18

Disk Belleği: Giriş

Bazen, herhangi bir alan yönetimi sorununu çözerken işletim sisteminin iki yaklaşımdan birini aldığı söylenir. İlk yaklaşım, sanal bellekte **segmentasyonda (segmentation)** gördüğümüz gibi, nesneleri *değişken-boyutlu* parçalara bölmektir. Ne yazık ki, bu çözümün kendine özgü zorlukları vardır. Özellikle, bir alanı farklı büyüklükteki parçalara bölerken, alanın kendisi **parçalanabilir (fragmented)** ve bu nedenle zaman içinde tahsis daha zor hale gelir.

Bu nedenle, ikinci yaklaşımı göz önünde bulundurmaya değer olabilir: alanı sabit boyutlu parçalara bölmek. Sanal bellekte buna fikir **sayfalama (paging)** diyoruz ve bu, eski ve önemli bir sisteme, Atlas'a [KE+62, L78] kadar uzanıyor. Bir işlemin adres alanını değişken boyutlu mantıksal parçalara (örneğin, kod, yığın, yığın) bölmek yerine, onu her birine **sayfa (page)** adını verdiğimiz sabit boyutlu birimlere ayırırız.
Buna bağlı olarak, fiziksel belleği **sayfa çerçeveleri (page frames)** adı verilen sabit boyutlu yuvalar dizisi olarak görüyoruz; bu çerçevelerin her biri tek bir sanal bellek sayfası içerebilir. Bizim meydan okumamız:

DÖNÜM NOKTASI:
SAYFALARLA BELLEĞİ NASIL SANALLAŞTIRILIR

Bölümleme sorunlarından kaçınmak için belleği sayfalarla nasıl sanallaştırabiliriz? Temel teknikler nelerdir? Bu teknikleri minimum yer ve zaman giderleri ile nasıl iyi çalışır hale getirebiliriz?

18.1 Basit Bir Örnek ve Genel Bakış

Bu yaklaşımı daha net hale getirmeye yardımcı olmak için basit bir örnekle açıklayalım. Şekil 18.1 (sayfa 2), dört adet 16 baytlık sayfa (sanal sayfalar 0, 1, 2 ve 3) ile toplam boyutu yalnızca 64 bayt olan küçük bir adres alanı örneği sunar. Gerçek adres alanları çok daha büyüktür, tabii ki genellikle 32 bit ve dolayısıyla 4 GB adres alanı, hatta 64 bit[1]; kitapta, sindirilmelerini kolaylaştırmak için genellikle küçük örnekler kullanacağız.

[1]A 64-bit adres alanını hayal etmek zor, inanılmaz derecede büyük. Bir benzetme yardımcı olabilir: 32 bitlik bir adres alanını bir tenis kortu büyüklüğünde düşünürseniz, 64 bitlik bir adres alanı yaklaşık Avrupa(!) boyutundadır.

Şekil 18.1: **64 baytlık Basit Bir Adres Alanı**

Şekil 18.2'de gösterildiği gibi, fiziksel bellek de bir dizi sabit-boyutlu yuvadan oluşur, bu durumda sekiz sayfa çerçevesi (128 baytlık bir fiziksel bellek oluşturur, yine gülünç derecede küçüktür). Diyagramda görebileceğiniz gibi, sanal adres alanının sayfaları fiziksel bellek boyunca farklı konumlara yerleştirilmiştir; diyagram ayrıca işletim sisteminin fiziksel belleğin bir kısmını kendisi için kullandığını gösterir.

Göreceğimiz gibi sayfalama, önceki yaklaşımlarımıza göre bir takım avantajlara sahiptir. Muhtemelen en önemli gelişme *esneklik* olacaktır: tamamen geliştirilmiş bir sayfalama yaklaşımıyla sistem, bir işlemin adres alanını nasıl kullandığından bağımsız olarak, bir adres alanının soyutlanmasını etkili bir şekilde destekleyebilecektir; örneğin, yığının ve yığının büyüme yönü ve nasıl kullanıldığı hakkında varsayımlarda bulunmayacağız.

Şekil 18.2: **128-Byte Fiziksel Bellekte 64-Byte Adres Alanı**

Diğer bir avantaj, sayfalamanın sağladığı boş alan yönetiminin basitliğidir. Örneğin, işletim sistemi 64 baytlık küçük adres alanımızı sekiz sayfalık fiziksel belleğimize yerleştirmek istediğinde, yalnızca dört boş sayfa bulur; belki işletim sistemi bunun için tüm boş sayfaların **ücretsiz bir listesini (free list)** tutar ve bu listeden ilk dört boş sayfayı alır. Örnekte, işletim sistemi adres alanının (AS) sanal 0. sayfasını fiziksel çerçeve 3'e, AS'nin sanal 1. sayfasını fiziksel çerçeve 7'ye, sayfa 2'yi çerçeve 5'e ve sayfa 3'ü çerçeve 2'ye yerleştirmiştir. Sayfa çerçeveleri 1 , 4 ve 6 şu anda ücretsiz.

Adres alanının her bir sanal sayfasının fiziksel bellekte nereye yerleştirildiğini kaydetmek için, işletim sistemi genellikle **sayfa tablosu (page table**) olarak bilinen işlem başına bir veri yapısı tutar. Sayfa tablosunun ana rolü, adres uzayındaki sanal sayfaların her biri için **adres çevirilerini (address translations)** depolamak ve böylece her sayfanın fiziksel bellekte nerede olduğunu bilmemizi sağlamaktır. Basit örneğimiz için (Şekil 18.2, sayfa 2), sayfa tablosunda şu dört giriş olacaktır: (Sanal Sayfa 0 → Fiziksel Çerçeve 3), (VP 1 → PF 7), (VP 2 → PF 5) ve (VP 3 → PF 2).

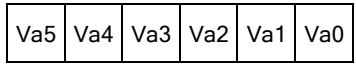
Bu sayfa tablosunun işlem başına bir veri yapısı olduğunu unutmamak önemlidir (tartıştığımız sayfa tablosu yapılarının çoğu işlem başına yapılardır; üzerinde duracağımız bir istisna **ters çevrilmiş sayfa tablosudur (inverted page table)**. Yukarıdaki örneğimizde başka bir işlem çalışacak olsaydı, sanal sayfaları açıkça farklı fiziksel sayfalarla eşleştiğinden (herhangi bir paylaşım modulo) OS'nin bunun için farklı bir sayfa tablosu yönetmesi gerekirdi.

Artık bir adres çevirisi örneği yapacak kadar bilgimiz var. Bu küçücük adres alanı (64 bayt) ile işlemin bir bellek erişimi gerçekleştirdiğini düşünelim:

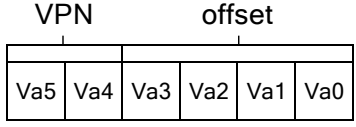
```
movl <virtual address>, %eax
```

Spesifik olarak, verilerin açık yüküne dikkat edelim. `<virtual address>` adresini kayıt `eax`'ına ekleyin (ve böylece daha önce gerçekleşmiş olması gereken talimat getirme işlemini göz ardı edin).

İşlemin oluşturduğu bu sanal adresi **çevirmek (translate)** için önce onu iki bileşene ayırmamız gerekir: **sanal sayfa numarası (virtual page number (VPN))** ve sayfa içindeki **ofset (offset)**. Bu örnek için, işlemin sanal adres alanı 64 bayt olduğundan, sanal adresimiz için toplam 6 bite ihtiyacımız var. (2^6 = 64). Böylece, sanal adresimiz aşağıdaki gibi kavramsallaştırılabilir:



Bu şemada, Va5 sanal adresin en yüksek dereceli bitidir ve Va0 en düşük dereceli bittir. Sayfa boyutunu (16 bayt) bildiğimiz için, sanal adresi aşağıdaki gibi daha da bölebiliriz:

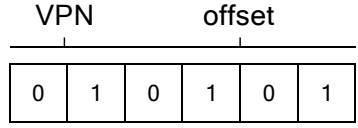


Sayfa boyutu, 64 baytlık bir adres alanında 16 bayttır; bu nedenle 4 sayfa seçebilmemiz gerekiyor ve adresin ilk 2 biti tam da bunu yapıyor. Böylece elimizde 2 bitlik bir sanal sayfa numarası (VPN) var. Kalan bitler bize sayfanın hangi baytıyla ilgilendiğimizi söyler, bu durumda 4 bit; buna ofset diyoruz.

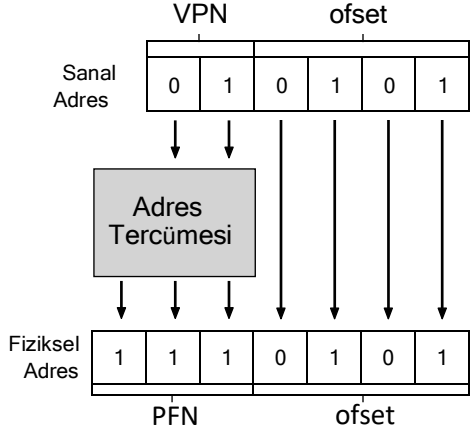
Bir işlem bir sanal adres oluşturduğunda, işletim sistemi ve donanımın bunu anlamlı bir fiziksel adrese çevirmek için birleşmesi gerekir. Örneğin, yukarıdaki yükün sanal adres 21'e olduğunu varsayalım:

```
movl 21, %eax
```

"21"i ikili forma çevirerek "010101" elde ederiz ve böylece bu sanal adresi inceleyebilir ve nasıl bir sanal sayfa numarasına (VPN) ayrıldığını ve ofsetini görebiliriz:



Böylece sanal adres "21", sanal sayfa "01"in (veya 1) 5. ("0101"inci) baytındadır. Sanal sayfa numaramızla artık sayfa tablomuzu indeksleyebilir ve sanal sayfa 1'in hangi fiziksel çerçeve içinde olduğunu bulabiliriz. Yukarıdaki sayfa tablosunda **fiziksel çerçeve numarası (physical frame number)(PFN)** (bazen **fiziksel sayfa numarası (physical page number)** veya **PPN** olarak da adlandırılır) 7'dir (ikili 111). Böylece VPN'i PFN ile değiştirerek bu sanal adresi çevirebilir ve ardından yükü fiziksel belleğe verebiliriz (Şekil 18.3).



Şekil 18.3: **Adres Çeviri Süreci**

Şekil 18.4: **Örnek: Çekirdek Fiziksel Belleğindeki Sayfa Tablosu**

Ofsetin aynı kaldığına dikkat edin (yani çevrilmez), çünkü ofset bize sayfada hangi baytı istediğimizi söyler. Nihai fiziksel adresimiz 1110101'dir (ondalık olarak 117) ve tam olarak yükümüzün veri getirmesini istediğimiz yerdir (Şekil 18.2, sayfa 2).

Bu temel genel bakışı göz önünde bulundurarak, artık sayfalama hakkında sahip olabileceğiniz birkaç temel soruyu sorabiliriz (ve umarız cevaplayabiliriz). Örneğin, bu sayfa tabloları nerede saklanıyor? Sayfa tablosunun tipik içerikleri nelerdir ve tablolar ne kadar büyük? Sayfalama sistemi (çok) yavaşlatıyor mu? Bu ve diğer aldatıcı sorular, en azından kısmen, aşağıdaki metinde yanıtlanmıştır. Okumaya devam etmek!

18.2 Sayfa Tabloları Nerede Saklanır?

Sayfa tabloları çok büyük olabilir, sayfa tablolarından çok daha büyük olabilir. küçük segment tablosu veya daha önce tartıştığımız taban/sınır çifti. Örneğin, 4 KB sayfaları olan tipik bir 32 bit adres alanı hayal edin. Bu sanal adres, 20 bitlik bir VPN ve 12 bitlik ofset olarak ikiye ayrılır (1 KB sayfa boyutu için 10 bitin gerekli olduğunu hatırlayın ve 4 KB'a ulaşmak için sadece iki tane daha ekleyin).

20 bitlik bir VPN, orada olduğu anlamına gelirişletim sisteminin her işlem için yönetmesi gereken 220 çeviridir (bu yaklaşık bir milyondur); fiziksel çeviriyi ve diğer yararlı şeyleri tutmak için **sayfa tablosu girişi (page table entry) (PTE)** başına 4 bayta ihtiyacımız olduğunu varsayarsak, her sayfa tablosu için gereken 4 MB'lık muazzam bir bellek elde ederiz! Bu oldukça büyük. Şimdi çalışan 100 işlem olduğunu hayal edin: bu, işletim sisteminin tüm bu adres çevirileri için 400 MB belleğe ihtiyaç duyacağı anlamına gelir! Modern çağda bile, nerede

KENARA: VERİ YAPISI — SAYFA TABLOSU

Modern bir işletim sisteminin bellek yönetimi alt sistemindeki en önemli veri yapılarından biri **sayfa tablosudur (page table)**. Genel olarak, bir sayfa tablosu **sanaldan fiziksele adres çevirilerini (virtual-to-physical address translations)** depolar, böylece sistemin bir adres alanındaki her sayfanın fiziksel bellekte gerçekte nerede bulunduğunu bilmesini sağlar. Her adres alanı bu tür çevirileri gerektirdiğinden, genellikle sistemde işlem başına bir sayfa tablosu bulunur. Sayfa tablosunun tam yapısı ya donanım tarafından belirlenir (eski sistemler) ya da işletim sistemi (modern sistemler) tarafından daha esnek bir şekilde yönetilebilir.

makinelerin gigabaytlarca belleği var, büyük bir kısmını sadece çeviriler için kullanmak biraz çılgınca görünüyor, değil mi? Ve böyle bir sayfa tablosunun 64 bitlik bir adres alanı için ne kadar büyük olacağını düşünmeyeceğiz bile; bu çok ürkütücü olur ve belki de seni tamamen korkutur.

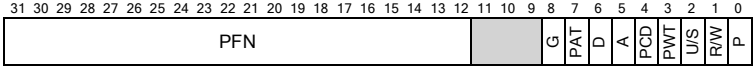
Sayfa tabloları çok büyük olduğu için, şu anda çalışan sürecin sayfa tablosunu depolamak için MMU'da herhangi bir özel çip üstü donanım tutmuyoruz. Bunun yerine, her işlem için sayfa tablosunu bellekte bir yerde saklarız. Şimdilik sayfa tablolarının işletim sisteminin yönettiği fiziksel bellekte yaşadığını varsayalım; daha sonra işletim sistemi belleğinin çoğunun kendisinin sanallaştırılabileceğini ve böylece sayfa tablolarının işletim sistemi sanal belleğinde saklanabileceğini (ve hatta diske değiştirilebileceğini) göreceğiz, ancak bu şu anda çok kafa karıştırıcı, bu yüzden inceleyeceğiz. boşver Şekil 18.4'te (sayfa 5), işletim sistemi belleğindeki bir sayfa tablosunun resmi gösterilmektedir; oradaki küçük çeviri setini görüyor musunuz?

18.3 Sayfa Tablosunda Gerçekte Neler Var?

Sayfa tablosu organizasyonu hakkında biraz konuşalım. Sayfa tablosu, yalnızca sanal adresleri (veya gerçekten sanal sayfa numaralarını) fiziksel adreslere (fiziksel çerçeve numaraları) eşlemek için kullanılan bir veri yapısıdır. Böylece, herhangi bir veri yapısı çalışabilir. En basit biçim, yalnızca bir dizi olan **doğrusal sayfa tablosu (linear page table)** olarak adlandırılır. İşletim sistemi, diziyi sanal sayfa numarasına (VPN) göre dizine ekler ve istenen fiziksel çerçeve numarasını (PFN) bulmak için bu dizinde sayfa tablosu girişine (PTE) bakar. Şimdilik ,bu basit lineer yapıyı kabul edeceğiz; sonraki bölümlerde, sayfalamayla ilgili bazı sorunları çözmeye yardımcı olması için daha gelişmiş veri yapılarından yararlanacağız.

Her bir PTE'nin içeriğine gelince, orada bir düzeyde anlaşılmaya değer birkaç farklı parçamız var. Belirli bir çevirinin geçerli olup olmadığını belirtmek için **geçerli bir bit (valid bit)** ortaktır; örneğin, bir program çalışmaya başladığında, adres alanının bir ucunda kod ve öbek, diğer ucunda yığın olacaktır. Aradaki kullanılmayan tüm alan geçersiz olarak işaretlenecek ve işlem bu tür bir belleğe erişmeye çalışırsa, işletim sistemine muhtemelen işlemi sonlandıracak bir tuzak oluşturacaktır. Bu nedenle, geçerli bit, seyrek bir adres alanını desteklemek için çok önemlidir; adres alanındaki kullanılmayan tüm

sayfaları **geçersiz (invalid)** olarak işaretleyerek, bu sayfalar için fiziksel çerçeve ayırma ihtiyacını ortadan kaldırır ve böylece büyük miktarda bellek tasarrufu sağlarız.



Şekil 18.5: **Bir x86 Sayfa Tablosu Girişi (PTE)**

Ayrıca, sayfanın okunup okunamayacağını, yazılacağını veya yürütülebileceğini gösteren **koruma bitlerimiz (protection bits)** olabilir. Yine bu bitlerin izin vermediği bir sayfaya erişim, işletim sistemine tuzak oluşturacaktır.

Birkaç önemli nokta daha var ama şimdilik çok fazla konuşmayacağız. **Mevcut bir bit (present bit)**, bu sayfanın fiziksel bellekte mi yoksa diskte mi olduğunu gösterir (yani, **değiştirilmiştir (swapped out)**). Fiziksel bellekten daha büyük olan adres alanlarını desteklemek için adres alanının parçalarının diske nasıl değiştirileceğini incelediğimizde bu mekanizmayı daha iyi anlayacağız; **takas (swap)**, işletim sisteminin nadiren kullanılan sayfaları diske taşıyarak fiziksel belleği boşaltmasına olanak tanır. Sayfanın belleğe alındığından beri değiştirilip değiştirilmediğini gösteren **kirli bir bit (dirty bit)** de yaygındır.

Bir sayfaya erişilip erişilmediğini izlemek için bazen bir **referans biti (reference bit)** (**erişilen bit (accessed bit)** olarak da bilinir) kullanılır ve hangi sayfaların popüler olduğunu ve bu nedenle bellekte tutulması gerektiğini belirlemede yararlıdır; bu tür bilgiler, sonraki bölümlerde ayrıntılı olarak inceleyeceğimiz bir konu olan **sayfa değiştirme (page replacement)** sırasında kritik öneme sahiptir.

Şekil 18.5, x86 mimarisinden [I09] örnek bir sayfa tablosu girişini göstermektedir. Mevcut bir bit (P) içerir; bu sayfaya yazmaya izin verilip verilmediğini belirleyen bir okuma/yazma biti (R/W); kullanıcı modu işlemlerinin sayfaya erişip erişemeyeceğini belirleyen bir kullanıcı/denetleyici biti (U/S); bu sayfalar için donanım önbelleğinin nasıl çalıştığını belirleyen birkaç bit (PWT, PCD, PAT ve G); erişilen bir bit (A) ve bir kirli bit (D); ve son olarak, sayfa çerçeve numarasının (PFN) kendisi.

x86 sayfalama desteği hakkında daha fazla ayrıntı için Intel Mimarisi Kılavuzlarını [I09] okuyun. Ancak önceden uyarılmalıdır; Bunlar gibi kılavuzları okumak, oldukça bilgilendirici olsa da (ve işletim sisteminde bu tür sayfa tablolarını kullanmak için kod yazanlar için kesinlikle gerekli), ilk başta zor olabilir. Biraz sabır ve çokça arzu gereklidir.

KENARA: NEDEN GEÇERLİ BİT YOK?

Intel örneğinde, ayrı geçerli ve mevcut bitlerin olmadığını, sadece mevcut bir bitin (P) olduğunu fark edebilirsiniz. Bu bit ayarlanmışsa (P=1), sayfanın hem mevcut hem de geçerli olduğu anlamına gelir. Değilse (P=0), bu, sayfanın bellekte bulunmadığı (ancak geçerli olduğu) veya geçerli olmadığı anlamına gelir. P=0 olan bir sayfaya erişim, işletim sistemine yönelik bir tuzağı tetikleyecektir; işletim sistemi daha sonra sayfanın geçerli olup olmadığını (ve bu nedenle belki de geri değiştirilmesi gerektiğini) veya olmadığını (ve dolayısıyla program belleğe yasa dışı bir şekilde erişmeye çalışıyor) belirlemek için sakladığı ek yapıları kullanmalıdır. Bu tür sağduyululuk, genellikle işletim sisteminin tam bir hizmet oluşturabileceği minimum özellik kümesini sağlayan donanımda yaygındır.

18.4 Disk Belleği: Ayrıca Çok Yavaş

Hafızadaki sayfa tabloları ile bunların çok büyük olabileceğini zaten biliyoruz. Görünüşe göre, işleri de yavaşlatabilirler. Örneğin, basit talimatımızı alın:

```
movl 21, %eax
```

Yine, adres 21'e yapılan açık referansı inceleyelim ve talimat getirme konusunda endişelenmeyelim. Bu örnekte, çeviriyi bizim yerimize donanımın yaptığını varsayacağız. İstenen veriyi getirmek için sistem önce sanal adresi (21) doğru fiziksel adrese (117) **çevirmelidir (translate)**. Bu nedenle, 117 adresinden verileri getirmeden önce, sistem önce işlemin sayfa tablosundan uygun sayfa tablosu girişini getirmeli, çeviriyi gerçekleştirmeli ve ardından verileri fiziksel bellekten yüklemelidir.

Bunu yapmak için, donanımın o anda çalışmakta olan işlem için sayfa tablosunun nerede olduğunu bilmesi gerekir. Şimdilik tek bir **sayfa tablosu temel kaydının (page-table base register)**, sayfa tablosunun başlangıç konumunun fiziksel adresini içerdiğini varsayalım. İstenen PTE'nin yerini bulmak için donanım böylece aşağıdaki işlevleri yerine getirecektir:

```
VPN     = (VirtualAddress & VPN_MASK) >> SHIFT
PTEAddr = PageTableBaseRegister + (VPN * sizeof(PTE))
```

Örneğimizde `VPN_MASK`, tam sanal adresten VPN bitlerini seçen 0x30 (onaltılık 30 veya ikili 110000) olarak ayarlanır; `SHIFT`, doğru tamsayı sanal sayfa numarasını oluşturmak için VPN bitlerini aşağı taşıyacak şekilde 4'e (ofsetteki bit sayısı) ayarlanır. Örneğin, sanal adres 21 (010101) ile ve maskeleme bu değeri 010000'e dönüştürür; kaydırma onu 01'e veya isteğe göre sanal sayfa 1'e dönüştürür. Daha sonra bu değeri, sayfa tablosu temel kaydı tarafından işaret edilen PTE dizisine bir dizin olarak kullanırız.

Bu fiziksel adres bilindiğinde, donanım PTE'yi bellekten alabilir, PFN'yi çıkarabilir ve istenen fiziksel adresi oluşturmak için sanal adresten ofset ile birleştirebilir. Spesifik olarak, PFN'nin `SHIFT` tarafından sola kaydırıldığını ve ardından son adresi aşağıdaki gibi oluşturmak için ofset ile bit düzeyinde OR'lendiğini düşünebilirsiniz:

```
offset   = VirtualAddress & OFFSET_MASK
PhysAddr = (PFN << SHIFT) | offset
```

Son olarak, donanım istenen verileri bellekten alabilir ve kayıt eax'ine koyabilir. Program artık bellekten bir değer yüklemeyi başardı!

Özetlemek gerekirse, şimdi her bellek referansında ne olduğuyla ilgili ilk protokolü açıklıyoruz. Şekil 18.6 (sayfa 10) yaklaşımı göstermektedir. Her bellek referansı için (bir talimat getirme veya açık bir yükleme veya depolama), sayfalama, önce sayfa tablosundan çeviriyi getirmek için fazladan bir bellek referansı gerçekleştirmemizi gerektirir. bu çok fazla

```
1  // Extract the VPN from the virtual address
2  VPN = (VirtualAddress & VPN_MASK) >> SHIFT
3
4  // Form the address of the page-table entry (PTE)
5  PTEAddr = PTBR + (VPN * sizeof(PTE))
6
7  // Fetch the PTE
8  PTE = AccessMemory(PTEAddr)
9
10 // Check if process can access the page
11 if (PTE.Valid == False)
12     RaiseException(SEGMENTATION_FAULT)
13 else if (CanAccess(PTE.ProtectBits) == False)
14     RaiseException(PROTECTION_FAULT)
15 else
16     // Access is OK: form physical address and fetch it
17     offset   = VirtualAddress & OFFSET_MASK
18     PhysAddr = (PTE.PFN << PFN_SHIFT) | offset
19     Register = AccessMemory(PhysAddr)
```

Şekil 18.6: **Disk Belleği ile Belleğe Erişme**

iş! Fazladan bellek referansları maliyetlidir ve bu durumda muhtemelen işlemi iki veya daha fazla kat yavaşlatacaktır.

Ve şimdi umarım çözmemiz gereken iki gerçek problem olduğunu görebilirsiniz. Hem donanımın hem de yazılımın dikkatli tasarımı olmadan, sayfa tabloları sistemin çok yavaş çalışmasına ve çok fazla bellek kaplamasına neden olur. Bellek sanallaştırma ihtiyaçlarımız için harika bir çözüm gibi görünse de, öncelikle bu iki önemli sorunun üstesinden gelinmesi gerekir.

18.5 Bir Hafıza İzi

Kapatmadan önce, sayfalama kullanılırken meydana gelen sonuçta ortaya çıkan tüm bellek erişimlerini göstermek için basit bir bellek erişim örneğini izleyeceğiz. İlgilendiğimiz kod parçacığı (C'de `array.c` adlı bir dosyada) aşağıdaki gibidir:

```
int array[1000];
...
for (i = 0; i < 1000; i++)
    array[i] = 0;
```

Array.c'yi derliyoruz ve aşağıdaki komutlarla çalıştırıyoruz:

```
prompt> gcc -o array array.c -Wall -O
prompt> ./array
```

Tabii ki, bu kod parçacığının (sadece bir diziyi başlatan) hangi belleğe eriştiğini gerçekten anlamak için, birkaç şey daha bilmemiz (veya varsaymamız) gerekecek. Öncelikle, diziyi bir döngüde başlatmak için hangi derleme yönergelerinin kullanıldığını görmek için (Linux'ta objdump veya Mac'te otool kullanarak) ortaya çıkan ikiliyi **parçalara ayırmamız (disassemble)** gerekecek. İşte ortaya çıkan montaj kodu:

```
1024 movl $0x0,(%edi,%eax,4)
1028 incl %eax
1032 cmpl $0x03e8,%eax
1036 jne  0x1024
```

Biraz **x86** biliyorsanız, kodu anlamak aslında oldukça kolaydır2. İlk talimat, sıfır değerini (`$0x0` olarak gösterilir) dizinin konumunun sanal bellek adresine taşır; bu adres, `%edi`'nin içeriği alınarak ve buna `%eax` çarpı dört eklenerek hesaplanır. Bu nedenle, `%edi` dizinin temel adresini tutarken, `%eax` dizi indeksini (`i`) tutar; dizi, her biri dört bayt boyutunda bir tamsayılar dizisi olduğu için dört ile çarparız.

İkinci komut, `%eax` içinde tutulan dizi indeksini artırır ve üçüncü komut, bu kaydın içeriğini onaltılık değer `0x03e8` veya ondalık sayı 1000 ile karşılaştırır. `jne` komut testleri), dördüncü komut döngünün en üstüne atlar.

Bu komut dizisinin hangi belleğe eriştiğini anlamak için (hem sanal hem de fiziksel düzeylerde), sanal bellekte kod parçacığının ve dizinin nerede bulunduğu ve ayrıca sayfa tablosunun içeriği ve konumu hakkında bir şeyler varsaymamız gerekecek. .

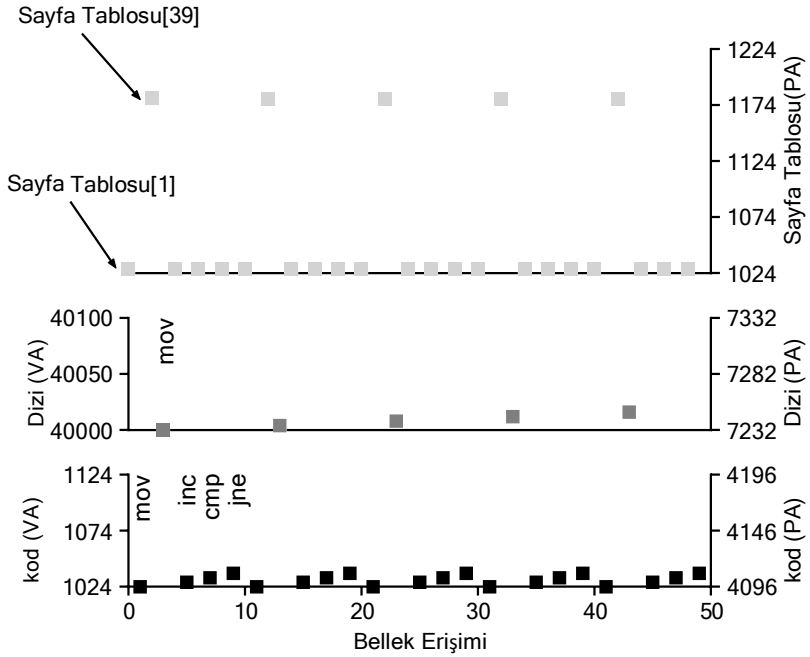
Bu örnek için, 64 KB boyutunda (gerçekçi olmayacak kadar küçük) bir sanal adres alanı varsayıyoruz. Ayrıca sayfa boyutunun 1 KB olduğunu varsayıyoruz.

Şimdi bilmemiz gereken tek şey, sayfa tablosunun içeriği ve fiziksel bellekteki konumu. Doğrusal (dizi tabanlı) bir sayfa tablomuz olduğunu ve bunun 1KB (1024) fiziksel adresinde bulunduğunu varsayalım.

İçeriğine gelince, bu örnek için haritalandırmayı düşünmemiz gereken sadece birkaç sanal sayfa var. İlk olarak, kodun yaşadığı sanal sayfa var. Sayfa boyutu 1KB olduğundan, sanal adres 1024, sanal adres alanının ikinci sayfasında bulunur (VPN=1, çünkü VPN=0 ilk sayfadır). Bu sanal sayfanın fiziksel çerçeve 4 (VPN 1 → PFN 4) ile eşleştiğini varsayalım.

Ardından, dizinin kendisi var. Boyutu 4000 bayttır (1000 tamsayı) ve 40000 ile 44000 arasındaki sanal adreslerde bulunduğunu varsayıyoruz (son bayt hariç). Bu ondalık aralık için sanal sayfalar VPN=39 ... VPN=42'dir. Bu nedenle, bu sayfalar için eşlemelere ihtiyacımız var. Örnek için bu sanal-fiziksel eşlemeleri varsayalım: (VPN 39 → PFN 7), (VPN 40 → PFN 8), (VPN 41 → PFN 9), (VPN 42 → PFN 10).

[2]Basitlik için her talimatın dört bayt boyutunda olduğunu varsayarak burada biraz hile yapıyoruz; gerçekte, x86 komutları değişken boyutludur.



Şekil 18.7: **Sanal (Ve Fiziksel) Bir Bellek İzi**

Artık programın bellek referanslarını izlemeye hazırız. Çalıştırıldığında, her talimat getirme iki bellek referansı üretecektir: biri talimatın içinde bulunduğu fiziksel çerçeveyi bulmak için sayfa tablosuna, diğeri ise onu işlenmek üzere CPU'ya getirmek için talimatın kendisine. Ek olarak, `mov` talimatı biçiminde bir açık bellek referansı vardır; bu, önce (dizi sanal adresini doğru fiziksel adrese çevirmek için) başka bir sayfa tablosu erişimi ve ardından dizinin kendisine erişim ekler.

İlk beş döngü yinelemesi için tüm süreç Şekil 18.7'de (sayfa 12) gösterilmektedir. En alttaki grafik, y eksenindeki komut belleği referanslarını siyah olarak gösterir (sanal adresler solda ve gerçek fiziksel adresler sağda); ortadaki grafik, dizi erişimlerini koyu gri renkte gösterir (yine solda sanal ve sağda fiziksel olarak); son olarak, en üstteki grafik sayfa tablosu belleği erişimlerini açık gri renkte gösterir (bu örnekteki sayfa tablosu fiziksel bellekte bulunduğu için yalnızca fiziksel). Tüm izleme için x ekseni, döngünün ilk beş yinelemesindeki bellek erişimlerini gösterir; döngü başına 10 bellek erişimi vardır; bu, dört talimat getirme, bir açık bellek güncellemesi ve bu dört getirme ve bir açık güncellemeyi çevirmek için beş sayfa tablosu erişimi içerir.

Bu görselleştirmede ortaya çıkan kalıpları anlamlandırabilecek misiniz bir bakın. Özellikle, döngü bu ilk beş yinelemenin ötesinde çalışmaya devam ettikçe ne değişecek? Hangi yeni bellek konumlarına erişilecek? Anlayabilir misin?

Bu, örneklerin en basitiydi (yalnızca birkaç satır C kodu) ve yine de gerçek uygulamaların gerçek bellek davranışını anlamanın karmaşıklığını şimdiden hissedebiliyor olabilirsiniz. Endişelenmeyin: kesinlikle daha kötüye gidiyor, çünkü tanıtmak üzere olduğumuz mekanizmalar zaten karmaşık olan bu mekanizmayı daha da karmaşık hale getiriyor. Üzgünüm[3]!

18.6 Özet

Belleği sanallaştırma sorunumuz için bir çözüm olarak **sayfalama (paging)** kavramını tanıttık. Sayfalamanın önceki yaklaşımlara göre (segmentasyon gibi) birçok avantajı vardır. Birincisi, disk belleği (tasarım gereği) belleği sabit boyutlu birimlere böldüğü için harici parçalanmaya yol açmaz. İkincisi, oldukça esnektir ve sanal reklam alanlarının seyrek olarak kullanılmasını sağlar.

Bununla birlikte, sayfalama desteğini dikkatsizce uygulamak, daha yavaş bir makineye (sayfa tablosuna erişmek için birçok ekstra bellek erişimi) ve ayrıca bellek israfına (kullanışlı uygulama verileri yerine sayfa tablolarıyla dolu bellek) yol açacaktır. Bu nedenle, yalnızca çalışan değil, aynı zamanda iyi çalışan bir çağrı sistemi bulmak için biraz daha fazla düşünmemiz gerekecek. Neyse ki sonraki iki bölüm bize bunu nasıl yapacağımızı gösterecek.

[3]Gerçekten üzgün değiliz. Ancak, eğer mantıklıysa, üzgün olmadığımız için üzgünüz.

Referanslar

[KE+62] “Tek Düzeyli Depolama Sistemi”, T. Kilburn, D.B.G. Edwards, MJ Lanigan, FH Sum ner. IRE Trans. EC-11, 2, 1962. Bell ve Newell'de yeniden basılmıştır, "Computer Structures: Readings and Samples". McGraw-Hill, New York, 1971. *Atlas, belleği sabit boyutlu sayfalara bölme fikrine öncülük etti ve birçok anlamda modern bilgisayar sistemlerinde gördüğümüz bellek yönetimi fikirlerinin erken bir biçimiydi.*

[I09] "Intel 64 ve IA-32 Mimarileri Yazılım Geliştirici Kılavuzları" Intel, 2009. Mevcut: http://www.intel.com/products/processor/manuals. *Özellikle, "Cilt 3A: Sistem Programlama Kılavuzu Bölüm 1" ve "Cilt 3B: Sistem Programlama Kılavuzu Bölüm 2"ye dikkat edin.*

[L78] "The Manchester Mark I and Atlas: A Historical Perspective", SH Lavington. İletişimmunications of the ACM, Cilt 21:1, Ocak 1978. *Bu makale, bazı önemli bilgisayar sistemlerinin gelişim tarihinin büyük bir retrospektifidir. ABD'de bazen unuttuğumuz gibi, bu yeni fikirlerin çoğu denizaşırı ülkelerden geldi.*

Ödev (Simülasyon)

Bu ödevde, sanaldan fiziksele adres çevirisinin doğrusal sayfa tablolarıyla nasıl çalıştığını anlayıp anlamadığınızı görmek için `paging-linear-translate.py`, olarak bilinen basit bir program kullanacaksınız. Ayrıntılar için BENİOKU'ya (README) bakın.

- Komutlarla kullanılmış olan bayrakların amaçları:

-s bayrağı rastgele çekirdeği değiştirir ve böylece çevrilecek farklı sanal adreslerin yanı sıra farklı sayfa tablosu değerleri oluşturur.

-a bayrağı, adres alanının boyutunu değiştirir.

-p bayrağı, fiziksel belleğin boyutunu değiştirir.

-P bayrağı bir sayfanın boyutunu değiştirir.

-n bayrağı, çevrilecek daha fazla adres oluşturmak için kullanılabilir.

-u bayrağı, geçerli eşlemelerin oranını %0'dan (-u 0) %100'e (-u 100) kadar değiştirir.

-v bayrağı, hayatınızı kolaylaştırmak için VPN numaralarını yazdırır.

Sorular

1. Herhangi bir çeviri yapmadan önce, farklı parametreler verildiğinde doğrusal sayfa tablolarının boyutunun nasıl değiştiğini incelemek için simülatörü kullanalım. Farklı parametreler değiştikçe doğrusal sayfa tablolarının boyutunu hesaplayın. Önerilen bazı girdiler aşağıdadır; `-v` işaretini kullanarak, kaç tane sayfa tablosu girişinin doldurulduğunu görebilirsiniz. İlk olarak, adres alanı büyüdükçe doğrusal sayfa tablosu boyutunun nasıl değiştiğini anlamak için şu bayraklarla çalıştırın:

`-P 1k -a 1m -p 512m -v -n 0`

Komut tarafından sağlanan verilere göre: adres alanı boyutu **1m (-a bayrağı)**, fiziksel bellek boyutu **512m (-p bayrağı)**, sayfa boyutu **1k (-P bayrağı)**, tohum **0** olarak belirtilmiştir.
-v işaretini kullanarak kaç tane sayfa tablosu girişinin doldurulduğunu görebiliriz. Sayfa tablosu girişinin formatı basittir: en soldaki (yüksek dereceli) bit, geçerli bittir. Bit 1 ise, girişin geri kalanı PFN'dir.Bit 0 ise, sayfa geçersizdir.

Bu komutu çalıştırdığımızda adres alanı büyüdükçe doğrusal sayfa tablosu boyutunun nasıl değiştiğini görebiliriz.

`./paging-linear-translate.py -P 1k -a 1m -p 512m -v -n 0`

Bu süreci çalıştırdığımızda aşağıdaki gibi bir ekran görüntüsü oluşmaktadır:

```
ayoubabdallah@ubuntu:~/Desktop/Ostep/ostep-homework/vm-paging$ python3 paging-linear-translate.py -P 1k -a 1m -p 512m -v -n 0
ARG seed 0
ARG address space size 1m
ARG phys mem size 512m
ARG page size 1k
ARG verbose True
ARG addresses -1


The format of the page table is simple:
The high-order (left-most) bit is the VALID bit.
  If the bit is 1, the rest of the entry is the PFN.
  If the bit is 0, the page is not valid.
Use verbose mode (-v) if you want to print the VPN # by
each entry of the page table.
```

```
  [      515]  0x00000000
  [      516]  0x00000000
  [      517]  0x800779c6
  [      518]  0x00000000
  [      519]  0x00000000
  [      520]  0x800248b7
  [     1019]  0x8002e9c9
  [     1020]  0x00000000
  [     1021]  0x00000000
  [     1022]  0x00000000
  [     1023]  0x00000000

Virtual Address Trace

For each virtual address, write down the physical address it translates to
OR write down that it is an out-of-bounds address (e.g., segfault).
```

`-P 1k -a 2m -p 512m -v -n 0`

Komut tarafından sağlanan verilere göre: adres alanı boyutu **2m (-a bayrağı)**, fiziksel bellek boyutu **512m (-p bayrağı)**, sayfa boyutu **1k (-P bayrağı)**, tohum **0** olarak belirtilmiştir.
-v işaretini kullanarak kaç tane sayfa tablosu girişinin doldurulduğunu görebiliriz. Sayfa tablosu girişinin formatı basittir: en soldaki (yüksek dereceli) bit, geçerli bittir. Bit 1 ise, girişin geri kalanı PFN'dir.Bit 0 ise, sayfa geçersizdir.

Bu komutu çalıştırdığımızda adres alanı büyüdükçe doğrusal sayfa tablosu boyutunun nasıl değiştiğini görebiliriz.

`./paging-linear-translate.py -P 1k -a 2m -p 512m -v -n 0`

Bu süreci çalıştırdığımızda aşağıdaki gibi bir ekran görüntüsü oluşmaktadır:

```
ayoubabdallah@ubuntu:~/Desktop/Ostep/ostep-homework/vm-paging$ python3 paging-linear-translate.py -P 1k -a 2m -p 512m -v -n 0
ARG seed 0
ARG address space size 2m
ARG phys mem size 512m
ARG page size 1k
ARG verbose True
ARG addresses -1


The format of the page table is simple:
The high-order (left-most) bit is the VALID bit.
  If the bit is 1, the rest of the entry is the PFN.
  If the bit is 0, the page is not valid.
Use verbose mode (-v) if you want to print the VPN # by
each entry of the page table.
  [     1031]  0x00000000
  [     1032]  0x00000000
  [     1033]  0x80030e30
  [     1034]  0x80037c8b
  [     1035]  0x00000000
  [     1036]  0x8004ed14
  [     1037]  0x00000000
  [     2043]  0x00000000
  [     2044]  0x00000000
  [     2045]  0x00000000
  [     2046]  0x8000eedd
  [     2047]  0x00000000

Virtual Address Trace

For each virtual address, write down the physical address it translates to
OR write down that it is an out-of-bounds address (e.g., segfault).
```

-P 1k -a 4m -p 512m -v -n 0

Komut tarafından sağlanan verilere göre: adres alanı boyutu **4m (-a bayrağı)**, fiziksel bellek boyutu **512m (-p bayrağı)**, sayfa boyutu **1k (-P bayrağı)**, tohum **0** olarak belirtilmiştir.
-v işaretini kullanarak kaç tane sayfa tablosu girişinin doldurulduğunu görebiliriz. Sayfa tablosu girişinin formatı basittir: en soldaki (yüksek dereceli) bit, geçerli bittir. Bit 1 ise, girişin geri kalanı PFN'dir.Bit 0 ise, sayfa geçersizdir.

Bu komutu çalıştırdığımızda adres alanı büyüdükçe doğrusal sayfa tablosu boyutunun nasıl değiştiğini görebiliriz.

./paging-linear-translate.py -P 1k -a 4m -p 512m -v -n 0

Bu süreci çalıştırdığımızda aşağıdaki gibi bir ekran görüntüsü oluşmaktadır:

```
ayoubabdallah@ubuntu:~/Desktop/Ostep/ostep-homework/vm-paging$ python3 paging-linear-translate.py -P 1k -a 4m -p 512m -v -n 0
ARG seed 0
ARG address space size 4m
ARG phys mem size 512m
ARG page size 1k
ARG verbose True
ARG addresses -1


The format of the page table is simple:
The high-order (left-most) bit is the VALID bit.
  If the bit is 1, the rest of the entry is the PFN.
  If the bit is 0, the page is not valid.
Use verbose mode (-v) if you want to print the VPN # by
each entry of the page table.
  [     2016]  0x8004fe74
  [     2017]  0x00000000
  [     2018]  0x8007f84f
  [     2019]  0x00000000
  [     2020]  0x800541d7
  [     2021]  0x00000000
    [     4091]  0x800160f8
    [     4092]  0x80015abc
    [     4093]  0x8001483a
    [     4094]  0x00000000
    [     4095]  0x8002e298

Virtual Address Trace

For each virtual address, write down the physical address it translates to
OR write down that it is an out-of-bounds address (e.g., segfault).
```

Ardından, sayfa boyutu büyüdükçe doğrusal sayfa tablosu boyutunun nasıl değiştiğini anlamak için:

-P 1k -a 1m -p 512m -v -n 0

Komut tarafından sağlanan verilere göre: adres alanı boyutu **1m (-a bayrağı)**, fiziksel bellek boyutu **512m (-p bayrağı)**, sayfa boyutu **1k (-P bayrağı)**, tohum **0** olarak belirtilmiştir.
-v işaretini kullanarak kaç tane sayfa tablosu girişinin doldurulduğunu görebiliriz. Sayfa tablosu girişinin formatı basittir: en soldaki (yüksek dereceli) bit, geçerli bittir. Bit 1 ise, girişin geri kalanı PFN'dir.Bit 0 ise, sayfa geçersizdir.

Bu komutu çalıştırdığımızda sayfa boyutu büyüdükçe doğrusal sayfa tablosu boyutunun nasıl değiştiğini görebiliriz.

./paging-linear-translate.py -P 1k -a 1m -p 512m -v -n 0

Bu süreci çalıştırdığımızda aşağıdaki gibi bir ekran görüntüsü oluşmaktadır:

```
ayoubabdallah@ubuntu:~/Desktop/Ostep/ostep-homework/vm-paging$ python3 paging-linear-translate.py -P 1k -a 1m -p 512m -v -n 0
ARG seed 0
ARG address space size 1m
ARG phys mem size 512m
ARG page size 1k
ARG verbose True
ARG addresses -1


The format of the page table is simple:
The high-order (left-most) bit is the VALID bit.
  If the bit is 1, the rest of the entry is the PFN.
  If the bit is 0, the page is not valid.
Use verbose mode (-v) if you want to print the VPN # by
each entry of the page table.

  [       550]  0x80064065
  [       551]  0x80048e54
  [       552]  0x8001b529
  [       553]  0x80068786
  [       554]  0x8002d3e4
  [      1019]  0x8002e9c9
  [      1020]  0x00000000
  [      1021]  0x00000000
  [      1022]  0x00000000
  [      1023]  0x00000000

Virtual Address Trace

For each virtual address, write down the physical address it translates to
OR write down that it is an out-of-bounds address (e.g., segfault).
```

-P 2k -a 1m -p 512m -v -n 0

Komut tarafından sağlanan verilere göre: adres alanı boyutu **1m (-a bayrağı)**, fiziksel bellek boyutu **512m (-p bayrağı)**, sayfa boyutu **2k (-P bayrağı)**, tohum **0** olarak belirtilmiştir.
-v işaretini kullanarak kaç tane sayfa tablosu girişinin doldurulduğunu görebiliriz. Sayfa tablosu girişinin formatı basittir: en soldaki (yüksek dereceli) bit, geçerli bittir. Bit 1 ise, girişin geri kalanı PFN'dir. Bit 0 ise, sayfa geçersizdir.

Bu komutu çalıştırdığımızda sayfa boyutu büyüdükçe doğrusal sayfa tablosu boyutunun nasıl değiştiğini görebiliriz.

./paging-linear-translate.py -P 2k -a 1m -p 512m -v -n 0
Bu süreci çalıştırdığımızda aşağıdaki gibi bir ekran görüntüsü oluşmaktadır:

```
ayoubabdallah@ubuntu:~/Desktop/Ostep/ostep-homework/vm-paging$ python3 paging-linear-translate.py -P 2k -a 1m -p 512m -v -n 0
ARG seed 0
ARG address space size 1m
ARG phys mem size 512m
ARG page size 2k
ARG verbose True
ARG addresses -1


The format of the page table is simple:
The high-order (left-most) bit is the VALID bit.
  If the bit is 1, the rest of the entry is the PFN.
  If the bit is 0, the page is not valid.
Use verbose mode (-v) if you want to print the VPN # by
each entry of the page table.
```

```
[        250]   0x00000000
[        251]   0x8003dfd8
[        252]   0x80039ab7
[        253]   0x80024ba5
[        254]   0x8003386e
  [     507]  0x00000000
  [     508]  0x8001a7f2
  [     509]  0x8001c337
  [     510]  0x00000000
  [     511]  0x00000000

Virtual Address Trace

For each virtual address, write down the physical address it translates to
OR write down that it is an out-of-bounds address (e.g., segfault).
```

`-P 4k -a 1m -p 512m -v -n 0`

Komut tarafından sağlanan verilere göre: adres alanı boyutu **1m (-a bayrağı)**, fiziksel bellek boyutu **512m (-p bayrağı)**, sayfa boyutu **4k (-P bayrağı)**, tohum **0** olarak belirtilmiştir.
-v işaretini kullanarak kaç tane sayfa tablosu girişinin doldurulduğunu görebiliriz. Sayfa tablosu girişinin formatı basittir: en soldaki (yüksek dereceli) bit, geçerli bittir. Bit 1 ise, girişin geri kalanı PFN'dir. Bit 0 ise, sayfa geçersizdir.

Bu komutu çalıştırdığımızda sayfa boyutu büyüdükçe doğrusal sayfa tablosu boyutunun nasıl değiştiğini görebiliriz.

`./paging-linear-translate.py -P 4k -a 1m -p 512m -v -n 0`
Bu süreci çalıştırdığımızda aşağıdaki gibi bir ekran görüntüsü oluşmaktadır:

```
ayoubabdallah@ubuntu:~/Desktop/Ostep/ostep-homework/vm-paging$  python3 paging-linear-translate.py -P 4k -a 1m -p 512m -v -n 0
ARG seed 0
ARG address space size 1m
ARG phys mem size 512m
ARG page size 4k
ARG verbose True
ARG addresses -1


The format of the page table is simple:
The high-order (left-most) bit is the VALID bit.
  If the bit is 1, the rest of the entry is the PFN.
  If the bit is 0, the page is not valid.
Use verbose mode (-v) if you want to print the VPN # by
each entry of the page table.
   [        125]   0x00000000
   [        126]   0x00000000
   [        127]   0x8000d621
   [        128]   0x800017f3
   [        129]   0x00000000
  [     251]  0x8001efec
  [     252]  0x8001cd5b
  [     253]  0x800125d2
  [     254]  0x80019c37
  [     255]  0x8001fb27

Virtual Address Trace

For each virtual address, write down the physical address it translates to
OR write down that it is an out-of-bounds address (e.g., segfault).
```

Bunlardan herhangi birini çalıştırmadan önce, beklenen trendler hakkında düşünmeye çalışın. Adres alanı büyüdükçe sayfa tablosu boyutu nasıl değişmelidir? Sayfa boyutu büyüdükçe? Neden genel olarak büyük sayfalar kullanmıyorsunuz?

Adres alanı büyüdükçe sayfa tablosu boyutu da artar, çünkü tüm adres alanını kaplamak için daha fazla sayfaya ihtiyacımız vardır. Sayfa boyutları arttığında, tüm adres alanını kaplamak için daha az sayfaya ihtiyacımız olduğu için (boyut olarak daha büyük oldukları için) sayfa tablosu boyutu azalır.

Genel olarak çok büyük sayfalar kullanmamalıyız çünkü çok fazla hafıza kaybı olur. Çünkü çoğu işlem çok az bellek kullanır.

2. Şimdi birkaç çeviri yapalım. Bazı küçük örneklerle başlayın ve -u bayrağıyla adres alanına ayrılan sayfaların sayısını değiştirin. Örneğin:

```
-P 1k -a 16k -p 32k -v -u 0
```

Komut tarafından sağlanan verilere göre: adres alanı boyutu **16k**, fiziksel bellek boyutu **32k**, sayfa boyutu **1k**, tohum **0** olarak belirtilmiştir.
Bu komutu çalıştırdığımızda -u bayrağıyla adres alanına ayrılan sayfaların sayısını nasıl değiştiğini görebiliriz.
-v işareti VPN'yi (dizin) sayfa tablosuna yazdırır.
O halde verilen sanal adresleri fiziksel adreslere çevirmek için bu sayfa tablosunu kullanmaktır.
Bir sanal adresi fiziksel bir adrese çevirmek için önce onu bileşen bileşenlerine ayırmalıyız: sanal sayfa numarası ve ofset.

```
./paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 0
```

Bu süreci çalıştırdığımızda aşağıdaki gibi bir ekran görüntüsü oluşmaktadır:

```
ayoubabdallah@ubuntu:~/Desktop/Ostep/ostep-homework/vm-paging$ python3 paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 0
ARG seed 0
ARG address space size 16k
ARG phys mem size 32k
ARG page size 1k
ARG verbose True
ARG addresses -1

The format of the page table is simple:
The high-order (left-most) bit is the VALID bit.
  If the bit is 1, the rest of the entry is the PFN.
  If the bit is 0, the page is not valid.
Use verbose mode (-v) if you want to print the VPN # by
each entry of the page table.

Page Table (from entry 0 down to the max size)
  [       0]  0x00000000
  [       1]  0x00000000
  [       2]  0x00000000
  [       3]  0x00000000
  [       4]  0x00000000
  [       5]  0x00000000
  [       6]  0x00000000
  [       7]  0x00000000
  [       8]  0x00000000
  [       9]  0x00000000
  [      10]  0x00000000
  [      11]  0x00000000
  [      12]  0x00000000
  [      13]  0x00000000
  [      14]  0x00000000
  [      15]  0x00000000

Virtual Address Trace
  VA 0x00003a39 (decimal:    14905) --> PA or invalid address?
  VA 0x00003ee5 (decimal:    16101) --> PA or invalid address?
  VA 0x00003da (decimal:     13274) --> PA or invalid address?
  VA 0x000039bd (decimal:    14781) --> PA or invalid address?
  VA 0x000013d9 (decimal:     5081) --> PA or invalid address?

For each virtual address, write down the physical address it translates to
OR write down that it is an out-of-bounds address (e.g., segfault).
```

-P 1k -a 16k -p 32k -v -u 25

Komut tarafından sağlanan verilere göre: adres alanı boyutu **16k**, fiziksel bellek boyutu **32k**, sayfa boyutu **1k**, tohum **0** olarak belirtilmiştir.
Bu komutu çalıştırdığımızda -u bayrağıyla adres alanına ayrılan sayfaların sayısını nasıl değiştiğini görebiliriz.
-v işareti VPN'yi (dizin) sayfa tablosuna yazdırır.
O halde verilen sanal adresleri fiziksel adreslere çevirmek için bu sayfa tablosunu kullanmaktır.
Bir sanal adresi fiziksel bir adrese çevirmek için önce onu bileşen bileşenlerine ayırmalıyız: sanal sayfa numarası ve ofset.

./paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 25

Bu süreci çalıştırdığımızda aşağıdaki gibi bir ekran görüntüsü oluşmaktadır:

```
ayoubabdallah@ubuntu:~/Desktop/Ostep/ostep-homework/vm-paging$ python3 paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 25
ARG seed 0
ARG address space size 16k
ARG phys mem size 32k
ARG page size 1k
ARG verbose True
ARG addresses -1


The format of the page table is simple:
The high-order (left-most) bit is the VALID bit.
  If the bit is 1, the rest of the entry is the PFN.
  If the bit is 0, the page is not valid.
Use verbose mode (-v) if you want to print the VPN # by
each entry of the page table.

Page Table (from entry 0 down to the max size)
  [       0]  0x80000018
  [       1]  0x00000000
  [       2]  0x00000000
  [       3]  0x00000000
  [       4]  0x00000000
  [       5]  0x80000009
  [       6]  0x00000000
  [       7]  0x00000000
  [       8]  0x80000010
  [       9]  0x00000000
  [      10]  0x80000013
  [      11]  0x00000000
  [      12]  0x8000001f
  [      13]  0x8000001c
  [      14]  0x00000000
  [      15]  0x00000000

Virtual Address Trace
  VA 0x00003986 (decimal:    14726) --> PA or invalid address?
  VA 0x00002bc6 (decimal:    11206) --> PA or invalid address?
  VA 0x00001e37 (decimal:     7735) --> PA or invalid address?
  VA 0x00000671 (decimal:     1649) --> PA or invalid address?
  VA 0x00001bc9 (decimal:     7113) --> PA or invalid address?

For each virtual address, write down the physical address it translates to
OR write down that it is an out-of-bounds address (e.g., segfault).
```

-P 1k -a 16k -p 32k -v -u 50

Komut tarafından sağlanan verilere göre: adres alanı boyutu **16k**, fiziksel bellek boyutu **32k**, sayfa boyutu **1k**, tohum **0** olarak belirtilmiştir.
Bu komutu çalıştırdığımızda -u bayrağıyla adres alanına ayrılan sayfaların sayısını nasıl değiştiğini görebiliriz.
-v işareti VPN'yi (dizin) sayfa tablosuna yazdırır.
O halde verilen sanal adresleri fiziksel adreslere çevirmek için bu sayfa tablosunu kullanmaktır.
Bir sanal adresi fiziksel bir adrese çevirmek için önce onu bileşen bileşenlerine ayırmalıyız: sanal sayfa numarası ve ofset.

./paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 50

Bu süreci çalıştırdığımızda aşağıdaki gibi bir ekran görüntüsü oluşmaktadır:

```
ayoubabdallah@ubuntu:~/Desktop/Ostep/ostep-homework/vm-paging$ python3 paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 50
ARG seed 0
ARG address space size 16k
ARG phys mem size 32k
ARG page size 1k
ARG verbose True
ARG addresses -1


The format of the page table is simple:
The high-order (left-most) bit is the VALID bit.
  If the bit is 1, the rest of the entry is the PFN.
  If the bit is 0, the page is not valid.
Use verbose mode (-v) if you want to print the VPN # by
each entry of the page table.

Page Table (from entry 0 down to the max size)
  [       0]  0x80000018
  [       1]  0x00000000
  [       2]  0x00000000
  [       3]  0x8000000c
  [       4]  0x80000009
  [       5]  0x00000000
  [       6]  0x8000001d
  [       7]  0x80000013
  [       8]  0x00000000
  [       9]  0x8000001f
  [      10]  0x8000001c
  [      11]  0x00000000
  [      12]  0x8000000f
  [      13]  0x00000000
  [      14]  0x00000000
  [      15]  0x80000008

Virtual Address Trace
  VA 0x00003385 (decimal:    13189) --> PA or invalid address?
  VA 0x0000231d (decimal:     8989) --> PA or invalid address?
  VA 0x000000e6 (decimal:      230) --> PA or invalid address?
  VA 0x00002e0f (decimal:    11791) --> PA or invalid address?
  VA 0x00001986 (decimal:     6534) --> PA or invalid address?

For each virtual address, write down the physical address it translates to
OR write down that it is an out-of-bounds address (e.g., segfault).
```

`-P 1k -a 16k -p 32k -v -u 75`

Komut tarafından sağlanan verilere göre: adres alanı boyutu **16k**, fiziksel bellek boyutu **32k**, sayfa boyutu **1k**, tohum **0** olarak belirtilmiştir.
Bu komutu çalıştırdığımızda **-u** bayrağıyla adres alanına ayrılan sayfaların sayısını nasıl değiştiğini görebiliriz.
-v işareti VPN'yi (dizin) sayfa tablosuna yazdırır.
O halde verilen sanal adresleri fiziksel adreslere çevirmek için bu sayfa tablosunu kullanmaktır.
Bir sanal adresi fiziksel bir adrese çevirmek için önce onu bileşen bileşenlerine ayırmalıyız: sanal sayfa numarası ve ofset.

`./paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 75`

Bu süreci çalıştırdığımızda aşağıdaki gibi bir ekran görüntüsü oluşmaktadır:

```
ayoubabdallah@ubuntu:~/Desktop/Ostep/ostep-homework/vm-paging$ python3 paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 75
ARG seed 0
ARG address space size 16k
ARG phys mem size 32k
ARG page size 1k
ARG verbose True
ARG addresses -1


The format of the page table is simple:
The high-order (left-most) bit is the VALID bit.
  If the bit is 1, the rest of the entry is the PFN.
  If the bit is 0, the page is not valid.
Use verbose mode (-v) if you want to print the VPN # by
each entry of the page table.

Page Table (from entry 0 down to the max size)
  [       0]  0x80000018
  [       1]  0x80000008
  [       2]  0x8000000c
  [       3]  0x80000009
  [       4]  0x80000012
  [       5]  0x80000010
  [       6]  0x8000001f
  [       7]  0x8000001c
  [       8]  0x80000017
  [       9]  0x80000015
  [      10]  0x80000003
  [      11]  0x80000013
  [      12]  0x8000001e
  [      13]  0x8000001b
  [      14]  0x80000019
  [      15]  0x80000000

Virtual Address Trace
  VA 0x00002e0f (decimal:    11791) --> PA or invalid address?
  VA 0x00001986 (decimal:     6534) --> PA or invalid address?
  VA 0x000034ca (decimal:    13514) --> PA or invalid address?
  VA 0x00002ac3 (decimal:    10947) --> PA or invalid address?
  VA 0x00000012 (decimal:       18) --> PA or invalid address?

For each virtual address, write down the physical address it translates to
OR write down that it is an out-of-bounds address (e.g., segfault).
```

-P 1k -a 16k -p 32k -v -u 100

Komut tarafından sağlanan verilere göre: adres alanı boyutu **16k**, fiziksel bellek boyutu **32k**, sayfa boyutu **1k**, tohum **0** olarak belirtilmiştir.
Bu komutu çalıştırdığımızda **-u** bayrağıyla adres alanına ayrılan sayfaların sayısını nasıl değiştiğini görebiliriz.
-v işareti VPN'yi (dizin) sayfa tablosuna yazdırır.
O halde verilen sanal adresleri fiziksel adreslere çevirmek için bu sayfa tablosunu kullanmaktır.
Bir sanal adresi fiziksel bir adrese çevirmek için önce onu bileşen bileşenlerine ayırmalıyız: sanal sayfa numarası ve ofset.

./paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 100

Bu süreci çalıştırdığımızda aşağıdaki gibi bir ekran görüntüsü oluşmaktadır:

```
ayoubabdallah@ubuntu:~/Desktop/Ostep/ostep-homework/vm-paging$ python3 paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 100
ARG seed 0
ARG address space size 16k
ARG phys mem size 32k
ARG page size 1k
ARG verbose True
ARG addresses -1


The format of the page table is simple:
The high-order (left-most) bit is the VALID bit.
  If the bit is 1, the rest of the entry is the PFN.
  If the bit is 0, the page is not valid.
Use verbose mode (-v) if you want to print the VPN # by
each entry of the page table.

Page Table (from entry 0 down to the max size)
  [       0]  0x80000018
  [       1]  0x80000008
  [       2]  0x8000000c
  [       3]  0x80000009
  [       4]  0x80000012
  [       5]  0x80000010
  [       6]  0x8000001f
  [       7]  0x8000001c
  [       8]  0x80000017
  [       9]  0x80000015
  [      10]  0x80000003
  [      11]  0x80000013
  [      12]  0x8000001e
  [      13]  0x8000001b
  [      14]  0x80000019
  [      15]  0x80000000

Virtual Address Trace
  VA 0x00002e0f (decimal:    11791) --> PA or invalid address?
  VA 0x00001986 (decimal:     6534) --> PA or invalid address?
  VA 0x000034ca (decimal:    13514) --> PA or invalid address?
  VA 0x00002ac3 (decimal:    10947) --> PA or invalid address?
  VA 0x00000012 (decimal:       18) --> PA or invalid address?

For each virtual address, write down the physical address it translates to
OR write down that it is an out-of-bounds address (e.g., segfault).
```

Her bir adres alanına ayrılan sayfaların yüzdesini artırdığınızda ne olur?

1. Adres alanı kullanılmaz (doldurulmaz). Bu nedenle her bellek erişimi başarılı olamadı.

2. Yalnızca VA 0x2bc6 geçerlidir.

- Adres alanı 16 KB olduğundan toplamda 14 bitimiz olur. Ancak sayfa boyutumuz 1kb olduğu için sayfamızda dolaşmak için 10 bite ihtiyacımız var. Böylece, diğer 4 bit, sayfa tablomuzun indeksi olarak kullanılacaktır.

- 0x2bc6'nın ikili değeri 10 1011 1100 0110'dur.

- Yani, VPN 1010, Sayfa Tablosunun 10. indeksinin değerinin PPN olduğu anlamına gelir

- Ofset 11 1100 0110

- Sayfa tablosuna baktığımızda 10. indekste 0x13 elde ederiz.
1. Ancak, 10 kez sola kaydırmamız gerekiyor (ofset bit sayısı)
2. Böylece 0100 1100 0000 0000 olur.

- O zaman bunu ofset ile VEYA yapmalıyız (0011 1100 0110 VEYA 0100 1100 0000 0000)

- Sonunda 0x4FC6 olur

Ayrılan sayfaların yüzdesi veya adres alanı kullanımı arttıkça, daha fazla bellek erişim işlemi geçerli olur ancak boş alan azalır.

3. Şimdi çeşitlilik için bazı farklı rasgele tohumlar ve bazı farklı (ve bazen oldukça çılgın) adres alanı parametrelerini deneyelim:

`-P 8 -a 32 -p 1024 -v -s 1`

Komut tarafından sağlanan verilere göre: adres alanı boyutu **32,** fiziksel bellek boyutu **1024,** sayfa boyutu **8,** rastgele tohumu **1** olarak belirtilmiştir.

-s bayrağı kullanarak rastgele çekirdeği (tohumu) değiştirebiliriz ve böylece çevrilecek farklı sanal adreslerin yanı sıra farklı sayfa tablosu değerleri oluşturur.

-v işareti VPN'yi (dizin) sayfa tablosuna yazdırır.
O halde verilen sanal adresleri fiziksel adreslere çevirmek için bu sayfa tablosunu kullanmaktır.
Bir sanal adresi fiziksel bir adrese çevirmek için önce onu bileşen bileşenlerine ayırmalıyız: sanal sayfa numarası ve ofset.

`./paging-linear-translate.py -P 8 -a 32 -p 1024 -v -s 1`
Bu süreci çalıştırdığımızda aşağıdaki gibi bir ekran görüntüsü oluşmaktadır:

```
ayoubabdallah@ubuntu:~/Desktop/Ostep/ostep-homework/vm-paging$ python3 paging-linear-translate.py -P 8 -a 32 -p 1024 -v -s 1
ARG seed 1
ARG address space size 32
ARG phys mem size 1024
ARG page size 8
ARG verbose True
ARG addresses -1


The format of the page table is simple:
The high-order (left-most) bit is the VALID bit.
  If the bit is 1, the rest of the entry is the PFN.
  If the bit is 0, the page is not valid.
Use verbose mode (-v) if you want to print the VPN # by
each entry of the page table.

Page Table (from entry 0 down to the max size)
  [       0]  0x00000000
  [       1]  0x80000061
  [       2]  0x00000000
  [       3]  0x00000000

Virtual Address Trace
  VA 0x0000000e (decimal:       14) --> PA or invalid address?
  VA 0x00000014 (decimal:       20) --> PA or invalid address?
  VA 0x00000019 (decimal:       25) --> PA or invalid address?
  VA 0x00000003 (decimal:        3) --> PA or invalid address?
  VA 0x00000000 (decimal:        0) --> PA or invalid address?

For each virtual address, write down the physical address it translates to
OR write down that it is an out-of-bounds address (e.g., segfault).
```

-P 8k -a 32k -p 1m -v -s 2

Komut tarafından sağlanan verilere göre: adres alanı boyutu **32k**, fiziksel bellek boyutu **1m**, sayfa boyutu **8k**, rastgele tohumu **2** olarak belirtilmiştir.

-s bayrağı kullanarak rastgele çekirdeği (tohumu) değiştirebiliriz ve böylece çevrilecek farklı sanal adreslerin yanı sıra farklı sayfa tablosu değerleri oluşturur.

-v işareti VPN'yi (dizin) sayfa tablosuna yazdırır.
O halde verilen sanal adresleri fiziksel adreslere çevirmek için bu sayfa tablosunu kullanmaktır.
Bir sanal adresi fiziksel bir adrese çevirmek için önce onu bileşen bileşenlerine ayırmalıyız: sanal sayfa numarası ve ofset.

./paging-linear-translate.py -P 8 -a 32 -p 1024 -v -s 2
Bu süreci çalıştırdığımızda aşağıdaki gibi bir ekran görüntüsü oluşmaktadır:

```
ayoubabdallah@ubuntu:~/Desktop/Ostep/ostep-homework/vm-paging$ python3 paging-linear-translate.py -P 8k -a 32k -p 1m   -v -s 2
ARG seed 2
ARG address space size 32k
ARG phys mem size 1m
ARG page size 8k
ARG verbose True
ARG addresses -1


The format of the page table is simple:
The high-order (left-most) bit is the VALID bit.
  If the bit is 1, the rest of the entry is the PFN.
  If the bit is 0, the page is not valid.
Use verbose mode (-v) if you want to print the VPN # by
each entry of the page table.

Page Table (from entry 0 down to the max size)
  [       0]  0x80000079
  [       1]  0x00000000
  [       2]  0x00000000
  [       3]  0x8000005e

Virtual Address Trace
  VA 0x000055b9 (decimal:    21945) --> PA or invalid address?
  VA 0x00002771 (decimal:    10097) --> PA or invalid address?
  VA 0x00004d8f (decimal:    19855) --> PA or invalid address?
  VA 0x00004dab (decimal:    19883) --> PA or invalid address?
  VA 0x00004a64 (decimal:    19044) --> PA or invalid address?

For each virtual address, write down the physical address it translates to
OR write down that it is an out-of-bounds address (e.g., segfault).
```

-P 1m -a 256m -p 512m -v -s 3

Komut tarafından sağlanan verilere göre: adres alanı boyutu **256m**, fiziksel bellek boyutu **512m**, sayfa boyutu **1m**, rastgele tohumu **3** olarak belirtilmiştir.

-s bayrağı kullanarak rastgele çekirdeği (tohumu) değiştirebiliriz ve böylece çevrilecek farklı sanal adreslerin yanı sıra farklı sayfa tablosu değerleri oluşturur.

-v işareti VPN'yi (dizin) sayfa tablosuna yazdırır.
O halde verilen sanal adresleri fiziksel adreslere çevirmek için bu sayfa tablosunu kullanmaktır.
Bir sanal adresi fiziksel bir adrese çevirmek için önce onu bileşen bileşenlerine ayırmalıyız: sanal sayfa numarası ve ofset.

./paging-linear-translate.py -P 8 -a 32 -p 1024 -v -s 3
Bu süreci çalıştırdığımızda aşağıdaki gibi bir ekran görüntüsü oluşmaktadır:

```
ayoubabdallah@ubuntu:~/Desktop/Ostep/ostep-homework/vm-paging$ python3 paging-linear-translate.py -P 1m -a 256m -p 512m -v -s 3
ARG seed 3
ARG address space size 256m
ARG phys mem size 512m
ARG page size 1m
ARG verbose True
ARG addresses -1


The format of the page table is simple:
The high-order (left-most) bit is the VALID bit.
  If the bit is 1, the rest of the entry is the PFN.
  If the bit is 0, the page is not valid.
Use verbose mode (-v) if you want to print the VPN # by
each entry of the page table.
   [         121]    0x800001ac
   [         122]    0x80000109
   [         123]    0x00000000
   [         124]    0x00000000
   [         125]    0x00000000
   [         126]    0x00000000
   [         127]    0x80000092
   [     252]  0x00000000
   [     253]  0x00000000
   [     254]  0x80000159
   [     255]  0x00000000

Virtual Address Trace
  VA 0x0308b24d (decimal: 50901581) --> PA or invalid address?
  VA 0x042351e6 (decimal: 69423590) --> PA or invalid address?
  VA 0x02feb67b (decimal: 50247291) --> PA or invalid address?
  VA 0x0b46977d (decimal: 189175677) --> PA or invalid address?
  VA 0x0dbcceb4 (decimal: 230477492) --> PA or invalid address?

For each virtual address, write down the physical address it translates to
OR write down that it is an out-of-bounds address (e.g., segfault).
```

Bu parametre kombinasyonlarından hangisi gerçekçi değil? Neden?

1. (birinci) Parametre kombinasyonu gerçekçi değil, bunun nedeni, boyutunun çok küçük olmasıdır.

4. Diğer sorunları denemek için programı kullanın. Programın artık çalışmadığı yerlerin sınırlarını bulabilir misiniz? Örneğin, adres alanı boyutu fiziksel bellekten büyükse ne olur?

Ne zaman işe yaramayacak?

- Sayfa boyutu adres alanından daha büyük.
- Adres alanı boyutu fiziksel bellekten daha büyük.
- Fiziksel bellek boyutu sayfa boyutunun katları değil.
- Adres alanı, sayfa boyutunun katları değil.
- Sayfa boyutu negatif.
- Fiziksel hafıza negatiftir.
- Adres alanı negatiftir.