepki süresi

< 10 S : Kullanıcının uygulamaya odaklanması için gereken tepki süresi

- “Usability Engineering” by Jakob Nielsen, 1993 kitabından alınmıştır

Bu şeyler o zamandan bu zamana değişmemiştir!

Zaman Ölçüm Araçları

- Unix'te time komutu
  - time java Foo
  - time curl http://www.example.com
  - time ls
- Windows PowerShell:
  - Measure-Command { ls }

```
-bash$ time curl http://www.example.com
<!doctype html>
<html>
...
</html>

real    0m0.021s
user    0m0.002s
sys     0m0.004s
```

Bu tepki süresidir

Bunu daha sonra konuşacağız

Kullanılabilirlik Testi

- Kullanılabilirlik genellikle çalışma süresi olarak adlandırılır
  - Sisteme kullanıcı tarafından erişilebilen zamanın yüzde kaçı?

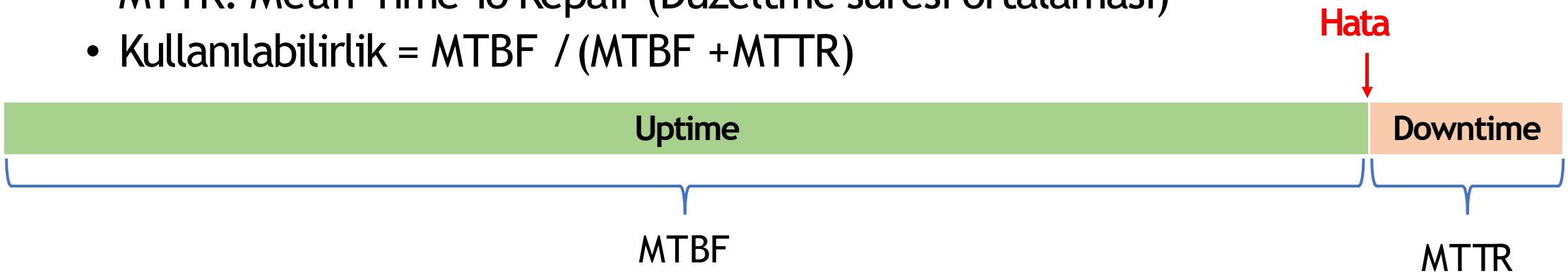
- Genellikle bir SLA (service-levelagreement)'da garanti edilir
  - "Ben bir web host'um ve belirli bir ayda kullanıcıların %99'unun erişebileceğini garanti ederim."

Dokuzlar…

- Çalışma süresi genellikle 9'lar ile ifade edilir (3 dokuz, 5 dokuz gibi)
- 9 sayısı ne kadar süre kullanılabilir olduğunu söyler
  - 1 dokuz: 90% kullanılabilirlik (Yılda 36.5 gün kapalı)
  - 2 dokuz : 99% kullanılabilirlik (Yılda 3.65 gün kapalı)
  - 3 dokuz : 99.9% kullanılabilirlik (Yılda 8.76 saat kapalı)
  - 4 dokuz : 99.99% kullanılabilirlik (Yılda 52.56 dakika kapalı)
  - 5 dokuz : 99.999% kullanılabilirlik (Yılda 5.26 dakika kapalı)
  - 6 dokuz : 99.9999% kullanılabilirlik (Yılda 31.5 saniye kapalı)
  - 9 dokuz : 99.9999999% kullanılabilirlik (Yılda 31.5 ms kapalı)

Nasıl test edilir?

- Zorluk: Deploy etmeden önce birkaç yıl test etmek mümkün değil
- Sistem modelleme ve çalışma süresi tahmini en uygun yaklaşımdır
- Model Metriği
  - MTBF: Mean Time Between Failures (Hatalar arası ortalama zaman)
  - MTTR: Mean Time To Repair (Düzeltme süresi ortalaması)
  - Kullanılabilirlik = MTBF / (MTBF +MTTR)



MTTR ve MTBF Ölçümü

- MTTR ölçümü kolaydır
  - Ortalama makinanın yeniden başlama süresi
  - Bir diski değiştirmenin ortalama süresi

- MTBF ölçümü zordur
  - Sistemin ne kadar stres altında olduğuna bağlıdır
  - Kullanım senaryolarına bağlıdır
  - Farklı kullanım senaryoları için MTBF ölçümü
    → Bu senaryolar için ağırlıklandırılmış bir ortalama MTBF ölçülmelidir

Yük Testi ile MTBF Ölçümü

- Yük testi:
  - Yük altında sistem ne kadar süre hata vermeden çalışacak?
  - Yük eş zamanlı istekler ve kullanıcılar ile ifade edilir
- Yük Testi Çeşitleri:
  - Temel Test (Baseline) – Bir temel sağlamak için minimum miktarda kullanım
  - Islanma/Stabilite (Soak / Stability) Testi – Uzun süreli standart kullanım testi
  - Stres Testi – Kısa aralıklarla yüksek aktivite testi
- Test sonuçlarına ve geçmiş yük verilerine göre MTBF tahmini
  - Eğer 90% temel kullanım, 10% yüksek kullanım var ise, MTBF = Islanma Testi MTBF * 0.9 + Stres Testi MTBF * 0.1

MTBF yalnızca uygulamanız ile ilgili değildir

- Gerçek kullanılabilirliği belirlemek için ayrıca şunları da belirlemelisiniz:
  - Donanım arızası olasılığı
  - OS çökme olasılığı
  - Veri merkezi soğutma sistem hatası olasılığı
  - Planlı bakımlar
  - vs.

İşler hala ters gidebilir…

- Bütün bunlar doğru çalışsa da bir şeyler yanlış gidiyor olabilir

- Pek çok büyük servis sağlayıcıları bazı aylarda SLA'larını ihlal eder
  - Microsoft Azure ve Amazon Web Services de dahil olmak üzere
  - Genellikle para otomatik olarak iade edilir

Servis Odaklı Test Planı Geliştirme

- Kullanıcı bakış açısıyla düşün!
  - Bunun ortalama olarak ne kadar hızlı olmasını bekliyorum?
  - Tepki süresindeki yüksek varyans uygun mu?
  - Bunun ne sıklıkla kullanılabilir olmasını bekliyorum?

Acil durum planları düşün!

- Performans gereksinimleri karşılanmazsa ne olur?
- Karşılansa bile yüksek maliyet/kaynak tüketimine ihtiyaç duyarsa ne olur?
- Ya olmazsa?
- vs.

Verimlilik Odaklı Performans Göstergelerinin Test Edilmesi

Throughput / Resource Utilization

Verim/Kaynak Kullanımı

Neden verimlilik odaklı göstergeler kullanılır?

1. Servis odaklı göstergelerden daha ayrıntılıdır
   - Performans sorununun tam olarak nerede olduğunu saptamak daha kolaydır
2. Sorunun nasıl çözüleceğini belirlemek mümkündür:
   - Yazılım değişikliği (daha iyi algoritma, daha iyi veri yapısı vs.)
   - OS ayarlama(OS güncelleme, OS planlayıcı ayarlama, I/O driver ayarlama vs.)
   - Donanım değişikliği (donanım ölçekleme, donanım güncelleme vs.)

Throughput Testi

- Throughput
  - Verilen zaman diliminde sistemin işleyebileceği eylem sayısı

- Örnek:
  - Saniye başına paket (Router tarafından ele alınır)
  - Dakika başına sayfa (Sunucu tarafından ele alınır)
  - Eş zamanlı kullanıcı sayısı (Oyun sunucusu tarafından ele alınır)

Peki bunun servis odaklı testten farkı nedir?

1. Kullanıcılar througput'u bilmez ve umursamazlar
   - Kullanıcılar yalnızca QoS (Quality of Service) odaklıdır (Yüksek tepki süresi veya hizmette kesinti var mı?)
   - Ancak mühendisler ilgilenirler çünkü problemi teşhis etmek veya öngörmek için gereklidir (Sunucu en yüksek throughput'a yakınsa yeni bir makine eklemek istenebilir)

2. Daha ayrıntılı
   - Kullanıcı tarafından görülmeyen belirli bir sistem davranışını tanımlar (sunulan sayfa sayısı/saniye, SQL sorgu sayısı/saniye, DNS sorgusu/saniye vs.)
   - Her bir alt komponentin performansı ölçülebilir (web server, database server, router, vs.)

Throughput Ölçümü: Yük Testi

- Yük testi kullanılabilirliğin yanında throughput testi için de kullanılabilir

- Temel olarak sistemin ele alabileceği maksimum yükü belirlemek için kullanılır
  - Quality of Software'i düşürmeden (QoA)
  - Tepki süresi belirtilen thresholdu aşana kadar saniye başına eylem sayısı artırılır
  - Eylem/saniye sonucu sistemin throughput'udur.

Test Kullanımı

- *Öncelikle bir araca ihtiyacınız vardır*
  - Fanın sesinden CPU üzerinde ne kadar işlem yapıldığını söyleyemediğiniz sürece ☺

Araçlar

- Genel Amaçlı
  - Windows Sistemler– Görev yöneticisi, perfmon
  - OS X - Activity Monitor veya Instruments, top
  - Unix systems - top, iostat, sar, time

- Program-Specific Araçlar

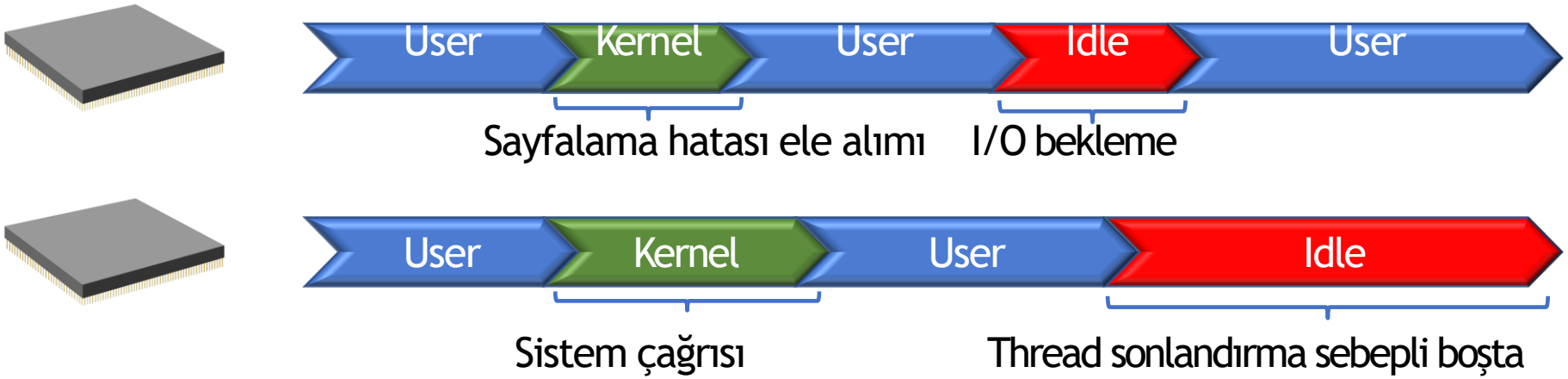
CPU Kullanım Ölçümü

- real time: "Gerçek" geçen süre (duvar saati zamanı)
- **user time**: Kullanıcı kodunun CPU üzerinde yürütüldüğü süre
- **system time**: Kernel (OS) kodunun CPU'da yürütüldüğü süre
- **total time**: user time + system time = **CPU kullanım süresi**

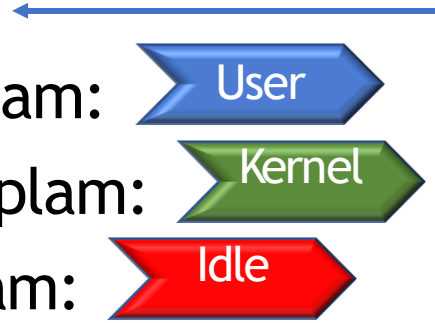
- real time ≠ total time
  - real time = total time + idle time
  - idle time: zaman uygulaması bazı işlemleri bekliyor (CPU çalıştırmadan) olabilir (I/O, senkronizasyon, interrupt'lar vs.)

Zaman Ölçüm Örneği

- 2 CPU üzerinde çalışan bir uygulama için örnek zaman dökümü



- Real time:
- User time: Toplam:
- Kernel time: Toplam:
- Idle time: Toplam:



- Şimdi bir önceki denklemi gözden geçirelim:
  Real time = Total time + Idle time
- Bu yalnızca tek CPU içindi. Çoklu CPUs için:
  Real time = Total time + Idle time / CPUs

“time” ile Zaman Ölçümü

- Unix için time komutu
  - time java Foo
  - time curl http://www.example.com
  - time ls
- Windows PowerShell için:
  - Measure-Command { ls }

```
-bash$ time curl http://www.example.com
<!doctype html>
<html>
...
</html>

real    0m0.021s
user    0m0.002s
sys     0m0.004s
```

- Real time = User time + Kernel time + Idle time / CPUs
- 0.021s = 0.002s + 0.004s + Idle time / 1 (single-threaded)
- Idle time = 0.015s → Zamanın büyük bir çoğunluğu sunucu cevabını beklemekle geçmiş

CPU Kullanımı için Performans Göstergeleri

- Servis Odaklı Test için
  - Kullanıcı yalnızca real time ile ilgilenir (response time)

- Verimlilik Odaklı Test için
  - Geliştiriciler verimlilik sorunlarını analiz etmeye yardımcı diğer zaman metriklerini de önemsemelidir
  - Yüksek oranda **usertime**?
    - Etkili veri yapıları kullanarak algoritma optimize edilmelidir
  - Yüksek oranda **kerneltime**?
    - İşletim sistemi sistem çağrılarını ve interruptları işlemek için çok fazla CPU zamanı kullanıyor
  - Yüksek oranda **idle time**?
    - Uygulama I/O için veya senkronizasyon için çok fazla bekliyor
    - CPU kullanımı problem yaratmıyor, verimlilik sorunu başka noktalardadır.

Genel Amaçlı Araçlar Tarafından İzlenebilir Kaynaklar

- CPU Kullanımı
- Thread'ler
- Fiziksel Memory
- Sanal Memory
- Disk I/O
- Network I/O

Diğer Kullanım Performans Göstergeleri

- Disk önbelleği yetersiz- yüksek disk kullanımının nedeni olabilir
- CPU önbelleği yetersiz- yüksek memory trafiğinin sebebi olabilir
- Dosya temizleme- Sık sık zorunlu dosya temizleme yüksek I/O kullanımına sebep olabilir
- Giden ağ paketlerinin atılması- network trafik problemi?

Genel amaçlı araçlar yalnızca genel bilgi verir

- Çok fazla memory kullanımı...
  - ...ancak hangi nesne/sınıf/veri tarafından?
- Çok fazla CPU kullanımı...
  - ...ancak hangi metot/fonksiyon tarafından?
- Çok fazla paket gönderimi...
  - ...ancak neden ve içerisinde ne bulunuyor?

Araçlar

- Genel Amaçlı
  - Windows Sistemler- Görev yöneticisi, perfmon
  - OS X - Activity Monitor veya Instruments, top
  - Unix systems - top, iostat, sar, time

- Program-Specific Araçlar

Program-Specific Araçlar

- Protokol Analizi
  - Ör., Wireshark veya tcpdump
  - Tam olarak hangi paketler gönderiliyor/alınıyor görülebilir
- Profiler
  - ör. JProfiler, VisualVM, gprof, ve daha bunlar gibi bir sürü profiler
  - Hangi nesne hafızayı tam olarak ne kadar kullanıyor takibi için
  - Hangi metotlar hangi süreyle ve ne sıklıkla çağırılıyor

Sonuç Olarak…

“Erken optimizasyon tüm kötülüklerin anasıdır” ☺
– Donald Knuth

- İlk olarak servis odaklı test gerçekleştir
  - Temel performans göstergeleri hedefe erişebilirse uğraşmaya gerek kalmaz
  - Aksi takdirde verimlilik odaklı testlerle detaya inilir

Servis Odaklı Testlerle Çözüm

- Varsayım: Rent-A-Cat uygulaması hangi kedinin kiralamaya uygun olduğunu gösteren bir Web API olsun ☺

1. Servis Odaklı Test
   - Response time: Sıralanmış-kedi-listeleme API'ı performans hedefi olan 100ms hedefine ulaşamamış olsun
2. Verimlilik Odaklı Test
   - Throughput testi: Max throughput = 100 istek/ saniye
     Tepki süresinde problem çok daha az bir istek ile gerçekleşir (10 istek / saniye)
   - Kullanım testi (istek başına):
     Network trafik kullanımı 1%
     Memory kullanımı 2%
     Ancak CPU 1 saniyeliğine %99'da sabitlendi
   - Teşhis: Throughput problemi değildir problem CPU işlemlerindeki gecikmelerden kaynaklanmaktadır

Servis Odaklı Testlerle Çözüm

3. Verimlilik odaklı test --- CPU Profiling
   - VisualVM sortCats() metodunun fazla zaman aldığını söylemektedir

4. Çözüm
   - Kediler insertion sıralama ile sıralanmaktadır, daha iyi algoritma kullanınız
   - Eğer bu da yetersiz kalırsa paralel sıralama algoritması kullanılabilir

Diğer Olası Sorunlar ve Çözümler

- Çöp toplamada (Garbage collection) çok fazla zaman harcanıyorsa
  - Profil memory'sini ve en fazla memory'i kaplayan nesneleri azaltmaya çalışın
  - Garbage collector'ı daha az sıklıkta ve daha verimli çalışacak şekilde ayarlayın
- Eğer network kullanımı problem ise
  - Network trafiğini azaltmak için JavaScript kodunu küçültün
  - Proxy sunucularda sıralama sonuçlarını cache'leyerek kullanın

Sürümler Boyunca Performansı Takip Edin

- Performans testleri regresyon paketlerinin bir parçası olmalı
- Tıpkı işlevsel kusurlarda olduğu gibi, bir performans sorununun ne zaman/nerede ortaya çıktığını tam olarak söyleyebilmelisiniz.
- Bu durum özellik veya geliştirmenin harcanacak efora değip değmeyeceğini belirleme konusunda yardımcı olur

Erciyes Üniversitesi
Bilgisayar Mühendisliği Bölümü

BZ 313 Yazılım Mühendisliği
15. Performans

Ders Sonu