İşletim Sistemleri Uygulama 11

Linux Bellek Yönetimi

Bilgisayar Mühendisliği

İstanbul Teknik Üniversitesi
34469 Maslak, İstanbul

May 4, 2011

Bugün

İşletim Sistemleri Uygulama 11

Süreçlerde Bellek Segmanlama

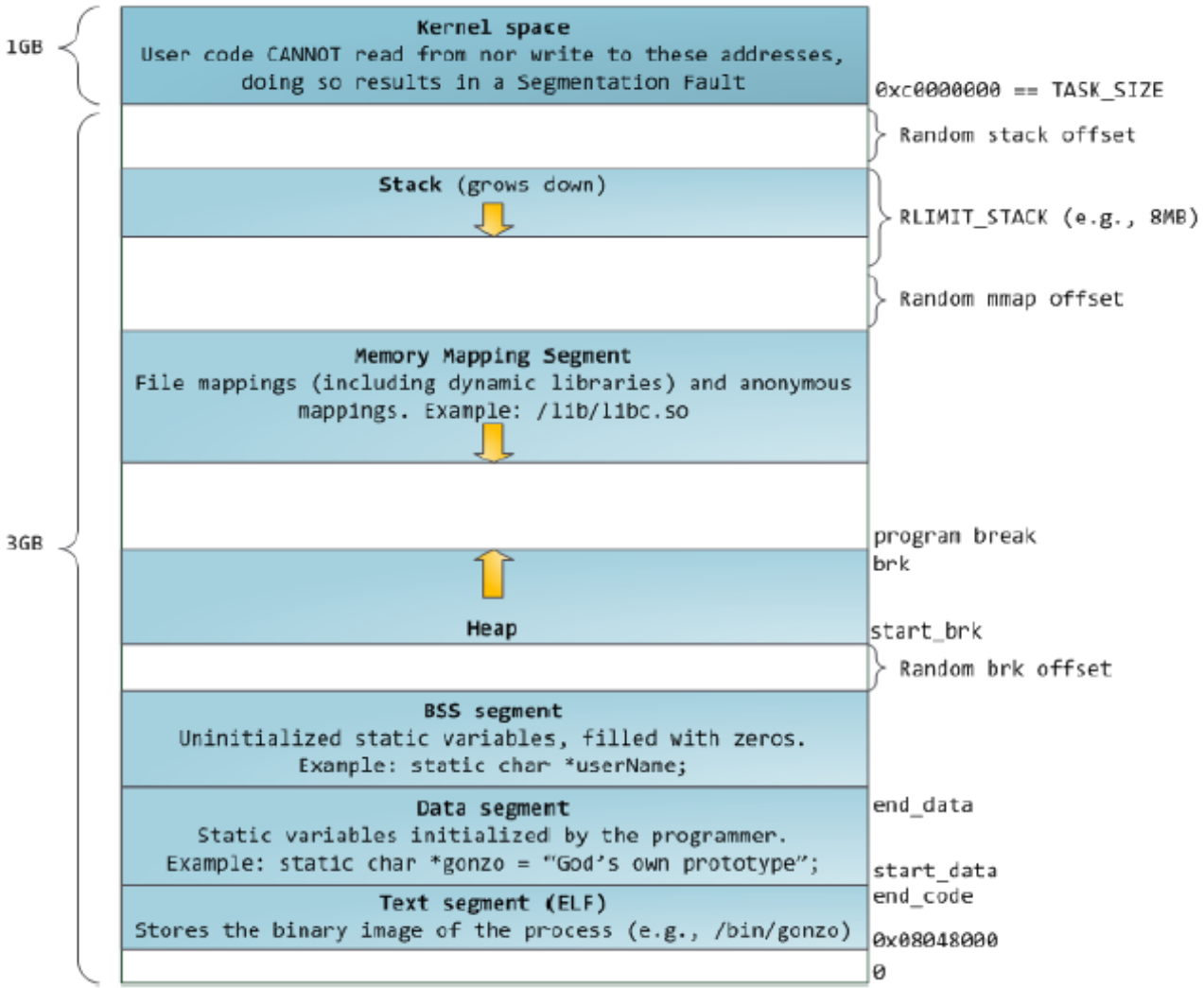
Süreçlerde Yer Değiştirme

Linux'de Sayfalama

Bellek Yönetimi Kısıtları

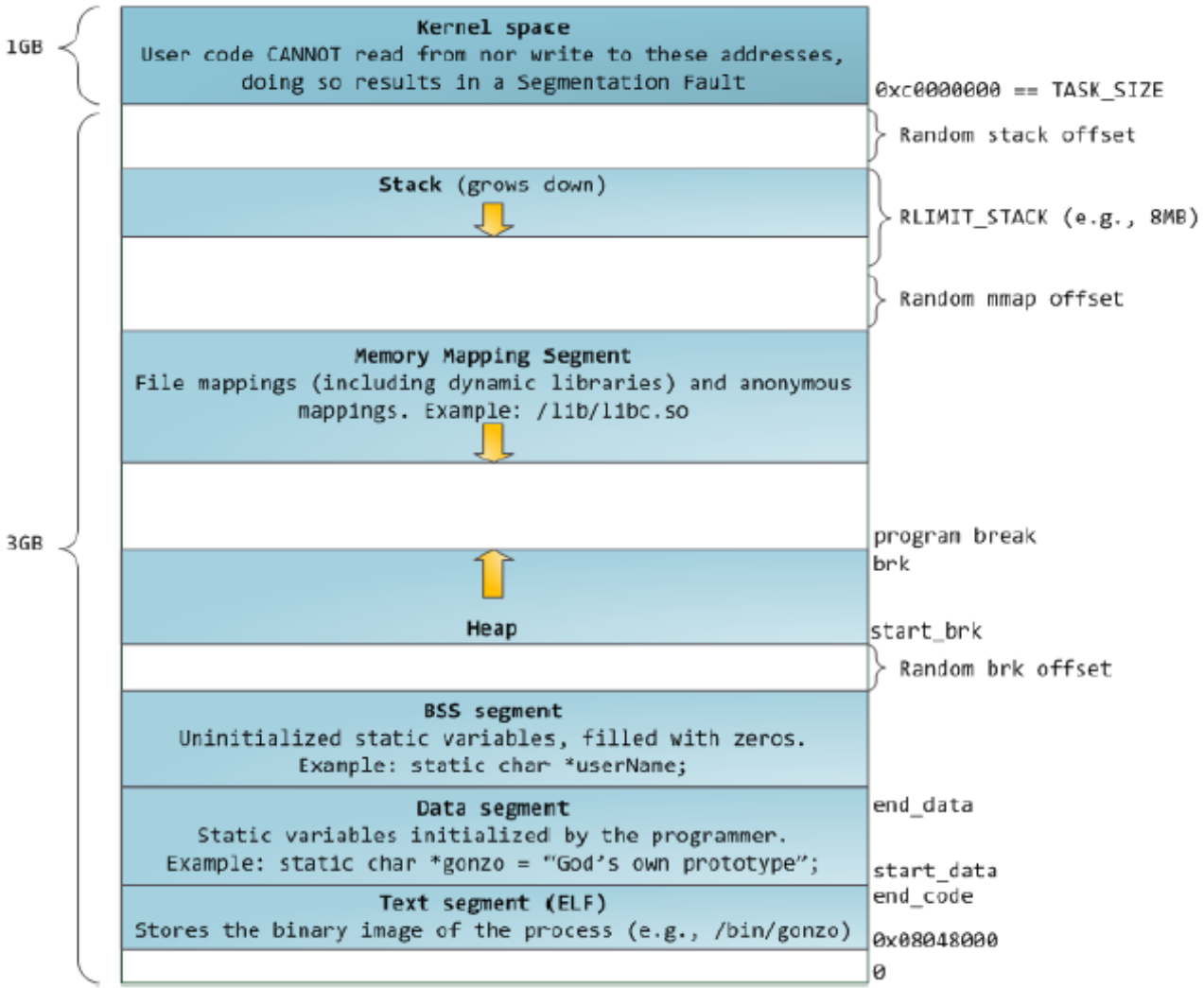
- 32 bitlik adresler kullanıldığında bir proses tarafından adreslenebilecek bellek miktarı sabit : $2^2 \times 2^{30}$=4GB.
- Ama sistemdeki ana bellek 4GB'dan çok daha büyük olabilir.
- Peki ya çekirdek tarafından kullanılan bellek? Tüm prosesler için erişilebilir olmalı.
- Bütün bu sorunları çözmek için her sürecin kendine ait 4GB'lik bir bellek bölgesi bulunur. Bu bölgedeki adresler belirli dönüşümlerden geçirilerek ana bellekle eşleştirildikleri gerçek adresler elde edilir. Bu 4GB'lik alanın adreslenmesi *sanal adresleme* olarak adlandırılır.
- Bu 4GB her süreç için 1GBlik çekirdek(windowsta 2GB) ve 3GBlik kullanıcı alanlarına bölünmüştür.
- Çekirdek alanı tüm süreçler için ortak fiziksel adreslere yönlenirken, kullanıcı alanı belirli alt alanlardan oluşur ve her süreç için farklı fiziksel adres kümesine eşlenir.

Süreçlere Ait Kullanıcı Bellek Alanları



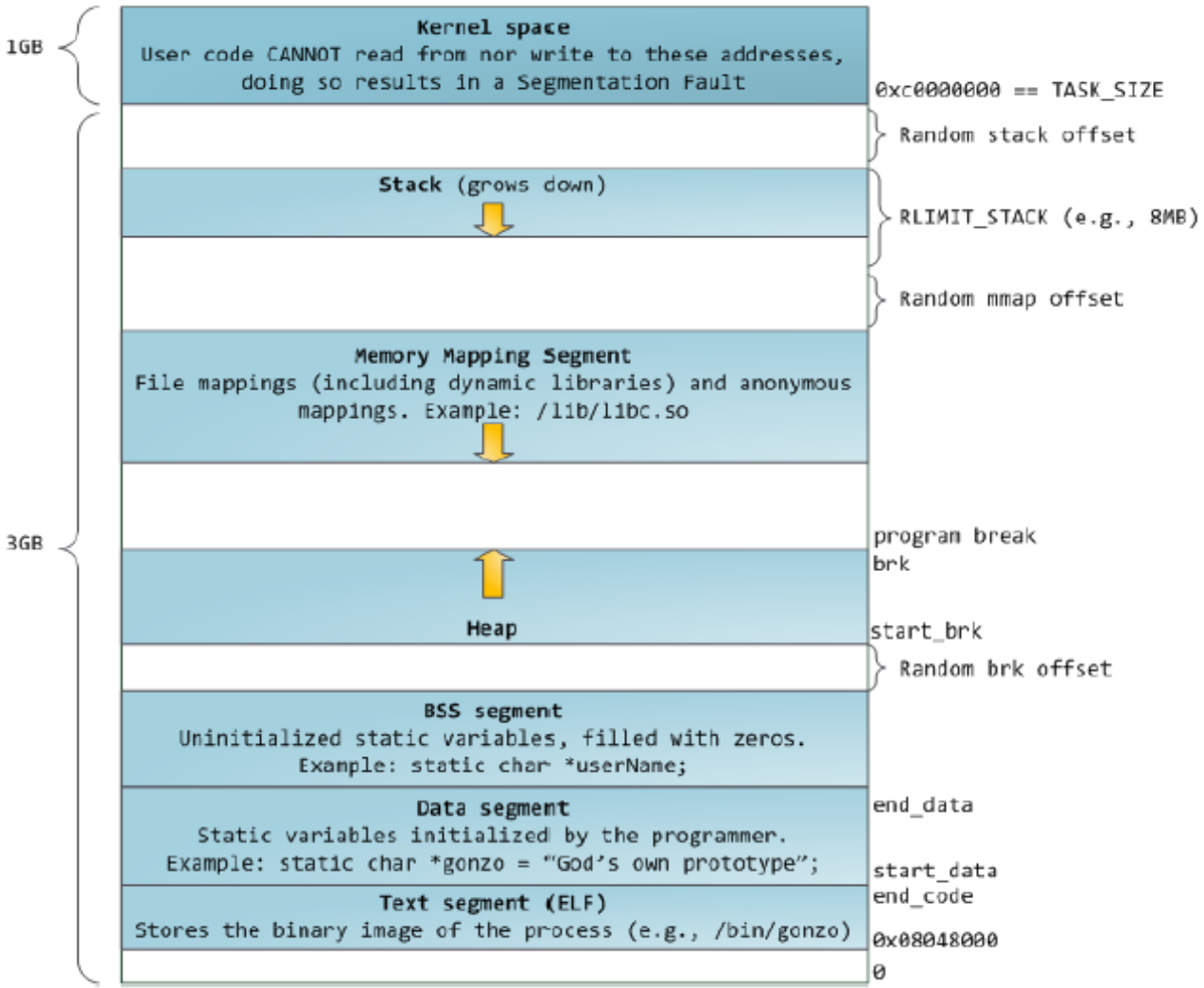
- En üstteki alan çekirdek alanı. Ayrılan belleğin son 1GBında yer alır.
- Kullanıcı alanının en sonunda yığın bulunur.
- Çağırılan her yordamla yığına yeni bir çerçeve eklenir.
- Yığın dinamik olarak büyür. Küçülmez!
- Sadece yığın işlemleri sonucu ana bellekle eşlenmemiş bölgelere erişilebilir. Aksi halde *Segmentation Fault* verir.

Süreçlere Ait Kullanıcı Bellek Alanları



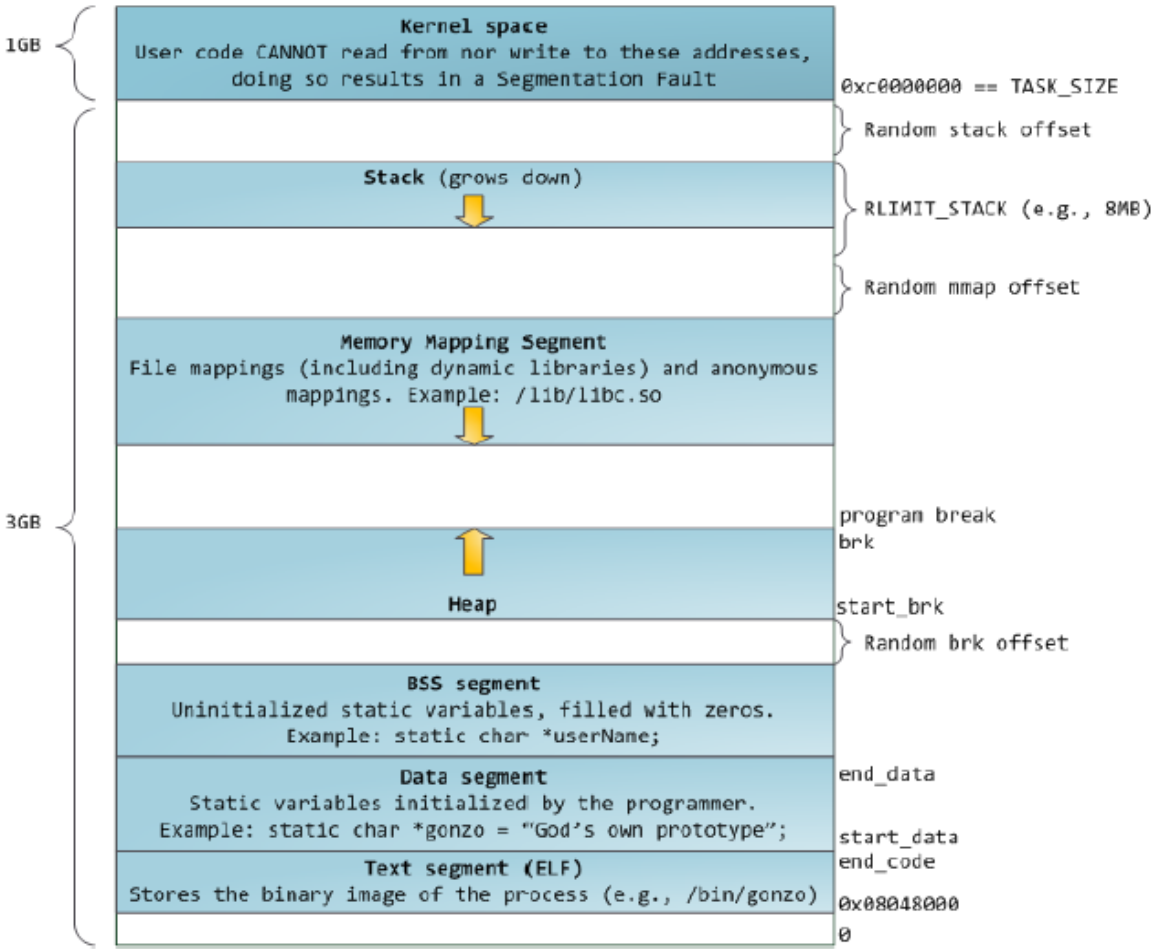
- Yığın alanından sonra bellek-eşlenmiş alan yer alır.
- Bu alan doğrudan ana bellekle eşleştirilmiştir ve süratli G/Ç işlemleri için buradan yer ayrılabilir.
- Genelde paylaşılan kütüphanelere bu bölge üzerinden erişilir.
- `mmap()` sistem çağrısı ile kullanıcı da bu bölgeden yer alabilir.

Süreçlere Ait Kullanıcı Bellek Alanları



- Bir sonraki bölge heap bölgesidir. Yordamlar tarafından kullanılan bellek bu alanda tutulur ve yığının aksine yordam çalışmasını bitirince yok olmaz.
- `malloc()` ve `new` gibi komutlarla bu kısımdan bellek alınabilir.
- Başlangıçta ayrılan heap'in yeterli olmaması durumunda `brk()` sistem çağrısıyla çekirdeğin alanı genişletmesi sağlanabilir.

Süreçlere Ait Kullanıcı Bellek Alanları



- BSS kısmı ilk değer verilmemiş sabit değişkenleri tutar. Bu kısım gerçek program dosyasıyla eşleştirilmez.
- Veri segmanında ilklendirilmiş sabit değişkenler tutulur. Bu kısım gerçek program dosyasıyla eşleştirilmiştir ama değer atamaları program dosyasını değiştirmez.
- Veri segmanında tutulan işaretçiler, metin segmanında tutulur.
- Metin segmanı program kodunun yaşadığı kısımdır. Doğrudan gerçek program dosyasıyla eşleştirilmiştir fakat programlar buraya yazamaz.
- `/proc/<pid>/maps` dosyası incelenerek ilgili bellek alanları görülebilir.

Yer Değiştirme Kavramı ve Yer Değiştirme Alanı

- Peki ya mevcut fiziksel bellek alanından daha fazla bellek alanına ihtiyacımız varsa?
- Sistemin daha az erişilen(süreç bazında ve genelde) bellek bölgelerini bir yerlerde saklayarak sabit diskten yeni verileri belleğe taşıması gerekli.
- Bu amaçla diskte bir bölme(partition) ya da bir dosya halinde yer değiştirme alanı(swap space) tutulur. Ama dikkat edilmelidir ki bu yer değiştirme alanına erişim belleğe oranla çok daha yavaştır.
- `swapon -s` komutu ile mevcut yer değiştirme alanları listelenebilir. Sistem birden fazla alan kullanabilir ve bunlar arasında öncelik atamaya izin verir.
- Sistemin yer değiştirme sıklığı da kullanıcı tarafından ayarlanabilir. Ör.`echo 50 > /proc/sys/vm/swappiness`. Bu değer kalıcı olarak ise `/etc/sysctl.conf` dosyası içerisindeki `vm.swappiness` parametresi ile oynanarak değiştirilebilir. Varsayılan değer 60'dır.
- `vmstat` komutu ile yer değiştirmeye ve birçok bellek özelleğine ilişkin bilgi edinilebilir.
- Yer değiştirme işleminden `kswapd` isimli çekirdek süreci sorumludur.
- `kswapd`, bellekte aranan sayfanın bulunamaması sonucu ortaya çıkan *Page Fault Exception* sonucu tetiklenir.

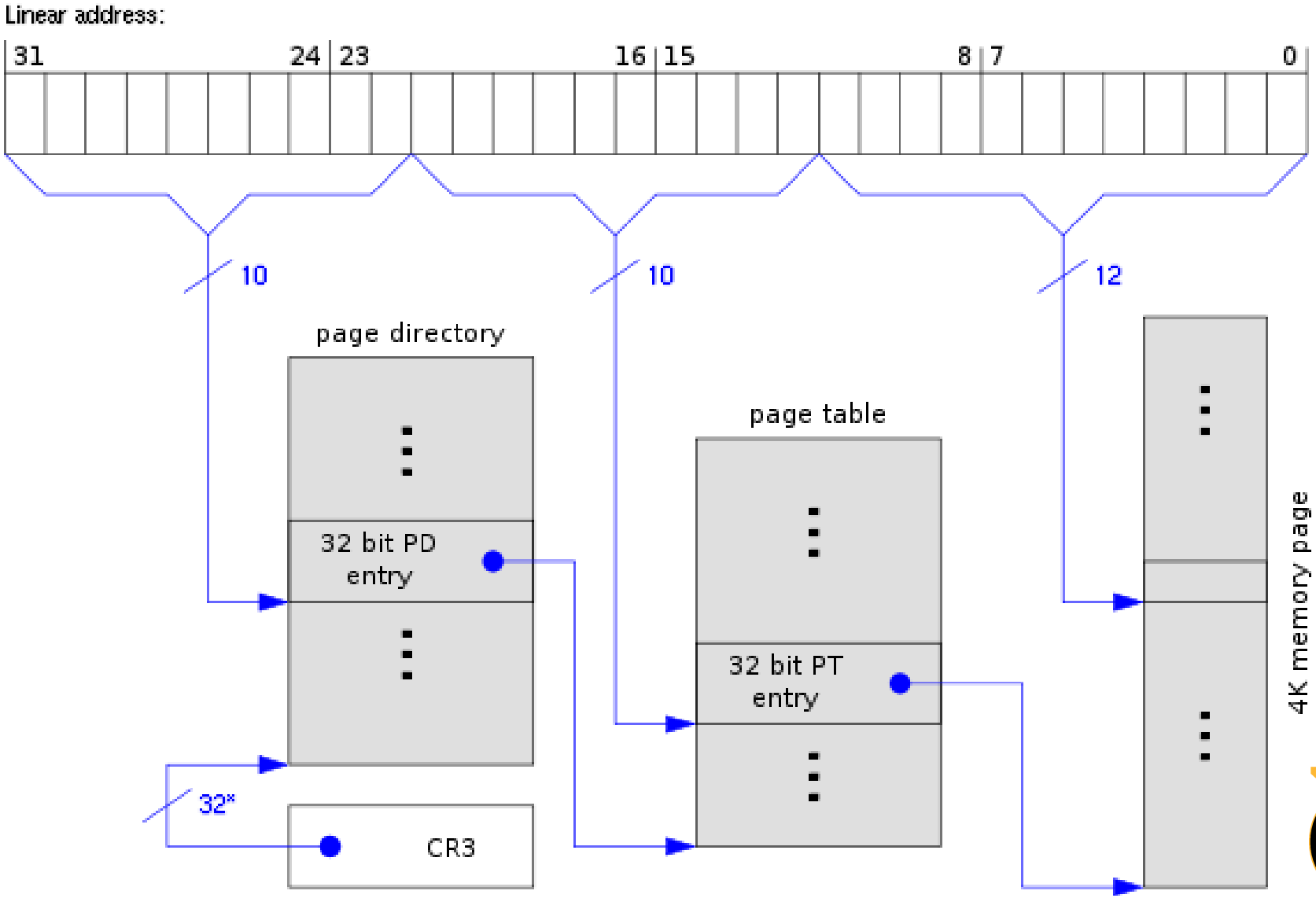
Linux'de sayfalama

- Bir uygulamanın tamamının belleğe alınması uygun olmayacaktır
- Uygulama bellek alanına sığmayabilir
- Uygulamanın tamamını belleğe taşımak yük getirir
- Bellekte kırıntılanma olabilir
- Disk ile bellek arasında yer değiştirmeler standart boyda sayfalara ayrılırsa bu sorunların tamamı çözülür
- Kod metni, veri ve yığın gerektikçe devingen olarak belleğe yerleşir
- 2 numaralı süreç, yani "page daemon" tarafından gerçekleştirilir

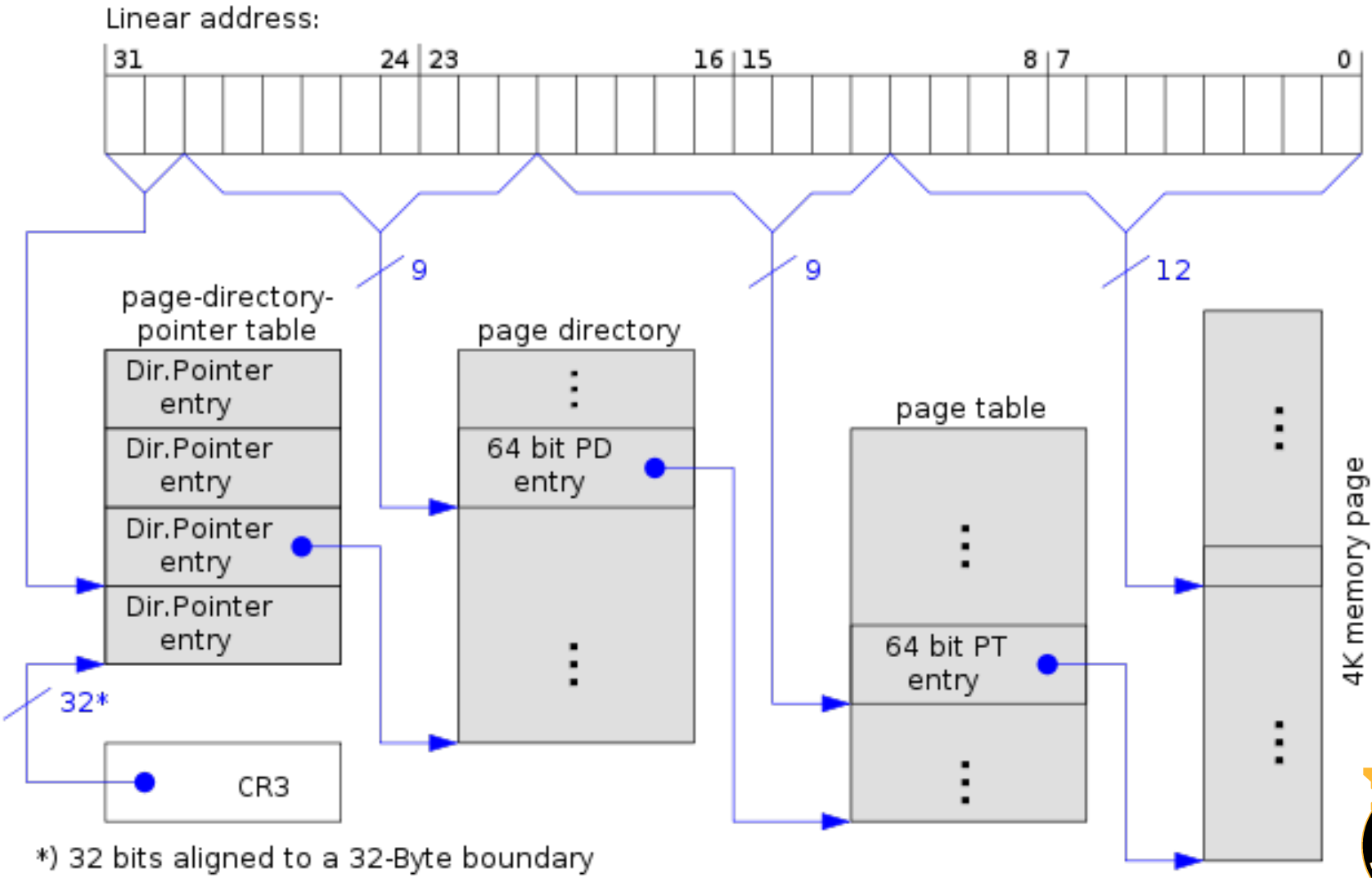
Fiziksel Adres Genişletme(PAE-Physical Address Extension)

- 32 bit adresleme ile en fazla 4GB bellek kullanılabilir
- Uygulama 4GB'a sığmayabilir
- Bu durumda işletim sisteminin de desteğiyle adres uzayı genişletilebilir

Linux'de bir adresin yapısı - Genişletme yokken



Linux'de bir adresin yapısı - Genişletme varken



Buddy bellek ayırma algoritması

- İşleyişi
  - Ayrılabilecek alanların en genişi ve en küçüğü seçilir
  - Bellek, 2'nin sıralı kuvvetleri olan genişlikte alanlara ayrılır
  - Bellekte alan gerektiğinde, ayrılacak alan 2'nin kendinden büyük en yakın kuvvetini seçerek o büyüklükteki alana yerleşir
  - Bellekte belirlenden büyük alan varsa, alt alanlara parçalanıp atanır
  - Bellek alanı boşaltılacaksa, alan boşaltılır ve buna komşu olan boş alan varsa özyinelemeli birleştirilerek oluşturulabilecek en büyük alan oluşturulur
  - Sözde kodu görmek için `http://en.wikipedia.org/wiki/Buddy_memory_allocation` bağına bakabilirsiniz.
- Özellikleri
  - Basit ancak etkin bir yöntem
  - Dışsal kırıntılanmayı azaltır
  - Algoritmanın yapısı gereği içsel kırıntılanma riski vardır
  - Yürütülürken fazla kaynak gerektirmez
  - Belleği toparlamak (compaction) kolaydır

Buddy bellek ayırma algoritması

		64K	64K	64K	64K	64K	64K	64K	64K
Başta	$t = 0$	1024K							
A 34K ister	$t = 1$	A, 64K	64K	128K		256K			
B 66K ister	$t = 2$	A, 64K	64K	B, 128K		256K			
C 45K ister	$t = 3$	A, 64K	C, 64K	B, 128K		256K			
D 92K ister	$t = 4$	A, 64K	C, 64K	B, 128K		D, 128K		128K	
C bırakır	$t = 5$	A, 64K	64K	B, 128K		D, 128K		128K	
A bırakır	$t = 6$	128K		B, 128K		D, 128K		128K	
B bırakır	$t = 7$	256K				D, 128K		128K	
D bırakır	$t = 8$	1024K							

Sayfa değiştirme algoritmaları - Tek süreç işlerken

"Referenced" biti olmayan mimarileri konu dışında bırakırsak:

- RAND –Rastgele bir sayfa seçilir
- FIFO –Kuyruğun başındaki sayfa seçilir, "Belady's anomaly" riski var
  3 2 1 0 3 2 4 3 2 1 0 4 dizisi 3 sayfalı sistemde 9 hata verirken 4 sayfalı sistemde 10 hata verecektir
- Second chance –İyileştirilmiş FIFO, kuyruğun başındaki sayfanın "referenced" biti 1 ise sıfırlanıp kuyruğa yeniden eklenir, erişilmemiş sayfa bulunduğunda seçilir
- Clock –İyileştirilmiş Second chance, kuyruk yerine çevrel liste kullanılır
- OPT (Belady's min) –En iyi teorik sonuç, sıradan bilgisayarlarda gerçeklenemez
- LRU (Least recently used) –OPT'ye yakın sonuç, genelde pahalı bir yöntem.
  Donanım desteği yokksa bağlı listeyle gerçeklenir, çok pahalı
  Donanım desteği varsa her sayfa en son kullanıldığı zamanı(?) tutan bir saklayıcıya sahip olur, yine de İS her seferinde tüm sayfaları taramak zorunda kalır
- NRU (Not recently used) –İS belli aralıklarla sayfaların "referenced" bitini sıfırlar, sıfırlama yakın zamanda olduysa RAND gibi davranır
- NFU (Not frequently used, Sampled LRU, SLRU) –Her sayfa için İS'te bir değişken tutulur, her erişimde değeri artırılır, en düşük değerli sayfa seçilir
- Aging –Her sayfa için İS'te bir değişken tutulur, her saat işaretinde 1 ya da 0 yazılıp sola kaydırılır, değişkeninde en az bir olan sayfa seçilir

"Dirty" biti kaldırılan sayfanın diske yazılıp yazılmayacağını belirtir

Sayfa değiştirme algoritmaları - Çok süreç işlerken

- Çok süreç işlerken LRU işlemez, süreçler çalışırken çok zaman geçebilir
- Bellek isteyen süreç ile G/Ç isteyen süreç dengesiz kullanılır
- Fixed allocation –Her sürece belli sayıda sayfa alanı ayrılır
- PFF (Page fault frequency) –Fixed allocation iyileştirmesi, İS her sürecin sayfa ıskasını sayar ve devingen olarak süreçlere sayfa alanı ayırır
- WS (Working set) –Sürecin kullandığı sayfalar kümesi topluca bellekten kaldırılır ya da belleğe alınır
- WSClock
  - Clock ve NFU yaklaşımlarını birlikte kullanır ancak kümeler üzerinde çalışır
  - Her küme için en son erişildiği zamanı gösteren bir değişken tutulur
  - En son erişilen zaman τ 'dan büyükse ve kümenin "referenced" biti sıfırsa sayfa seçilir
  - "Referenced" biti birse kümeye yeni şans verilir
  - En son erişilen zaman τ 'dan küçükse kümeye yeni şans verilir
  - En son erişilen zaman τ 'dan büyükse ancak süreç etkin değilse kümeye yeni şans verilebilir