

Soyutlama: Adres Alanları

Eskiden bilgisayar sistemleri oluşturmak kolaydı. Nedenini biliyor musun? Çünkü kullanıcılar fazla bir şey beklemiyordu. Tüm bu baş ağrılarına "kullanım kolaylığı", "yüksek performans", "güvenilirlik" vb. beklentileri olan kullanıcılar neden olmuştur. Bir sonraki seferde bu bilgisayar kullanıcılarından biriyle tanıştığınızda, neden oldukları tüm sorunlar için onlara teşekkür edebilirsiniz.

13.1 İlkel Sistemler

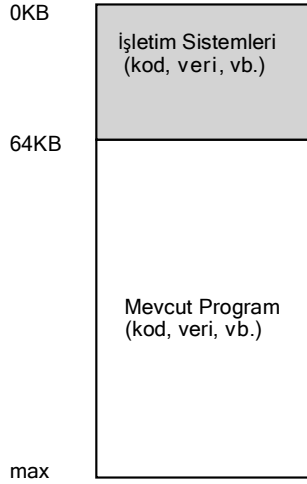
Bellek açısından bakıldığında, ilk makineler kullanıcılara çok fazla soyutlama sağlamadı. Temel olarak, makinenin fiziksel belleği (Sayfa 2, Şekil 13.1'de) gördüğünüz gibi görünüyordu.

İşletim Sistemi, bellekte bulunan (bu örnekte fiziksel adres 0'dan başlayarak) bir dizi rutin şeklindeydi (aslında bir kitaplık) ve şu an da fiziksel bellekte (başlangıçta başlayan) çalışan bir program (bir işlem) olacaktı. (Bu örnek de 64k fiziksel adresinde) ve belleğin geri kalanını kullandı. Burada çok az yanılama vardı ve kullanıcı işletim sisteminden pek bir şey beklemiyordu. O günlerde işletim sistemi geliştiricileri için hayat kesinlikle kolaydı, değil mi?

Çoklu Programlama ve Zaman Paylaşımı

Bir süre sonra makineler pahalı olduğu için insanlar makineleri daha etkin bir şekilde paylaşmaya başladılar. Böylece, birden çok işlemin belirli bir zamanda çalışmaya hazır olduğu ve örneğin bir Giriş/Çıkış işlemi gerçekleştirmeye karar verildiğinde işletim sisteminin bunlar arasında geçiş yaptığı **çoklu programlama (multiprogramming)** çağı [DV66] doğdu. Bunu yapmak, CPU'nun etkin **kullanımını(utilization)** artırdı. **Verimlilikteki(Efficiency)** bu tür artışlar, her makinenin yüz binlerce hatta milyonlarca dolara mal olduğu (ve Mac'inizin pahalı olduğunu düşündüğünüz!) günlerde bile önemliydi.

Ancak çok geçmeden insanlar daha fazla makine talep etmeye başladı ve **zaman paylaşımı(time sharing)** çağı doğdu [S59, L60, M62, M83]. Spesifik olarak, birçoğu, özellikle uzun (ve dolayısıyla etkisiz) program hata ayıklama döngülerinden bıkmış olan programcıların [CV65] üzerinde, toplu hesaplamaların sınırlamalarını fark etti.

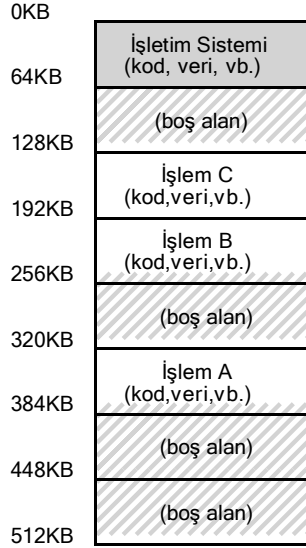


Şekil 13.1: İşletim Sistemleri: İlkel Dönem

Pek çok kullanıcı aynı anda bir makineyi kullanıyor olabileceğinden ve her biri yürütmekte oldukları görevlerden zamanında bir yanıt beklediğinden (veya umduğundan) **etkileşim(interactivity)** kavramı önemli hale gelmişti. Zaman paylaşımını uygulamanın bir yolu, bir işlemi kısa bir süre çalıştırarak tüm belleğe tam erişim vermek (Şekil 13.1), ardından durdurmak, tüm durumunu bir tür diske kaydetmek (tüm fiziksel bellek dahil olabilir). Başka bir işlemin durumunu yükleyin, bir süre çalıştırın ve böylece uygulayın. Makinenin ham olan bir tür paylaşımı olarak [M+63] gösterilebilir.

Ne yazık ki, bu yaklaşımın büyük bir sorunu var: özellikle hafıza arttıkça çok yavaş kalıyor. Kayıt düzeyinde durumun (Bilgisayar, genel amaçlı yazmaçlar, vb.) kaydedilmesi ve geri yüklenmesi nispeten hızlı olsa da, belleğin tüm içeriğini diske kaydetmek performansızdır. Bu nedenle, işlemleri aralarında geçiş yaparken bellekte bırakmak, işletim sisteminin zaman paylaşımını verimli bir şekilde uygulamasına izin verir ve (Sayfa 3, Şekil 13.2'de gösterildiği gibi) yapmayı tercih ederiz.

Diagramda üç işlem (A, B ve C) vardır ve her biri kendileri için oyulmuş 512 KB'lık fiziksel belleğin küçük bir kısmına sahiptir. Tek bir CPU varsayarsak, işletim sistemi süreçlerden birini (örneğin A) çalıştırmayı seçerken, diğerleri (B ve C) hazır kuyruğunda çalışmayı bekler. Zaman paylaşımı daha popüler hale geldikçe, muhtemelen işletim sistemine yeni talepler getirildiğini tahmin edebilirsiniz. Özellikle birden çok programın aynı anda bellekte bulunmasına izin verilmesi, **korumayı(protection)** önemli bir sorun haline getirir; okumak için bir süreç olmasını istemiyorsanız veya daha da kötüsü olmaması için, başka bir işlemin hafızasını yazın.



Şekil 13.2: Üç işlem: Belleği Paylaşmak

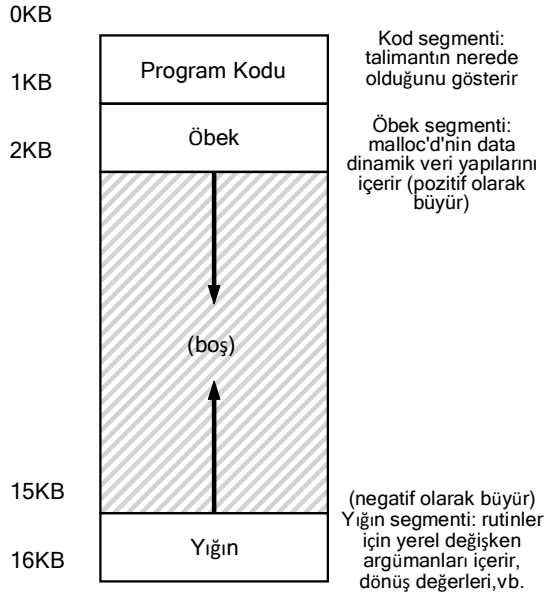
13.2 Adres Yolu

Ancak, bu can sıkıcı kullanıcıları aklımızda tutmalıyız ve bunu yapmak için işletim sisteminin **kullanımı kolay(easy to use)** bir fiziksel bellek soyutlaması oluşturması gerekir. Bu soyutlamaya **adres alanı(address space)** diyoruz ve çalışan programın sistemdeki belleğe yansımadır. Belleğin temel soyutlamasını anlamak, belleğin nasıl sanallaştırıldığını anlamanın anahtarıdır. Bir işlemin adres alanı, çalışan programın tüm bellek durumunu içerir. Örneğin, programın **kodu(code)** (talimatlar)

bellekte bir yerde bulunmak zorundadır ve bu nedenle adres alanındadır. Program çalışırken, işlev çağrı zincirinde nerede olduğunu takip etmek ve ayrıca yerel değişkenleri tahsis etmek, rutinlere ve rutinlerden parametreleri ve dönüş değerlerini iletmek için bir **yığın(stack)** kullanır. Son olarak, **öbek(heap)** dinamik olarak ayrılmış, kullanıcı tarafından yönetilen bellek için kullanılır; örneğin, C'deki bir malloc() çağrısından veya C++ ya da Java gibi nesne yönelimli bir dilde new komutu ile alabileceğiniz gibi orada başka şeyler de var (örneğin, statik olarak başlatılmış değişkenler), ama şimdilik sadece

bu üç bileşeni varsayalım: kod, yığın, öbek. Şekil 13.3'teki örnekte (sayfa 4), küçük bir adres alanımız var (yalnızca 16 KB)¹.

¹Sıklıkla bunun gibi küçük örnekler kullanacağız çünkü (a) 32 bitlik bir adres alanını temsil etmek zahmetlidir ve (b) matematik daha zordur. Basit matematiği severiz.



Şekil 13.3: Adres alanının bir örneği

Program kodu, adres alanının en üstünde yer alır. (bu örnekte 0'dan başlayarak ve adres alanının ilk 1K'lık kısmına sıkıştırılmıştır). Kod statiktir (ve bu nedenle belleğe yerleştirilmesi kolaydır) bu nedenle onu adres alanının en üstüne yerleştirebilir ve program çalışırken daha fazla alana ihtiyaç duymayacağını bilebiliriz.

Sonrasında, program çalışırken büyüyen (ve küçülebilen) adres alanının iki bölgesine sahibiz. Öbek en üstte and yığın da altta olacak şekilde yerleşmelidir. Onları bu şekilde yerleştiriyoruz çünkü her biri büyüyebilmek istiyor ve onları adres uzayının zıt uçlarına koyarak böyle bir büyümeye izin verebiliriz: sadece zıt yönlerde büyümeleri gerekiyor.

Böylece yığın, koddan hemen sonra başlar (1 KB'de) ve aşağı doğru büyür (kullanıcı malloc() aracılığıyla daha fazla bellek istediğinde); yığın 16 KB'de başlar ve yukarı doğru büyür (bir kullanıcı prosedür çağırısı yaptığında söyler). Ancak, yığının ve öbeğin bu yerleşimi yalnızca bir kuraldır; isterseniz adres alanını farklı bir şekilde düzenleyebilirsiniz (daha sonra göreceğimiz gibi, bir adres alanında birden fazla **iş parçacığı(threads)** bir arada bulunduğu anda, adres alanını bu şekilde bölmenin ne yazık ki iyi bir yolu yok).

Elbette adres alanını tarif ettiğimizde, tarif ettiğimiz şey işletim sisteminin çalışan programa sağladığı **soyutlamadır(abstraction)**. Program, 0 ile 16KB arasındaki fiziksel adreslerde gerçekten bellekte değildir; bunun yerine bazı rasgele fiziksel adres(ler)e yüklenir. Şekil 13.2'deki A, B ve C süreçlerini inceleyin; orada her işlemin farklı bir adreste belleğe nasıl yüklendiğini görebilirsiniz.

En önemli nokta: Bellek nasıl sanallaştırılır?

İşletim sistemi, birden çok çalışan işlem (tümü paylaşım belleği) için tek bir fiziksel belleğin üzerinde özel, potansiyel olarak büyük bir adres alanının bu soyutlamasını nasıl oluşturabilir?

Sorun da burada başlıyor. İşletim sistemi bunu yaptığında, işletim sisteminin **belleği sanallaştırdığını(virtualizing memory)** söylüyoruz, çünkü çalışan program belleğe belirli bir adreste(0 diyelim) yüklendiğini düşünüyor ve potansiyel olarak çok büyük bir adres alanına sahip(32bit veya 64-bit diyelim); fakat gerçeklik oldukça farklıdır.

Örneğin, Şekil 13.2'deki A işlemi, 0 adresinde (biz buna **sanal adres(virtual address)** diyeceğiz) bir yük gerçekleştirmeye çalıştığında, bir şekilde işletim sistemi, bazı donanım desteğiyle birlikte, yükün gerçekten yüklenmediğinden emin olmak zorunda kalacaktır. fiziksel adres 0'a değil, fiziksel adres 320KB'ye (A'nın belleğe yüklendiği yer) gider. Bu, dünyadaki her modern bilgisayar sisteminin temelini oluşturan belleği sanallaştırmanın anahtarıdır.

13.3 Hedefler

Böylece, bu not dizisinde işletim sisteminin yaptığı işe geliyoruz: belleği sanallaştırmak. Ancak işletim sistemi yalnızca belleği sanallaştırmaz. İşletim sisteminin bunu yaptığından emin olmak için bize rehberlik edecek bazı hedeflere ihtiyacımız vardır. Bu hedefleri daha önce gördük (Girişteki cümleleri düşünün) ve tekrar göreceğiz, ancak tekrar etmeye değer olduğu için tekrardan söylemede sakınca yoktur.

Bir sanal bellek(SB) sisteminin ana hedeflerinden biri **şeffaflıktır(transparency)**². İşletim sistemi, sanal belleği çalışan programa görünmeyecek şekilde uygulamalıdır. Bu nedenle program, belleğin sanallaştırıldığının farkında olmamalıdır; bunun yerine program kendi özel fiziksel belleğine sahipmiş gibi davranır. Perde arkasında, işletim sistemi (ve donanım) belleği birçok farklı iş arasında çoğullamak için tüm işi yapar ve bu nedenle yanılmasını gerçekleştirir.

Sanal makinenin bir diğer amacı da **verimlilik(efficiency)**. İşletim Sistemi, hem zaman (yani, programları çok daha yavaş çalıştırmamak için) hem de alan (yani, sanallaştırmayı desteklemek için gereken yapılar için çok fazla bellek kullanmaması lazım) açısından sanallaştırmayı mümkün olduğunca **verimli(efficient)** hale getirmeye çalışmalıdır. Zaman açısından verimli sanallaştırmayı uygularken, İşletim Sistemi, Çeviri görünümü arabellek gibi (ileride öğreneceğimiz) donanım özellikleri de dahil olmak üzere donanım desteğine güvenmek zorunda kalacaktır.

²Bu şeffaflık kullanımı bazen kafa karıştırıcıdır; bazı öğrenciler "şeffaf olmanın" her şeyi açıkta tutmak anlamına geldiğini düşünüyor. Burada bunun tam tersi: İşletim sistemi tarafından sağlanan yanılmanın uygulamalar tarafından görülmemesi gerektiği anlamına gelir.

İpucu: İzolasyon Prensibi

İzolasyon, güvenilir sistemler oluşturmak için temel bir ilkedir. İki varlık birbirinden düzgün bir şekilde izole edilirse bu, birinin diğerini etkilemeden başarısız olabileceği anlamına gelir. İşletim sistemleri, süreçleri birbirinden izole etmeye ve bu şekilde birinin diğerine zarar vermesini engellemeye çalışır. İşletim sistemi, bellek yalıtımını kullanarak ayrıca çalışan programların temeldeki işletim sisteminin çalışmasını etkilememesini sağlar. Bazı modern işletim sistemleri, işletim sisteminin parçalarını işletim sisteminin diğer parçalarından ayırarak izolasyonu daha da ileri götürür. Bu tür **mikro çekirdekler(mikrokernels)** [BH70, R+89, S+03] bu nedenle tipik tek parça şeklindeki çekirdek tasarımlarından daha fazla güvenilirlik sağlayabilir.

Son olarak, sanal makinenin üçüncü hedefi korumadır. İşletim sistemi, işlemleri birbirinden ve işletim sisteminin kendisini işlemlerden koruduğundan emin olmalıdır. Bir işlem bir yükleme, depolama veya talimat getirme işlemi gerçekleştirdiğinde, başka bir işlemin veya işletim sisteminin kendisinin (yani, adres alanı dışındaki herhangi bir şeyin) bellek içeriğine erişmemeli veya bunları hiçbir şekilde etkilememelidir. Böylece koruma, süreçler arasında **yalıtım(isolation)** özelliğini sunmamızı sağlar; her süreç, diğer hatalı ve hatta kötü niyetli süreçlerin tahribatından uzak bir şekilde çalışmalıdır.

Sonraki bölümlerde, donanım ve işletim sistemleri desteği de dahil olmak üzere, belleği sanallaştırmak için gereken temel **mekanizmalara(mechanisms)** odaklanacağız. Ayrıca, boş alanın nasıl yönetileceği ve alanınız azaldığında hangi sayfaların bellekten atılacağı da dahil olmak üzere, işletim sistemlerinde karşılaşacağınız daha ilgili **politikalar(polices)** bazılarını da araştıracağız. Bunu yaparken, modern bir sanal bellek sistemi gerçekten nasıl çalışıyor bunu anlayacağız³.

Özet

Büyük bir işletim sisteminin alt sisteminin tanıtıldığını gördük: buna sanal bellek deriz. SB sistemi, tüm talimatlarını ve verilerini burada tutan programlara geniş, seyrek, özel bir adres alanı yanılması sağlamaktan sorumludur. İşletim sistemi, bazı ciddi donanım yardımı ile, bu sanal bellek referanslarının her birini alacak ve bunları, istenen bilgileri almak için fiziksel belleğe sunulabilecek fiziksel adreslere dönüştürecektir. İşletim sistemi, programları birbirinden korumanın yanı sıra işletim sistemini de koruyarak bunu birçok işlem için aynı anda yapacaktır. Tüm yaklaşım, çalışmak için bazı kritik politikaların yanı sıra çok sayıda mekanizma (çok sayıda düşük seviyeli makine) gerektirir; önce kritik mekanizmaları açıklayarak aşağıdan yukarıya başlayacağız. Ve böyle devam edeceğiz!

³Ya da sizi kursu bırakmaya ikna edeceğiz. Ama bekleyin; sanal makine aracılığıyla yaparsanız, büyük ihtimalle bu işin sonuna kadar gidersiniz!

Ayrıca: Gördüğünüz her adres sanaldır

Hiç işaretçi yazdıran bir C programı yazdınız mı? Gördüğünüz değer (bazı büyük sayılar, genellikle onaltılık olarak yazdırılır), sanal bir adrestir. Programınızın kodunun nerede bulunduğunu hiç merak ettiniz mi? Bunu da yazdırabilirsiniz, eğer yazdırabilerseniz, bu aynı zamanda bir sanal adres olacaktır. Aslında, kullanıcı düzeyinde bir programın programcısı olarak görebileceğiniz herhangi bir adres sanal bir adrestir. Bu talimatların ve veri değerlerinin makinenin fiziksel belleğinin neresinde olduğunu bilen, belleği sanallaştırmaya yönelik kurnaz teknikleri sayesinde yalnızca işletim sistemidir. Bu yüzden asla unutmayın: Bir programda bir adres yazdırırsanız, bu sanal bir adrestir, her şeyin bellekte nasıl düzenlendiğine dair bir yanılsamadır; sadece işletim sistemi (ve donanım) gerçek gerçeği bilir.

İşte `main()` rutininin konumlarını (kodun olduğu yer), `malloc()`'tan döndürülen öbeği tahsisli bir değer değerini ve yığındaki bir tam sayının konumunu yazdıran küçük bir program (va.c):

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main(int argc, char *argv[]) {
4     printf("location of code : %p\n", main);
5     printf("location of heap : %p\n", malloc(100e6));
6     int x = 3;
7     printf("location of stack: %p\n", &x);
8     return x;
9 }
```

64 bitlik Mac'te çalıştırdığımızda aşağıdaki çıktıyı alırız:

kodun yeri:0x1095afe50

öbeğin yeri:0x1096008c0

yığının yeri:0x7fff691aea64

Buradan, kodun önce adres alanından, sonra öbekten geldiğinin ve yığının bu büyük sanal alanın diğer ucunda olduğunu görebilirsiniz. Bu adreslerin tümü sanaldır ve değerleri gerçek fiziksel konumlarından almak için işletim sistemi ve donanım tarafından çevrilmesi gerekir.

References

- [BH70] “The Nucleus of a Multiprogramming System” by Per Brinch Hansen. *Communications of the ACM*, 13:4, April 1970. *The first paper to suggest that the OS, or kernel, should be a minimal and flexible substrate for building customized operating systems; this theme is revisited throughout OS research history.*
- [CV65] “Introduction and Overview of the Multics System” by F. J. Corbato, V. A. Vyssotsky. Fall Joint Computer Conference, 1965. *A great early Multics paper. Here is the great quote about time sharing: “The impetus for time-sharing first arose from professional programmers because of their constant frustration in debugging programs at batch processing installations. Thus, the original goal was to time-share computers to allow simultaneous access by several persons while giving to each of them the illusion of having the whole machine at his disposal.”*
- [DV66] “Programming Semantics for Multiprogrammed Computations” by Jack B. Dennis, Earl C. Van Horn. *Communications of the ACM*, Volume 9, Number 3, March 1966. *An early paper (but not the first) on multiprogramming.*
- [L60] “Man-Computer Symbiosis” by J. C. R. Licklider. *IRE Transactions on Human Factors in Electronics*, HFE-1:1, March 1960. *A funky paper about how computers and people are going to enter into a symbiotic age; clearly well ahead of its time but a fascinating read nonetheless.*
- [M62] “Time-Sharing Computer Systems” by J. McCarthy. Management and the Computer of the Future, MIT Press, Cambridge, MA, 1962. *Probably McCarthy’s earliest recorded paper on time sharing. In another paper [M83], he claims to have been thinking of the idea since 1957. McCarthy left the systems area and went on to become a giant in Artificial Intelligence at Stanford, including the creation of the LISP programming language. See McCarthy’s home page for more info: <http://www-formal.stanford.edu/jmc/>*
- [M+63] “A Time-Sharing Debugging System for a Small Computer” by J. McCarthy, S. Boilen, E. Fredkin, J. C. R. Licklider. AFIPS ’63 (Spring), New York, NY, May 1963. *A great early example of a system that swapped program memory to the “drum” when the program wasn’t running, and then back into “core” memory when it was about to be run.*
- [M83] “Reminiscences on the History of Time Sharing” by John McCarthy. 1983. Available: <http://www-formal.stanford.edu/jmc/history/timesharing/timesharing.html>. *A terrific historical note on where the idea of time-sharing might have come from including some doubts towards those who cite Strachey’s work [S59] as the pioneering work in this area.*
- [NS07] “Valgrind: A Framework for Heavyweight Dynamic Binary Instrumentation” by N. Nethercote, J. Seward. PLDI 2007, San Diego, California, June 2007. *Valgrind is a lifesaver of a program for those who use unsafe languages like C. Read this paper to learn about its very cool binary instrumentation techniques – it’s really quite impressive.*
- [R+89] “Mach: A System Software kernel” by R. Rashid, D. Julin, D. Orr, R. Sanzi, R. Baron, A. Forin, D. Golub, M. Jones. COMPCON ’89, February 1989. *Although not the first project on microkernels per se, the Mach project at CMU was well-known and influential; it still lives today deep in the bowels of Mac OS X.*
- [S59] “Time Sharing in Large Fast Computers” by C. Strachey. *Proceedings of the International Conference on Information Processing, UNESCO*, June 1959. *One of the earliest references on time sharing.*
- [S+03] “Improving the Reliability of Commodity Operating Systems” by M. M. Swift, B. N. Bershad, H. M. Levy. SOSP ’03. *The first paper to show how microkernel-like thinking can improve operating system reliability.*

Ödev (Kod)

Bu ödevde, Linux tabanlı sistemlerde sanal bellek kullanımını incelemek için birkaç yararlı araç hakkında bilgi edineceğiz. Bu, neyin mümkün olduğuna dair yalnızca özet niteliğinde ipucu olacaktır; gerçekten bir uzman olmak için kendi başınıza daha derine dalmanız gerekecek (her zaman olduğu gibi!).

Sorular

1. Kontrol etmeniz gereken ilk Linux aracı olan `free`'dir ve çok basit bir araçtır. İlk olarak, `man free` yazın ve kılavuz sayfasının tamamını okuyun; kısa bir yazıdır merak etmeyin!

```

umut@umut-virtual-machine: ~
$ man free

```

İlk olarak `free`'nin ne anlama geldiğini araştırdım. `Free` komutunun, bir sistemin kullanılan ve kullanılmayan belleğin ve takas belleği(`swap`) hakkında bilgi verdiğini ve varsayılan olarak, belleği kb (kilobayt) cinsinden görüntülediğini, bellek esas olarak rastgele erişim belleği(RAM) ve takas belleğinden oluştuğunu öğrendim. Terminali açtım ve `man free` komutunu yazdım yukarıda gösterdiğim gibi. Yazdıktan sonra aşağıda ekran görüntülerini paylaştığım yazılar çıktı. Hepsini internette araştırdım sonra anlamaya çalışarak okudum.

```

free(1)
NAME
    free - Display amount of free and used memory in the system

SYNOPSIS
    free [options]

DESCRIPTION
    free displays the total amount of free and used physical and swap memory in the system, as well as the buffers and caches used by the kernel. The information is gathered by parsing /proc/meminfo. The displayed columns are:
    total Total installed memory (MemTotal in /proc/meminfo)
    used  Used memory (calculated as total - free - buffers - cache)
    free  Unused memory (MemFree and SwapFree in /proc/meminfo)
    shared Memory used (mostly) by tmpfs (Shmem in /proc/meminfo)
    buffers Memory used by kernel buffers (Buffers in /proc/meminfo)
    cache  Memory used by the page cache and slabs (Cached and SReclaimable in /proc/meminfo)
    buff/cache Sum of buffers and cache
    available Estimation of how much memory is available for starting new applications, without swapping. Unlike the data provided by the cache or free fields, this field takes into account page cache and also that not all reclaimable memory slabs will be reclaimed due to items being in use (MemAvailable in /proc/meminfo, available on kernels 3.2+, emulated on kernels 2.6.27+; otherwise the same as free)

OPTIONS
    -b, --bytes Display the amount of memory in bytes.
    -k, --kibibytes Display the amount of memory in kibibytes. This is the default.
    -m, --mebibytes Display the amount of memory in mebibytes.
    -g, --gibibytes Display the amount of memory in gibibytes.
    -t, --tebibytes Display the amount of memory in tebibytes.

```

`Man Free`'nin `Manual Free`'nin kısaltması olarak yazıldığını öğrendim. Yani kullanıcı özellikleri bir nevi kılavuz sayfası karşıma çıktı. İlk olarak isim başlığı çıktı altında da `free`-Sistemdeki boş ve kullanılan bellek miktarını görüntüleyin açıklamasını gördüm.

Onun altında `synopsis`(yani kısa bir özet) bloğu karşıma çıktı.

Bir sonraki blok `Tanım` bloğuydu. Altındaki açıklamada şu yazıyordu: `free`, çekirdek tarafından kullanılan arabellekler ve önbelleklerin yanı sıra sistemdeki toplam boş ve kullanılan fiziksel ve takas belleği(`swap`) miktarını görüntüler. Bilgiler `/proc/meminfo` ayrıştırılarak toplanır. Görüntülenen sütunlar da altında yazılmış durumdaydı.

OPERATING

SYSTEMS

[VERSION 1.01]

WWW.OSTEP.ORG

Toplam(total): Toplam yüklü bellek miktarını veriyor.

Kullanılan(used): Kullanılan bellek miktarını veriyor.

Boş alan(free): Kullanılmayan bellek miktarını veriyor.

Paylaşılmış(shared): Tmpfs tarafından kullanılan bellek anlamına geliyormuş. Tmpfs'nin ne olduğunu araştırdığımda ise, işletim sisteminde uygulanan geçici bir dosya depolama olduğunu ve verilerin kalıcı bir depolama aygıtı yerine geçici bellekte depolandığını öğrendim.

Arabellekler(buffers): Çekirdek(kernel) arabellekleri tarafından kullanılan bellektir.

Önbellek(cache): Sayfa önbelleği ve tabakalar tarafından kullanılan bellektir.

Arabellek/Önbellek(buff/cache): Arabellek ve önbelleğin toplamıdır.

Mevcut(available): Değiştirmeden yeni uygulamaları başlatmak için ne kadar bellek bulunduğunun tahminidir. Önbellek veya boş alanlar tarafından sağlanan verilerin aksine, bu alan sayfa önbelleğini ve ayrıca kullanımda olan ögeler nedeniyle tüm geri alınabilir bellek tabakalarının geri kazanılmayacağını dikkate alır.

Bu bilgilerinin altında özellikler adında bir blok gördüm. Burada aslında bellek boyutunun farklı türlerde göstermenin bizim elimizde olduğunu öğrendim.

free --bytes yazarsak byte şeklinde

free --kibi yazarsak kibibyte şeklinde (Hiçbir değişiklik yapmazsak bizim bellek değerlerimiz kibibyte şeklinde görünür)

free --mebi yazarsak mebibyte şeklinde

free --gibi yazarsak gibibyte şeklinde

free --tebi yazarsak tebibyte şeklinde

free --pebi yazarsak pebibyte şeklinde

free --kilo yazarsak kibibyte şeklinde

free --mega yazarsak megabyte şeklinde

free --giga yazarsak gigabyte şeklinde

free --tera yazarsak terabyte şeklinde

free --peta yazarsak petabyte şeklinde

```

[1] (C) Terminal Ara 12 15:39
umut@umut-virtual-machine: ~
--tebt Display the amount of memory in tebibytes.
--pebt Display the amount of memory in pebibytes.
--kktl Display the amount of memory in kilobytes. Implies --st.
--mekl Display the amount of memory in megabytes. Implies --st.
--gigl Display the amount of memory in gigabytes. Implies --st.
--terl Display the amount of memory in terabytes. Implies --st.
--petl Display the amount of memory in petabytes. Implies --st.
-h, --human
    Show all output fields automatically scaled to shortest three digit unit and display the units of print out. Following units are used.
        B = bytes
        Ki = kibibyte
        Mi = mebibyte
        Gi = gibibyte
        Ti = tebibyte
        Pi = pebibyte
    If unit is missing, and you have exbibyte of RAM or swap, the number is in tebibytes and columns might not be aligned with header.
-w, --wide
    Switch to the wide mode. The wide mode produces lines longer than 80 characters. In this mode buffers and cache are reported in two separate columns.
-c, --count count
    Display the result count times. Requires the -s option.
-l, --lohi
    Show detailed low and high memory statistics.
-s, --seconds delay
    Continuously display the result delay seconds apart. You may actually specify any floating point number for delay using either . or , for decimal point. usleep(3) is used for microsecond resolution delay times.
-si Use Ki, me, gi etc (power of 1000) instead of KiBi, meBi, giBi (power of 1024).
-t, --total
    Display a line showing the column totals.
--help Print help.
-V, --version
    Display version information.
Manual page free(1) line 47/49 88% (press h for help or q to quit)

```

-h, --human komutla da, Tüm çıktıları otomatik olarak en kısa üç basamaklı birime ölçeklenmiş olarak gösterir ve çıktı birimlerini görüntüler. Aşağıdaki birimleri kullanınız.

B = bytes

Ki = kibibyte

Mi = mebibyte

Gi = gibibyte

Ti = tebibyte

Pi = pebibyte

-w, --wide işlemiyle, Geniş moda geçirir. Geniş mod, daha uzun çizgiler üretir (80 karakterden fazla). Bu modda arabellekler ve önbellek iki ayrı sütunda raporlanır.

-c, --count count işlemi, sonuç sayım sürelerini gösterir.

-l, --lohi işlemi, düşük ve yüksek bellek istatistiklerini ayrıntılı olarak gösterir.

-s, --seconds delay işlemi, sonuç gecikmesini saniyeler arayla sürekli olarak görüntüleyebiliriz. Gecikme için herhangi bir kayan nokta sayısı belirtebiliriz. veya , ondalık nokta için kullanırız.

--si işlemi, kibi,mebi,gibi(1024'ün katı) değerler yerine kilo,mega,giga(1000'in katı) değerler kullanmak için kullanılır.

-t, --total işlemi, Kolon toplamalarının sayısını verir.

--help işlemleri terminal hakkında daha geniş bilgiye sahip olabiliriz. Kısaltmalarla alakalı bilgileri öğrenebiliriz. Ben yazdığımda bu satırlar çıktı.

```
umut@umut-virtual-machine: ~$ help
GNU bash, version 5.2.2(1)-release (x86_64-pc-linux-gnu)
These shell commands are defined internally. Type 'help' to see this list.
Type 'help name' to find out more about the function 'name'.
Use 'info bash' to find out more about the shell in general.
Use 'man -k' or 'info' to find out more about commands not in this list.

A star (*) next to a name means that the command is disabled.

job_spec [n]
[ ( (expression) )
  filename [arguments]
  ;
  [ arg... ]
  [ (expression) ]
  alias [-p] [name=value] ... ]
bg [job_spec ...]
bind [-lpsvSX] [-n keymap] [-f filename] [-q name] [-u name] [-r keyseq] [-x keyseq:shell-command]
break [n]
builtin [shell-builtin arg ...]
caller [expr]
case WORD in (PATTERN) [ (PATTERN)... ] COMMANDS ;;; esac
cd [-l|-P [-p]] [-q] [dir]
command [-pvn] Command [arg ...]
compgen [-abedfgksuv] [-o option] [-A action] [-C globpat] [-w wordlist] [-f function] [-C command]
complete [-abedfgksuv] [-p] [-dE] [-o option] [-A action] [-C globpat] [-w wordlist] [-f function]
compsort [-o option] [-DEI] [name ...]
compopt [-a]
coproc [NAME] command [redirections]
declare [-aAfFgIlrtux] [name=value] ... or declare -p [-aAfFIlrtux] [name ...]
dirs [-clw] [-v] [-P]
disown [-h] [-ar] [jobspec ... | pid ...]
echo [-neE] [arg ...]
enable [-a] [-dnps] [-f filename] [name ...]
eval [arg ...]
exec [-cl] [-a name] [command [argument ...]] [redirection ...]
exit [n]
export [-fn] [name=value] ... or export -p
false
fc [-e name] [-l-r] [first] [last] or fc -s (patrec) [command]
for NAME [in WORDS ... ] ; do COMMANDS ; done
for (( expr1; expr2; expr3 )) ; do COMMANDS ; done
function name ( [COMMANDS] ; ) or name () { [COMMANDS] ; }
getopts optstring name [arg ...]
hash [-lr] [-o pathname] [-dE] [name ...]
help [-dns] [pattern ...]
```

```
-V, --version
        Display version information.

FILES
        /proc/meminfo
        memory information

BUGS
        The value for the shared column is not available from kernels before 2.6.32 and is displayed as zero.

        Please send bug reports to
        (procp@freelists.org)

SEE ALSO
        ps(1), slabtop(1), top(1), vmstat(8).

procp-ng                                     2018-05-31                                     FREE(1)
Manual page: Free(1) Linux 62/109 (END) (press h for help or q to quit)
```

-V, --version işlemi, sürüm bilgilerini ekranda gösterir.

Böylelikle özellikler bloğu bitmiş oldu. En altta da Dosyalar, hatalar yeri var. Dosyalarda hafıza bilgisinin yeri görünüyor ve hata alırsak o hatayı göndereceğimiz bir web sitesi linki mevcut.

Ayrıca bakınız yerinde yazan ps(1), slabtop(1), top(1), vmstat(8) ifadelerini de yazdım.

```
umut@umut-virtual-machine: ~$ ps
  PID TTY          TIME CMD
 5101 pts/2    00:00:00 bash
 5108 pts/2    00:00:00 ps
umut@umut-virtual-machine: ~$
```

ps (process status) Şimdi çalışan işlemleri gösterir.

Burada süreç numarası(process id) ve teletypewriter(tty) ve CMD bilgileri çıktı.

```
umut@umut-virtual-machine: ~/Desktop
umut@umut-virtual-machine: ~/Desktop$ sudo slabtop
```

es Terminal

umut@umut-virtual-machine: ~/Desktop

umut@umut-virtual-machine: ~/Desktop

```

Active / Total Objects (% used) : 806955 / 842983 (95,7%)
Active / Total Slabs (% used)   : 34232 / 34232 (100,0%)
Active / Total Caches (% used)  : 121 / 169 (71,6%)
Active / Total Size (% used)    : 378634,47K / 389727,05K (97,2%)
Minimum / Average / Maximum Object : 0,01K / 0,46K / 8,00K

```

OBJS	ACTIVE	USE	OBJ	SIZE	SLABS	OBJ/SLAB	CACHE	SIZE	NAME
119340	119085	99%	0,10K	3060	39		12240K		buffer_head
100632	93279	92%	0,19K	4792	21		19168K		dentry
93330	88694	95%	0,05K	1098	85		4392K		ftrace_event_field
63072	62638	99%	0,12K	1971	32		7884K		kernfs_node_cache
50536	49953	98%	4,00K	6317	8		202144K		kmalloc-4k
40308	36157	89%	0,62K	3359	12		26872K		inode_cache
34580	34148	98%	0,20K	1820	19		7280K		vm_area_struct
31984	31621	98%	0,50K	1999	16		15992K		kmalloc-512
31434	31395	99%	1,16K	2418	13		38688K		ext4_inode_cache
29274	28748	98%	0,04K	287	102		1148K		ext4_extent_status
29056	28120	96%	0,06K	454	64		1816K		kmalloc-64
16896	16855	99%	0,06K	264	64		1056K		vmap_area
16832	15026	89%	0,06K	263	64		1052K		anon_vma_chain
15666	15661	99%	0,57K	1119	14		8952K		radix_tree_node
15104	14543	96%	0,02K	59	256		236K		kmalloc-16
12800	12304	96%	0,01K	25	512		100K		kmalloc-8
12160	10488	86%	0,03K	95	128		380K		kmalloc-32
11235	9659	85%	0,19K	535	21		2140K		kmalloc-192
11008	11008	100%	0,12K	344	32		1376K		kmalloc-128
9800	9627	98%	0,07K	175	56		700K		Acpi-Operand
9702	7565	77%	0,09K	231	42		924K		kmalloc-96
9350	7971	85%	0,02K	55	170		220K		lsm_file_cache
9282	8498	91%	0,10K	238	39		952K		anon_vma
9168	7956	86%	0,25K	573	16		2292K		filp
7648	7374	96%	0,25K	478	16		1912K		kmalloc-256
4462	4282	95%	0,09K	97	46		388K		trace_event_file
3979	3297	82%	0,70K	173	23		2768K		proc_inode_cache
3328	2543	76%	0,02K	13	256		52K		kmalloc-cg-16
3264	3147	96%	2,00K	204	16		6528K		kmalloc-2k
3136	3136	100%	0,06K	49	64		196K		kmalloc-rcl-64
2541	2431	95%	0,76K	121	21		1936K		shmem_inode_cache
2304	2170	94%	0,31K	192	12		768K		mnt_cache
2048	2048	100%	0,01K	4	512		16K		kmalloc-cg-8
1978	1978	100%	0,69K	86	23		1376K		squashfs_inode_cache
1664	1304	78%	0,12K	52	32		208K		pid
1616	1608	99%	1,00K	101	16		1616K		kmalloc-1k
1504	1504	100%	0,12K	47	32		188K		scsi_sense_cache
1463	1298	88%	0,81K	77	19		1232K		sock_inode_cache
1155	1155	100%	0,19K	55	21		220K		proc_dir_entry

Slabtop, gerçek zamanlı olarak ayrıntılı çekirdek tabakasının önbellek bilgilerini görüntüler. Listelenen sıralama ölçütlerinden birine göre sıralanmış en iyi önbelleklerin bir listesini görüntüler. Ayrıca döşeme katmanı bilgileriyle dolu yukarıda gösterilen istatistik başlığını gösterir.

```

[+] No Terminal
Anita 10/11
vmstat -m vmstat-machine - s top
vmstat-vmstat-machine - s top
top - 19:13:27 up 1:51, 1 user, load average: 0.11, 0.05, 0.01
Tasks: 285 total, 1 running, 284 sleeping, 0 stopped, 0 zombie
Cpu(s): 2.2 us, 3.6 sy, 0.0 ni, 94.2 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 3888.0 total, 3220.0 free, 156.0 used, 285.4 buff/cache
PiB Swap : 0 total, 0 swapin, 0 swpd, 0 used, 0 avail Mem

  PID USER      PP  NI    TST   RSS     SHR   SPCW   MEM% COMMAND
-----
4403 umut      20   0 522812 10540 40260 3 1.5 1.0 0:10.20 gnome-terminal
3093 umut      20   0 202200 62996 32400 3 1.2 1.0 0:00.00 vncviewer
1 root       0   0 108808 17564 8650 0 0.0 0.0 0:01.00 systemd
2 root       0   0 0 0 0 0 0.0 0.0 0:00.00 kthreadd
3 root       0 -20 0 0 0 1 0.0 0.0 0:00.00 rcu_gp
...
top - 19:13:27 up 1:51, 1 user, load average: 0.05, 0.01, 0.00
Tasks: 285 total, 3 running, 282 sleeping, 0 stopped, 0 zombie
Cpu(s): 3.1 us, 3.6 sy, 0.0 ni, 93.3 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 3888.0 total, 3470.0 free, 154.0 used, 276.0 buff/cache
PiB Swap : 0 total, 0 swapin, 0 swpd, 0 used, 0 avail Mem

  PID USER      PP  NI    TST   RSS     SHR   SPCW   MEM% COMMAND
-----
3670 umut      20   0 3094236 268924 127052 R 6.7 6.8 0:10.26 gnome-shell
4403 umut      20   0 581436 54240 40944 R 2.5 1.4 0:10.33 gnome-terminal
5099 umut      20   0 24528 4352 3500 R 0.4 0.1 0:00.45 top
1 root       20   0 108808 17564 8650 0 0.0 0.0 0:01.00 systemd
2 root       20   0 0 0 0 0 0.0 0.0 0:00.00 kthreadd
3 root       20 -20 0 0 0 1 0.0 0.0 0:00.00 rcu_gp
4 root       0 -20 0 0 0 1 0.0 0.0 0:00.00 rcu_par_gp
5 root       0 -20 0 0 0 1 0.0 0.0 0:00.00 netns
7 root       0 -20 0 0 0 0 1 0.0 0.0 0:00.00 kuorke/BH events_highpri
9 root       0 -20 0 0 0 0 1 0.0 0.0 0:00.73 kuorke/BH events_highpri
10 root      0 -20 0 0 0 1 0.0 0.0 0:00.00 mm_percpu_wq
11 root     20 -20 0 0 0 1 0.0 0.0 0:00.00 rcu_tasks_klhwad

```

Linux işlemlerini göstermek için top komutu kullanılır. Çalışan sistemin dinamik, gerçek zamanlı bir görünümünü sağlar. Genellikle, bu komut sistemin özet bilgilerini ve halihazırda Linux Çekirdeği tarafından yönetilen işlemlerin veya iş parçacıklarının(threads) listesini gösterir.

[illegible]

vmstat kodu sanal makinenin özelliklerini gösteriyor. Boş alanı(free), arabelleği(buff), ön belleği(cache) bilgilerini görebiliriz.

2. Şimdi, free komutunu çalıştırın, belki yararlı olabilecek bazı bağımsız değişkenleri kullanarak (örneğin, -m, bellek toplamalarını megabayt olarak görüntülemek için) çalıştırın. Sisteminizde ne kadar bellek var? Belleğin ne kadarı boş alan? Bu sayılar tahminlerinize uyuyor mu?

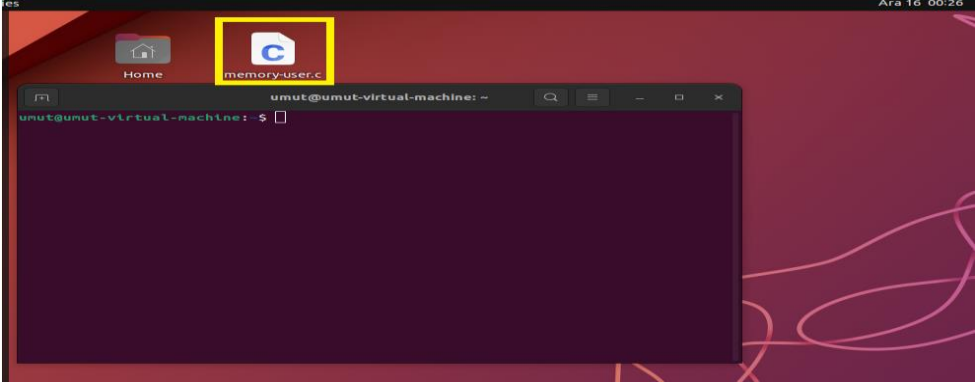
```

umut@umut-virtual-machine: ~/Desktop
umut@umut-virtual-machine: ~/Desktop
umut@umut-virtual-machine:~/Desktop$ free
              total        used        free      shared  buff/cache   available
Mem:           3982164      1759248       566388        44304       1656528       1926292
Swap:          3297276           0       3297276
umut@umut-virtual-machine:~/Desktop$ free -m
              total        used        free      shared  buff/cache   available
Mem:              4077         1801         579         45         1696         1972
Swap:             3376           0        3376
umut@umut-virtual-machine:~/Desktop$

```

İlk olarak free yazarak bilgilere ulaştım, ama bu veriler kibibyte olarak verilmişti. Sonra free --mega komutu ile verilerin megabyte olarak yazdırılmasını istedim. Yukarıdaki ekran görüntüsünde hem bellek(memory) hem de takas alanı(swap) hakkındaki bilgilere ulaşabiliyoruz. Ben sanal belleğin 4GB olmasını istemiştım(4.096MB) yukarıda hafızanın toplam 4077MB olduğu görünüyor istediğime çok yakın bir değer verdi. Belleğin 1801MB'si kullanılmış durumda. Bellekte 579MB boş alan olduğunu gösteriyor. 45MB paylaşılmış belleğe sahipmişim temporary filesystem(tmpfs)'in kullanabileceği yani kalıcı depolama yerine geçici dosya depolama alanı için kullanabileceğim 45MB'm olduğunu öğrendim. Arabellek/Önbellek oranının da 1696MB olduğu ve mevcut olarak kullanılabilecek 1972MB'min daha olduğu bilgisine eriştim. Takas alanıyla(swap) ilgili bilgiler de belleğin hemen altında verilmiş durumda. Toplam 3376MB alanım var. Şu anlık yüksek boyutta bir şey çalışmadığından takas alanına ihtiyaç yoktur. Kullanılan alan 0 olarak görünmesi normal bu yüzden de boş alan tam olarak toplamla aynı MB sayısına sahip.

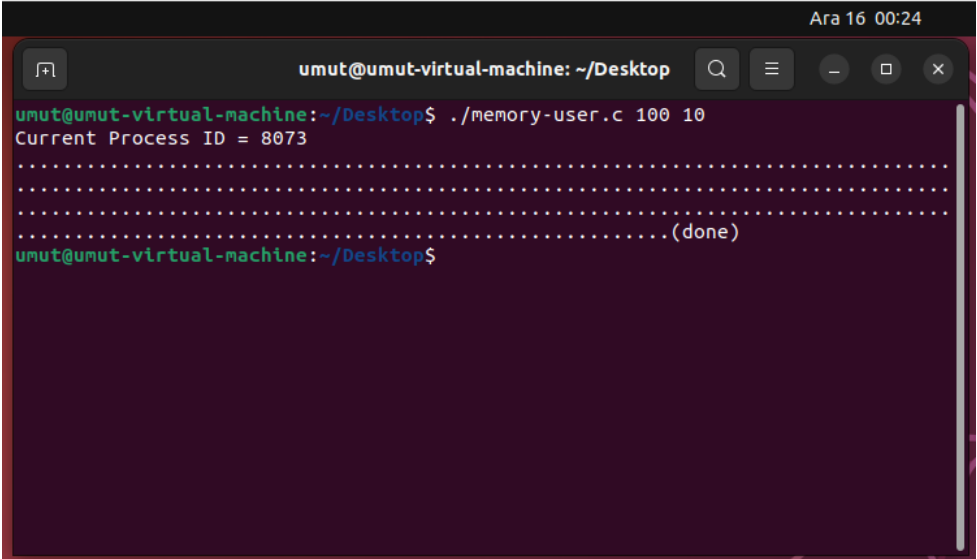
3. Ardından, memory-user.c adlı, belirli bir miktarda bellek kullanan küçük bir program oluşturun. Bu program bir komut satırı argümanı almalıdır:kullanılan megabayt bellek sayısını gösterebilir. Çalıştırıldığında, bir dizi tahsis etmeli ve her satıra yazarak dizi boyunca sürekli akış yapmalıdır. Program bunu süresiz olarak veya komut satırında belirtilen belirli bir süre boyunca yapmalıdır.



İlk olarak memory-user.c diye metin belgesi oluşturdum ve bunu masaüstüne kaydettim. İçini açıp benden istenen programın kodunu yazdım.



Kodu yazmaya #include<stdio.h>, #include<stdlib.h>, #include<time.h> #include<unistd.h>, #include<sys/types.h> yazarak gerekli kütüphaneleri ekleyerek başladım. Argc parametresi, girilecek parametre sayısını temsil ediyor. Eğer parametre sayısı 2 değilse Doğru sayıda argüment girilmedi hatasını veriyor. 2 değer yazarsak da if bloğuna girmeden koda devam ediyor. Argv(1)'i memory_usage'ye atadım. Uzunluk diye bir değer oluşturdum ve bu uzunluk denilen ifadenin karşılığı memory_usage/sizeof olarak yazdım. Counter değerini 0'dan başlattım. Clock değerini başlattım. Sonsuz döngü içinde time_spent diye bir değişken ekledim ve bunu clock()-begin/Clocks_per_sec şeklinde atadım. Eğer süre 30 saniyeyi geçerse kod break olsun diye de bir kod ekledim. Sonra for döngüsüne girerek arr değerini 1 arttırdım ve bu artan değer belleğe yansımaları için de free(arr); komutu ile de değişimin free'ye aktarılmasını sağladım. Return komutuyla da kodu bitirdim.



```
umut@umut-virtual-machine: ~/Desktop
umut@umut-virtual-machine:~/Desktop$ ./memory-user.c 100 10
Current Process ID = 8073
.....
.....
.....(done)
umut@umut-virtual-machine:~/Desktop$
```

Soruda da belirttiği gibi sorunun çözümü tek parametreyle ya da iki parametreyle yapılabilirdi. Metin belgesinin adı memory-user.c olduğu için belgedeki kodu çalıştırmam gerekiyordu onun için ./memory-user.c şeklinde yazmam gerekiyordu. Ben iki parametreyle yapmayı seçtim yani zaman değerini de kendim ekledim. Yukarıda görüldüğü gibi ilk değer (100 olan değer) boyutu ikinci değer de (10) süreyi gösteriyor. Bu kodu yazdığım zaman noktalar soldan başlayarak yazdığım süre boyunca yani 10 saniye dolana kadar ekranda nokta şeklinde yer kapladı.

4. Şimdi, bellek kullanıcı programınızı çalıştırırken, aynı zamanda (farklı bir terminal penceresinde, ancak aynı makinede) boş aracı çalıştırın. Programınız çalışırken bellek kullanım toplamaları nasıl değişir? Bellek kullanıcı programını sonlandırdığınızda ne olur? Rakamlar beklentilerinizle örtüşüyor mu? Farklı bellek kullanımı miktarları için bunu deneyin. Gerçekten büyük miktarda bellek kullandığınızda ne olur?

```

umut@umut-virtual-machine: ~/Desktop
umut@umut-virtual-machine:~/Desktop$ ./memory-user.c 100 5
Current Process ID = 8661
.....
.....(done)
umut@umut-virtual-machine:~/Desktop$

umut@umut-virtual-machine: $ free --mega
total        used        free      shared  buff/cache   available
Mem:      4077      1192      2766         3        139      2711
Swap:    3376         576      2799

umut@umut-virtual-machine: $ free --mega
total        used        free      shared  buff/cache   available
Mem:      4077      1192      2745         3        139      2691
Swap:    3376         575      2886
umut@umut-virtual-machine: $

```

İlk olarak solda kodu yazacağım terminali ve bellek özelliklerini görebileceğim terminali de sağda açtım. Sağdaki terminalde belleğin ilk değerlerini kaydettim sonra ./memory-user.c 100 5 kodu ile 100 değerini denedim ve 5 saniye boyunca ekrana nokta koyarak ilerledi ve yan pencerede tekrardan bellek değerlerinin nasıl değiştiğini görmek için free komutunu yazdım. Bellekte değişen değerler arasında kullanılan bellek miktarı vardı 1172MB iken, 1192MB değerine yükselmişti. Kullanılmayan bellek miktarında da azalış olduğunu gözlemladim 2766MB'den, 2745MB değerine düşmüştü. Mevcut bellek değeri de 2711MB'den 2691MB'ye düşüş göstermişti.

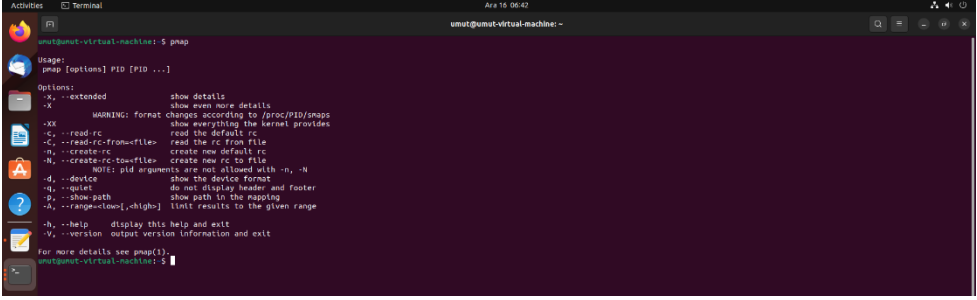
```

umut@umut-virtual-machine: ~/Desktop
umut@umut-virtual-machine:~/Desktop$ ./memory-user.c 10000 1
Current Process ID = 9110
.Segmentation fault (core dumped)
umut@umut-virtual-machine:~/Desktop$

```

Bellek kullanım miktarını yüksek bir değer yapınca da örneğin burada 10000 değerini seçtim. Segmentation fault (core dumped) hatasını verdi ve bu yüzden bellek özellikleri değişmedi.

5. Pmap olarak bilinen bir araç daha deneyelim. Anlamak için biraz zaman ayırın ve pmap kılavuz sayfasını ayrıntılı olarak okuyun.



```

umut@umut-virtual-machine:~$ pmap
Usage:
pmap [options] PID [PID ...]

Options:
-x, --extended          show details
-X                      show even more details
-XX                     WARNING: format changes according to /proc/PID/maps
                        show everything the kernel provides
-C, --read-rc            read the rc from file
-c, --read-rc-from-file  read the rc from file
-n, --create-rc          create new default rc
-N, --create-rc-to-file  create new rc to file
                        NOTE: pid arguments are not allowed with -n, -N
-d, --device            show the device format
-q, --quiet             do not display header and footer
-p, --show-path          show path to the mapping
-A, --range=<low>,<high> limit results to the given range

-h, --help              display this help and exit
-V, --version            output version information and exit

For more details see pmap(1).
umut@umut-virtual-machine:~$

```

Linux'ta pmap komutu, bir işlemin bellek haritasını görüntülemek için kullanılır. Bir hafıza haritası, hafızanın nasıl dağıldığını gösterir. Pmap yazdıktan sonra çıkan komutları yanındaki açıklamalarıyla birlikte okudum.

pmap -x PID komutu ile bilgileri genişletilmiş biçimde görüntüleriz.
 pmap -XX PID komutu, çekirdeğin bize sağladığı her türlü bilgiyi görmek için kullanılır.
 pmap -c PID komutu, varsayılan yapılandırmayı okumak için kullanılır.
 pmap -n komutu, yeni konfigürasyon oluşturmak için kullanılır.

pmap -d PID komutu ile ekstra bilgiler de görünür.
 pmap -q PID komutu üst bilgi ve alt bilgi satırını göstermez.
 pmap -p PID komutu eşleme sütunundaki dosyaların tam yolunu gösterir.
 pmap -A komutu ile verilen aralıktaki sonuçları görüntülemek için kullanılır.

pmap -h komutu, yardım metnini görüntülemek için kullanılır.
 pmap -V komutu sistemin kullandığı pmap sürümünü gösterir.

6. Mmap'i kullanmak için, üzerinde çalıştığınız işlemin işlem kimliğini bilmeniz gerekir. Bu nedenle, tüm işlemlerin bir listesini görmek için önce ps auxw'yi çalıştırın; ardından tarayıcı gibi ilginç bir tane seçin. Bu durumda bellek kullanıcı programınızı da kullanabilirsiniz (aslında, bu programın getpid()'i çağırmasını ve size kolaylık sağlamak için PID'ını yazdırmasını bile sağlayabilirsiniz).

ps auxw komutunu yazdım ve önüme çok uzun ve birçok değişkene sahip olan değerler çıktı. İlk önce çıkanların ekran görüntülerini koydum, açıklamayı altında yapacağım.

```

umut@umut-virtual-machine: ~
umut@umut-virtual-machine: ~$ ps auxw
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.2 169344 8176 ?        Ss   Ara15   0:09 /sbin/init auto noprompt splash
root         2  0.0  0.0      0   0 ?        Ss   Ara15   0:00 [kthreadd]
root         3  0.0  0.0      0   0 ?        I=   Ara15   0:00 [rcu_gp]
root         4  0.0  0.0      0   0 ?        I=   Ara15   0:00 [rcu_par_gp]
root         5  0.0  0.0      0   0 ?        I=   Ara15   0:00 [netns]
root         7  0.0  0.0      0   0 ?        I=   Ara15   0:00 [kworker/0:0H-events_highpri]
root         9  0.0  0.0      0   0 ?        I=   Ara15   0:04 [kworker/0:1H-events_highpri]
root        10  0.0  0.0      0   0 ?        I=   Ara15   0:00 [swapper/0]
root        11  0.0  0.0      0   0 ?        I=   Ara15   0:00 [rcu_tasks_kthread]
root        12  0.0  0.0      0   0 ?        I=   Ara15   0:00 [rcu_tasks_rude_kthread]
root        13  0.0  0.0      0   0 ?        I=   Ara15   0:00 [rcu_tasks_trace_kthread]
root        14  0.0  0.0      0   0 ?        S   Ara15   0:01 [ksoftirqd/0]
root        15  0.0  0.0      0   0 ?        S   Ara15   0:20 [rcu_preempt]
root        16  0.0  0.0      0   0 ?        S   Ara15   0:00 [migration/0]
root        17  0.0  0.0      0   0 ?        S   Ara15   0:00 [idle_inject/0]
root        19  0.0  0.0      0   0 ?        S   Ara15   0:00 [cpulp/0]
root        20  0.0  0.0      0   0 ?        S   Ara15   0:00 [cpulp/1]
root        21  0.0  0.0      0   0 ?        S   Ara15   0:00 [idle_inject/1]
root        22  0.0  0.0      0   0 ?        S   Ara15   0:02 [migration/1]
root        23  0.0  0.0      0   0 ?        S   Ara15   0:00 [ksoftirqd/1]
root        24  0.0  0.0      0   0 ?        S   Ara15   0:00 [kworker/1:0H-events_highpri]
root        25  0.0  0.0      0   0 ?        S   Ara15   0:00 [kdevtmpfs]
root        27  0.0  0.0      0   0 ?        I=   Ara15   0:00 [inet_frag_wq]
root        28  0.0  0.0      0   0 ?        S   Ara15   0:00 [kauditd]
root        30  0.0  0.0      0   0 ?        S   Ara15   0:00 [khungtaskd]
root        32  0.0  0.0      0   0 ?        S   Ara15   0:00 [oom_reaper]
root        34  0.0  0.0      0   0 ?        I=   Ara15   0:00 [writeback]
root        35  0.0  0.0      0   0 ?        S   Ara15   0:04 [kcompactd0]
root        36  0.0  0.0      0   0 ?        S   Ara15   0:00 [kcmd]
root        37  0.0  0.0      0   0 ?        SN   Ara15   0:00 [khugepaged]
root        38  0.0  0.0      0   0 ?        I=   Ara15   0:00 [kintegrityd]
root        39  0.0  0.0      0   0 ?        I=   Ara15   0:00 [kblockd]
root        40  0.0  0.0      0   0 ?        I=   Ara15   0:00 [blkcg_punt_bto]
root        42  0.0  0.0      0   0 ?        I=   Ara15   0:00 [tpm_dev_wq]
root        43  0.0  0.0      0   0 ?        S   Ara15   0:00 [ata_sff]
root        44  0.0  0.0      0   0 ?        I=   Ara15   0:00 [md]
root        45  0.0  0.0      0   0 ?        I=   Ara15   0:00 [edac-poller]
root        46  0.0  0.0      0   0 ?        I=   Ara15   0:00 [devfreq_wq]
root        47  0.0  0.0      0   0 ?        S   Ara15   0:00 [watchdogd]
root        48  0.0  0.0      0   0 ?        S   Ara15   0:03 [kworker/1:1H-events_highpri]
root        49  0.0  0.0      0   0 ?        I=   Ara15   0:25 [kswapd0]
root        50  0.0  0.0      0   0 ?        S   Ara15   0:00 [cryptfs-kthread]
root        52  0.0  0.0      0   0 ?        I=   Ara15   0:00 [kthrotld]
root        62  0.0  0.0      0   0 ?        S   Ara15   0:00 [irq/24-pctehp]
root        63  0.0  0.0      0   0 ?        S   Ara15   0:00 [irq/25-pctehp]
root        64  0.0  0.0      0   0 ?        S   Ara15   0:00 [irq/26-pctehp]
root        65  0.0  0.0      0   0 ?        S   Ara15   0:00 [irq/27-pctehp]
root        66  0.0  0.0      0   0 ?        S   Ara15   0:00 [irq/28-pctehp]
root        67  0.0  0.0      0   0 ?        S   Ara15   0:00 [irq/29-pctehp]
root        68  0.0  0.0      0   0 ?        S   Ara15   0:00 [irq/30-pctehp]
root        69  0.0  0.0      0   0 ?        S   Ara15   0:00 [irq/31-pctehp]
root        70  0.0  0.0      0   0 ?        S   Ara15   0:00 [irq/32-pctehp]
root        71  0.0  0.0      0   0 ?        S   Ara15   0:00 [irq/33-pctehp]
root        72  0.0  0.0      0   0 ?        S   Ara15   0:00 [irq/34-pctehp]
root        73  0.0  0.0      0   0 ?        S   Ara15   0:00 [irq/35-pctehp]
root        74  0.0  0.0      0   0 ?        S   Ara15   0:00 [irq/36-pctehp]
root        75  0.0  0.0      0   0 ?        S   Ara15   0:00 [irq/37-pctehp]
root        76  0.0  0.0      0   0 ?        S   Ara15   0:00 [irq/38-pctehp]
root        77  0.0  0.0      0   0 ?        S   Ara15   0:00 [irq/39-pctehp]
root        78  0.0  0.0      0   0 ?        S   Ara15   0:00 [irq/40-pctehp]
root        79  0.0  0.0      0   0 ?        S   Ara15   0:00 [irq/41-pctehp]
root        80  0.0  0.0      0   0 ?        S   Ara15   0:00 [irq/42-pctehp]
root        81  0.0  0.0      0   0 ?        S   Ara15   0:00 [irq/43-pctehp]
root        82  0.0  0.0      0   0 ?        S   Ara15   0:00 [irq/44-pctehp]
root        83  0.0  0.0      0   0 ?        S   Ara15   0:00 [irq/45-pctehp]
root        84  0.0  0.0      0   0 ?        S   Ara15   0:00 [irq/46-pctehp]
root        85  0.0  0.0      0   0 ?        S   Ara15   0:00 [irq/47-pctehp]
root        86  0.0  0.0      0   0 ?        S   Ara15   0:00 [irq/48-pctehp]
root        87  0.0  0.0      0   0 ?        S   Ara15   0:00 [irq/49-pctehp]
root        88  0.0  0.0      0   0 ?        S   Ara15   0:00 [irq/50-pctehp]
root        89  0.0  0.0      0   0 ?        S   Ara15   0:00 [irq/51-pctehp]
root        90  0.0  0.0      0   0 ?        S   Ara15   0:00 [irq/52-pctehp]
root        91  0.0  0.0      0   0 ?        S   Ara15   0:00 [irq/53-pctehp]
root        92  0.0  0.0      0   0 ?        S   Ara15   0:00 [irq/54-pctehp]
root        93  0.0  0.0      0   0 ?        S   Ara15   0:00 [irq/55-pctehp]
root        94  0.0  0.0      0   0 ?        I=   Ara15   0:00 [acpi_thermal_pm]
root        95  0.0  0.0      0   0 ?        S   Ara15   0:00 [xenbus_probe]
root        96  0.0  0.0      0   0 ?        S   Ara15   0:00 [scsi_eh_0]
root        97  0.0  0.0      0   0 ?        I=   Ara15   0:00 [scsi_tm_0]
root        98  0.0  0.0      0   0 ?        S   Ara15   0:00 [scsi_eh_1]
root        99  0.0  0.0      0   0 ?        I=   Ara15   0:00 [scsi_tm_1]
root       100  0.0  0.0      0   0 ?        I=   Ara15   0:00 [vfto-irqfd-clea]
root       101  0.0  0.0      0   0 ?        S   Ara15   0:00 [md]
root       102  0.0  0.0      0   0 ?        I=   Ara15   0:00 [lpv6_addrconf]
root       107  0.0  0.0      0   0 ?        I=   Ara15   0:00 [kstrtp]
root       113  0.0  0.0      0   0 ?        S   Ara15   0:00 [zswap-shrink]
root       114  0.0  0.0      0   0 ?        I=   Ara15   0:00 [kworker/u257:0]
root       166  0.0  0.0      0   0 ?        S   Ara15   0:00 [charger-manager]
root       205  0.0  0.0      0   0 ?        I=   Ara15   0:00 [mpt_poll_0]
root       266  0.0  0.0      0   0 ?        I=   Ara15   0:00 [mpt/0]
root       267  0.0  0.0      0   0 ?        S   Ara15   0:00 [scsi_eh_2]

```

Terminal Ara 16 08:24

umut@umut-virtual-machine: ~

root	207	0.0	0.0	0	0	7	S	Ara15	0:00	[scsl_eh_2]
root	208	0.0	0.0	0	0	7	I<	Ara15	0:00	[scsl_tnf_2]
root	209	0.0	0.0	0	0	7	S	Ara15	0:00	[scsl_eh_3]
root	210	0.0	0.0	0	0	7	I<	Ara15	0:00	[scsl_tnf_3]
root	211	0.0	0.0	0	0	7	S	Ara15	0:00	[scsl_eh_4]
root	212	0.0	0.0	0	0	7	I<	Ara15	0:00	[scsl_tnf_4]
root	213	0.0	0.0	0	0	7	S	Ara15	0:00	[scsl_eh_5]
root	214	0.0	0.0	0	0	7	I<	Ara15	0:00	[scsl_tnf_5]
root	215	0.0	0.0	0	0	7	S	Ara15	0:00	[scsl_eh_6]
root	216	0.0	0.0	0	0	7	I<	Ara15	0:00	[scsl_tnf_6]
root	217	0.0	0.0	0	0	7	S	Ara15	0:00	[scsl_eh_7]
root	218	0.0	0.0	0	0	7	I<	Ara15	0:00	[scsl_tnf_7]
root	219	0.0	0.0	0	0	7	S	Ara15	0:00	[scsl_eh_8]
root	220	0.0	0.0	0	0	7	I<	Ara15	0:00	[scsl_tnf_8]
root	221	0.0	0.0	0	0	7	S	Ara15	0:00	[scsl_eh_9]
root	222	0.0	0.0	0	0	7	I<	Ara15	0:00	[scsl_tnf_9]
root	223	0.0	0.0	0	0	7	S	Ara15	0:00	[scsl_eh_10]
root	224	0.0	0.0	0	0	7	I<	Ara15	0:00	[scsl_tnf_10]
root	225	0.0	0.0	0	0	7	S	Ara15	0:00	[scsl_eh_11]
root	226	0.0	0.0	0	0	7	I<	Ara15	0:00	[scsl_tnf_11]
root	227	0.0	0.0	0	0	7	S	Ara15	0:00	[scsl_eh_12]
root	228	0.0	0.0	0	0	7	I<	Ara15	0:00	[scsl_tnf_12]
root	229	0.0	0.0	0	0	7	S	Ara15	0:00	[scsl_eh_13]
root	230	0.0	0.0	0	0	7	I<	Ara15	0:00	[scsl_tnf_13]
root	231	0.0	0.0	0	0	7	S	Ara15	0:00	[scsl_eh_14]
root	232	0.0	0.0	0	0	7	I<	Ara15	0:00	[scsl_tnf_14]
root	233	0.0	0.0	0	0	7	S	Ara15	0:00	[scsl_eh_15]
root	234	0.0	0.0	0	0	7	I<	Ara15	0:00	[scsl_tnf_15]
root	235	0.0	0.0	0	0	7	S	Ara15	0:00	[scsl_eh_16]
root	236	0.0	0.0	0	0	7	I<	Ara15	0:00	[scsl_tnf_16]
root	237	0.0	0.0	0	0	7	S	Ara15	0:00	[scsl_eh_17]
root	238	0.0	0.0	0	0	7	I<	Ara15	0:00	[scsl_tnf_17]
root	239	0.0	0.0	0	0	7	S	Ara15	0:00	[scsl_eh_18]
root	240	0.0	0.0	0	0	7	I<	Ara15	0:00	[scsl_tnf_18]
root	241	0.0	0.0	0	0	7	S	Ara15	0:00	[scsl_eh_19]
root	242	0.0	0.0	0	0	7	I<	Ara15	0:00	[scsl_tnf_19]
root	243	0.0	0.0	0	0	7	S	Ara15	0:00	[scsl_eh_20]
root	244	0.0	0.0	0	0	7	I<	Ara15	0:00	[scsl_tnf_20]
root	245	0.0	0.0	0	0	7	S	Ara15	0:00	[scsl_eh_21]
root	246	0.0	0.0	0	0	7	I<	Ara15	0:00	[scsl_tnf_21]
root	247	0.0	0.0	0	0	7	S	Ara15	0:00	[scsl_eh_22]
root	248	0.0	0.0	0	0	7	I<	Ara15	0:00	[scsl_tnf_22]
root	249	0.0	0.0	0	0	7	S	Ara15	0:00	[scsl_eh_23]
root	250	0.0	0.0	0	0	7	I<	Ara15	0:00	[scsl_tnf_23]
root	251	0.0	0.0	0	0	7	S	Ara15	0:00	[scsl_eh_24]
root	252	0.0	0.0	0	0	7	I<	Ara15	0:00	[scsl_tnf_24]
root	253	0.0	0.0	0	0	7	S	Ara15	0:00	[scsl_eh_25]
root	254	0.0	0.0	0	0	7	I<	Ara15	0:00	[scsl_tnf_25]

Terminal Ara 16 08:26

umut@umut-virtual-machine: ~

root	254	0.0	0.0	0	0	7	S	Ara15	0:00	[scsl_eh_25]
root	255	0.0	0.0	0	0	7	I<	Ara15	0:00	[scsl_tnf_25]
root	256	0.0	0.0	0	0	7	S	Ara15	0:00	[scsl_eh_26]
root	257	0.0	0.0	0	0	7	I<	Ara15	0:00	[scsl_tnf_26]
root	258	0.0	0.0	0	0	7	S	Ara15	0:00	[scsl_eh_27]
root	259	0.0	0.0	0	0	7	I<	Ara15	0:00	[scsl_tnf_27]
root	260	0.0	0.0	0	0	7	S	Ara15	0:00	[scsl_eh_28]
root	261	0.0	0.0	0	0	7	I<	Ara15	0:00	[scsl_tnf_28]
root	262	0.0	0.0	0	0	7	S	Ara15	0:00	[scsl_eh_29]
root	263	0.0	0.0	0	0	7	I<	Ara15	0:00	[scsl_tnf_29]
root	264	0.0	0.0	0	0	7	S	Ara15	0:00	[scsl_eh_30]
root	265	0.0	0.0	0	0	7	I<	Ara15	0:00	[scsl_tnf_30]
root	267	0.0	0.0	0	0	7	S	Ara15	0:00	[scsl_eh_31]
root	268	0.0	0.0	0	0	7	I<	Ara15	0:00	[scsl_tnf_31]
root	298	0.0	0.0	0	0	7	S	Ara15	0:00	[scsl_eh_32]
root	299	0.0	0.0	0	0	7	I<	Ara15	0:00	[scsl_tnf_32]
root	325	0.0	0.0	0	0	7	S	Ara15	0:01	[h4d7ddab_4]
root	326	0.0	0.0	0	0	7	I<	Ara15	0:00	[ext4-rsv-conver]
root	366	0.0	0.0	63568	20752	?	Ss	Ara15	0:03	/lib/systemd/systemd-journald
root	398	0.0	0.0	152400	0	?	Ssl	Ara15	0:00	vmware-vmblock-fuse /run/vmblock-fuse -o rw,subtype=vmware-vmblock,default_permissions,allow_other,dev,suid
root	403	0.0	0.0	27900	12	?	Ss	Ara15	0:01	/lib/systemd/systemd-udev
root	404	0.0	0.0	0	0	?	S	Ara15	0:06	[lrq4-vmmfx]
root	405	0.0	0.0	0	0	?	S	Ara15	0:00	[card0-crtc0]
root	406	0.0	0.0	0	0	?	S	Ara15	0:00	[card0-crtc1]
root	407	0.0	0.0	0	0	?	S	Ara15	0:00	[card0-crtc2]
root	408	0.0	0.0	0	0	?	S	Ara15	0:00	[card0-crtc3]
root	409	0.0	0.0	0	0	?	S	Ara15	0:00	[card0-crtc4]
root	410	0.0	0.0	0	0	?	S	Ara15	0:00	[card0-crtc5]
root	411	0.0	0.0	0	0	?	S	Ara15	0:00	[card0-crtc6]
root	412	0.0	0.0	0	0	?	S	Ara15	0:00	[card0-crtc7]
systemd-	651	0.1	0.0	16004	528	?	Ss	Ara15	1:26	/lib/systemd/systemd-oond
systemd-	652	0.0	0.1	19340	4172	?	Ss	Ara15	0:02	/lib/systemd/systemd-resolved
systemd-	656	0.0	0.0	80496	492	?	Ssl	Ara15	0:00	/lib/systemd/systemd-timesyncd
root	670	0.0	0.0	65852	340	?	Ss	Ara15	0:00	/usr/bin/PGAuthService
root	672	0.1	0.0	255076	3884	?	Ssl	Ara15	1:41	/usr/bin/vmtoolsd
root	702	0.0	0.0	250092	336	?	Ssl	Ara15	0:02	/usr/libexec/accounts-daemon
avahi	771	0.0	0.0	8364	980	?	Ss	Ara15	0:00	avahi-daemon: running [umut-virtual-machine.local]
root	773	0.0	0.0	20772	724	?	Ss	Ara15	0:00	/usr/sbin/cron -f -P
messages	774	0.0	0.0	11808	3566	?	Ss	Ara15	0:03	dbus-daemon -system -addresssystemd: --nofork --nopidfile --systemd-activation --syslog-only
root	792	0.0	0.0	82792	468	?	Ss	Ara15	0:03	/usr/sbin/irqbalance --foreground
root	797	0.0	0.1	253884	6780	?	Ssl	Ara15	0:05	/usr/libexec/polkitd --no-debug
root	800	0.0	0.0	250856	256	?	Ssl	Ara15	0:00	/usr/libexec/power-profiles-daemon
syslog	808	0.0	0.0	222236	92	?	Ss	Ara15	0:00	/usr/sbin/rsyslogd -n -lNONE
root	819	0.0	0.5	948424	20740	?	Ssl	Ara15	0:13	/usr/lib/napd/napd
root	822	0.0	0.0	247312	584	?	Ssl	Ara15	0:00	/usr/libexec/switcheroo-control
root	829	0.0	0.0	16516	2628	?	Ss	Ara15	0:00	/lib/systemd/systemd-logind
root	835	0.0	0.0	466176	2788	?	Ssl	Ara15	0:00	/usr/libexec/udisks2/udisksd

```

$ terminal
Ar 16 08:28

umut@umut-virtual-machine: ~
root 835 0.0 0.0 466170 2788 ? Ssl ArA15 0:00 /usr/libexec/udisks2/udisksd
avahi 523 0.0 0.0 8176 16 ? S ArA15 0:00 avahi-daemon: chroot helper
root 868 0.0 0.0 316952 420 ? Ssl ArA15 0:00 /usr/sbin/NetworkManager --no-daemon
root 888 0.0 0.0 16494 0 ? Ss ArA15 0:00 /sbin/dmccs supplicant -s -5 -o /run/wpa_supplicant
root 948 0.0 0.0 123130 328 ? Ssl ArA15 0:00 /usr/bin/python3 /usr/share/unattended-upgrades/unattended-upgrade-shutdown --wait-for-signal
root 955 0.0 0.0 252020 508 ? Ssl ArA15 0:00 /usr/sbin/gdmg
kernoops 1027 0.0 0.0 12464 296 ? Ss ArA15 0:00 /usr/sbin/kerneloops --test
kernoops 1028 0.0 0.0 12464 296 ? Ss ArA15 0:00 /usr/sbin/kerneloops
root 1031 0.0 0.0 0 0 ? S ArA15 0:00 [cgroup]
rtkit 1170 0.0 0.0 88270 0 ? SNI ArA15 0:01 /usr/libexec/rtkit-daemon
root 1206 0.0 0.0 247092 244 ? Ssl ArA15 0:00 /usr/libexec/gnomev
root 1397 0.0 0.0 310032 2920 ? Ssl ArA15 0:00 /usr/libexec/packagekitd
root 1407 0.0 0.0 310032 2920 ? Ssl ArA15 0:00 /usr/libexec/colord
root 1909 0.0 0.0 179794 568 ? Sl ArA15 0:00 gdm-session-worker [pam/gdm-password]
umut 1942 0.0 0.1 70570 613 ? Ss ArA15 0:02 /lib/systemd/systemd --user
umut 1943 0.0 0.0 184364 48 ? S ArA15 0:00 [sd-pam]
umut 1973 0.0 0.2 143610 893 ? Ssl ArA15 0:07 /usr/bin/gnomev
umut 1976 0.0 0.1 20770 358 ? Ssl ArA15 0:01 /usr/bin/gnomev
umut 1982 0.0 0.1 4096 596 ? Ssl ArA15 0:01 /usr/bin/gnomev-pulse
umut 1988 0.0 0.0 18072 356 ? Ss ArA15 0:02 /usr/bin/bus-daemon --session --addresssystemd: --nofork --logfdfile --system-activation --syslog-only
umut 1995 0.0 0.0 253072 2340 ? Ssl ArA15 0:00 /usr/libexec/gnome-keyring-daemon --for-foreground --components=pkcs11,secrets --control-directory=/run/user/1000/keyring
umut 2003 0.0 0.0 255110 2508 ? Ssl ArA15 0:00 /usr/libexec/gvfsd
umut 2019 0.0 0.0 308364 212 ? Sl ArA15 0:00 /usr/libexec/gvfsd-fuse /run/user/1000/gvfs -f
umut 2042 0.0 0.0 547092 2476 ? Ssl ArA15 0:00 /usr/libexec/xdg-document-portal
umut 2050 0.0 0.0 240972 228 ? Ssl ArA15 0:00 /usr/libexec/xdg-permission-store
root 2060 0.0 0.0 5040 0 ? Ss ArA15 0:00 /usr/bin/xdg-desktop-portal,auto-umount,subtypes-portal -- /run/user/1000/doc
umut 2070 0.0 0.0 173232 408 tty2 Ssl ArA15 0:00 /usr/libexec/gdm-wayland-session env GNOME_SHELL_SESSION_MODE=ubuntu /usr/bin/gnome-session --session=ubuntu
umut 2103 0.0 0.0 274900 88 tty2 Sl ArA15 0:00 /usr/libexec/gnome-session-binary --session=ubuntu
umut 2103 0.0 0.0 638364 11496 ? SNI ArA15 0:04 /usr/libexec/tracker-nmfs-3
umut 2149 0.0 0.0 332024 3412 ? Ssl ArA15 0:00 /usr/libexec/gvfs-gsd-volume-monitor
umut 2172 0.0 0.0 88272 0 ? Ss ArA15 0:00 /usr/libexec/gvfs-smb-agent /run/user/1000/gcr
umut 2177 0.0 0.0 102692 0 ? Ssl ArA15 0:00 /usr/libexec/gnome-session-ccl --monitor
umut 2180 0.0 0.0 247456 2516 ? Ss ArA15 0:00 gdm-agent -o /run/user/1000/gdmsh_agent
umut 2185 0.0 0.0 247456 2516 ? Ssl ArA15 0:00 /usr/libexec/gvfs-goa-volume-monitor
umut 2191 0.0 0.0 638968 2602 ? Ssl ArA15 0:00 /usr/libexec/gnome-session-binary --systemd-service --session=ubuntu
umut 2204 0.0 0.0 332468 148 ? Sl ArA15 0:00 /usr/libexec/gnome-session-binary --systemd-service --session=ubuntu
umut 2206 0.0 0.0 240424 120 ? Ssl ArA15 0:00 /usr/libexec/gvfs-gphoto2-volume-monitor
umut 2217 0.0 0.0 24792 2650 ? Ssl ArA15 0:00 /usr/libexec/gvfs-np-volume-monitor
umut 2221 0.0 0.0 320088 2588 ? Ssl ArA15 0:04 /usr/libexec/gvfs-afc-volume-monitor
umut 2227 0.0 0.0 308 1876 ? Sl ArA15 0:00 /usr/libexec/at-spi-bus-launcher --launch-immediately
umut 2239 0.3 3.4 391648 13584 ? Ssl ArA15 3:57 /usr/bin/gnome-shell
umut 2247 0.0 0.0 638968 2602 ? Ssl ArA15 0:02 /usr/libexec/gdm-config-file=/usr/share/defaults/at-spi2/accessibility.conf --nofork --print-address 11 --address-unix-path=/run/
umut 2368 0.0 0.1 634588 4908 ? Ssl ArA15 0:02 /usr/libexec/gdm-desktop-portal
umut 2369 0.0 0.0 330870 3528 ? Ssl ArA15 0:00 /usr/libexec/gdm-desktop-portal
umut 2369 0.0 0.0 580294 392 ? Sl ArA15 0:00 /usr/libexec/gnome-shell-calendar-server
umut 2370 0.0 0.0 1173492 3136 ? Ssl ArA15 0:00 /usr/libexec/evolution-source-registry
umut 2391 0.0 0.1 117336 4396 ? Ssl ArA15 0:00 /usr/libexec/evolution-calendar-factory

```

```

$ terminal
Ar 16 08:29

umut@umut-virtual-machine: ~
umut 2391 0.0 0.1 1171256 4396 ? Ssl ArA15 0:00 /usr/libexec/evolution-calendar-factory
umut 2434 0.0 0.1 908072 5164 ? Ssl ArA15 0:00 /usr/libexec/evolution-addressbook-factory
umut 2435 0.0 0.0 314238 3664 ? Sl ArA15 0:00 /usr/libexec/gvfsd-trash --spawner 11.0 /org/gtk/gvfs/exec_spaw/0
umut 2450 0.0 0.0 273202 274 ? Ssl ArA15 0:00 /usr/libexec/gvfsd-smb-agent /run/user/1000/gcr
umut 2453 0.0 0.0 273794 408 ? Sl ArA15 0:00 /usr/bin/gjs /usr/share/gnome-shell/org.gnome.notifications
umut 2463 0.0 0.0 2736 0 ? Ss ArA15 0:00 sh -c /usr/bin/bus-daemon --panel-disable $( "XDG_SESSION_TYPE" = "x11" ] && echo "--x11"
umut 2474 0.0 0.0 332132 1900 ? Ssl ArA15 0:02 /usr/libexec/gdm-elyt-settings
umut 2474 0.0 0.0 332132 1900 ? Ssl ArA15 0:02 /usr/libexec/gdm-elyt-settings
umut 2483 0.0 0.0 308188 3124 ? Ssl ArA15 0:00 /usr/libexec/gsd-datetime
umut 2489 0.0 0.0 309456 370 ? Ssl ArA15 0:02 /usr/libexec/gsd-housekeeping
umut 2491 0.0 0.0 351408 3360 ? Ssl ArA15 0:00 /usr/libexec/gsd-keyboard
umut 2504 0.0 0.0 5131800 5812 ? Ssl ArA15 0:00 /usr/libexec/gsd-media-keys
umut 2506 0.0 0.0 240324 5272 ? Ssl ArA15 0:00 /usr/libexec/gsd-power
umut 2512 0.0 0.0 231680 0 ? Sl ArA15 0:00 /usr/libexec/gsd-disk-utility-notify
umut 2514 0.0 0.0 200808 0 ? Ssl ArA15 0:00 /usr/libexec/gsd-print-notifications
umut 2523 0.0 0.0 408684 2424 ? Ssl ArA15 0:00 /usr/libexec/gsd-remote-desktop
umut 2529 0.0 0.0 247108 1868 ? Ssl ArA15 0:00 /usr/libexec/gsd-screensaver-proxy
umut 2533 0.0 0.0 4166652 4072 ? Ssl ArA15 0:00 /usr/libexec/gsd-sharing
umut 2543 0.0 0.0 174292 532 ? Sl ArA15 0:00 /usr/libexec/ibus-memcat
umut 2546 0.0 0.0 330870 3528 ? Ssl ArA15 0:00 /usr/libexec/gsd-extension-gtk3
umut 2547 0.0 0.0 470600 2020 ? Ssl ArA15 0:00 /usr/libexec/gsd-smartcard
umut 2551 0.0 0.0 330884 1790 ? Ssl ArA15 0:00 /usr/libexec/gsd-sound
umut 2553 0.0 0.0 352080 3200 ? Ssl ArA15 0:00 /usr/libexec/gsd-wacom
umut 2564 0.0 0.0 248008 2400 ? Ssl ArA15 0:00 /usr/libexec/gsd-portal
umut 2571 0.0 0.0 307076 2092 ? Ssl ArA15 1:10 /usr/bin/portaloid in vmur -blacklist 3 --unpgrd 4
umut 2580 0.0 0.0 809324 1180 ? Sl ArA15 0:01 /usr/libexec/evolution-data-server/evolution-alarm-notify
umut 2626 0.0 0.0 353204 0 ? Sl ArA15 0:00 /usr/libexec/gsd-printer
umut 2646 0.0 0.0 173932 2400 ? Ssl ArA15 0:00 /usr/libexec/gvfsd-metadata
umut 2681 0.0 0.0 174424 2800 ? Sl ArA15 0:03 /usr/libexec/ibus-engine-single
umut 2731 0.0 0.0 273604 1664 ? S ArA15 0:00 /usr/bin/wayland -s -rootless --socket-access-core-auth /run/user/1000/.mutter-Xwaylandauth.UCOW -listen 4 -listen 5 -displayfd 6
umut 2742 0.0 0.0 2072396 610 ? Sl ArA15 0:00 /usr/bin/gjs /usr/share/gnome-shell/org.gnome.Screensaver
umut 2762 0.0 0.1 353584 5396 ? Ssl ArA15 0:00 /usr/libexec/xdg-desktop-portal-gtk
umut 2830 0.0 1.2 331616 49712 ? Ssl ArA15 0:11 gjs /usr/share/gnome-shell/extension/dingrastersoft.com/dmg.js & P /usr/share/gnome-shell/extension/dingrastersoft.com & @ 0.0 0.0
umut 2871 0.0 0.0 513700 4228 ? Ssl ArA15 0:00 /usr/libexec/gsd-xsettings
umut 2940 0.0 0.0 246404 1072 ? Sl ArA15 0:00 /usr/libexec/ibus
umut 3066 0.0 0.1 504024 12750 ? Sl ArA15 0:02 update-notifier
umut 3141 0.0 0.0 353584 2320 ? Ssl ArA15 0:00 /usr/libexec/cockpit-service
umut 3287 0.0 0.1 609894 6940 ? Ssl ArA15 1:54 /usr/libexec/gnome-terminal-server
umut 3359 0.0 0.0 723470 3112 ? Sl ArA15 0:01 /usr/bin/gnome-calendar --gapplication-service
umut 5185 0.0 0.0 506704 3672 ? Ssl ArA15 0:00 /usr/libexec/gnome --gapplication-service
umut 5279 0.0 0.0 335584 3710 ? Sl ArA15 0:00 /usr/libexec/gdm-recent --spawner 11.0 /org/gtk/gvfs/exec_spaw/1
umut 6037 0.0 0.0 330870 3528 ? Ssl ArA15 0:00 bash
umut 7181 0.0 0.0 22580 5220 pts/0 Ss ArA15 0:00 bash
umut 7347 0.0 0.0 70120 310 ? Ss ArA15 0:00 /snap/snapd-desktop-integration/49/usr/bin/snapd-desktop-integration
umut 7320 0.0 0.0 308080 1800 ? Sl ArA15 0:00 /snap/snapd-desktop-integration/49/usr/bin/snapd-desktop-integration
root 7802 0.0 0.0 48768 1800 ? Ss 00:00 0:00 /usr/sbin/cupsd -l
root 7803 0.0 0.0 351404 3672 ? Ss 00:00 0:00 /usr/bin/cups-browsed
umut 8885 0.0 0.1 22720 5988 pts/1 Ss 00:10 0:00 bash
umut 8985 0.0 0.1 22720 5988 pts/1 Ss 00:10 0:00 bash
umut 8995 0.0 0.1 1344092 132924 ? Sl 00:10 0:01 /usr/bin/gnome-text-editor --application-service
umut 9168 0.0 0.0 0 0 ? I 00:12 0:00 [worker/0:0 events]
root 9692 0.0 0.0 0 0 ? I 00:20 0:00 [worker/1:1 events]
root 9721 0.0 0.0 0 0 ? I 00:47 0:00 [worker/0:0:0 events_unbound]
root 9812 0.0 0.0 0 0 ? I 07:30 0:00 [worker/0:0:0 events_unbound]
umut 9729 0.0 0.0 22580 5220 pts/0 Ss 07:30 0:00 bash
umut 9934 0.0 0.1 22580 5840 pts/3 Ss 07:55 0:00 bash
umut 9956 0.0 0.0 0 0 ? I 00:07 0:00 [worker/1:1]
umut 9930 0.0 0.0 0 0 ? I 00:10 0:00 [worker/0:0:0 events_power_efficient]
umut 9980 0.0 0.1 22580 5812 pts/5 Ss 00:09 0:00 bash
umut 9967 0.0 0.0 0 0 ? I 00:09 0:00 [worker/0:0 events]
root 9988 0.0 0.0 0 0 ? R 00:17 0:00 [worker/0:0:0 events_unbound]
umut 10004 0.0 0.0 23880 3700 pts/5 R- 00:21 0:00 ps auxw

```

```

umut@umut-virtual-machine:~$ ps auxw
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.2 169344  8176 ?        Ss   Ara15    0:09 /sbin/init auto noprompt splash

```

Yukarıda örnek olarak verdiğim satırdaki değerleri açıkladım.

USER yani kullanıcı bilgisini veriyor bu root,umut,systemd+ veya syslog gibi başka kullanıcı olabilir.

PID(Process ID) işlem kimlik numarası yazıyor. PID özeldir ve her işlem için farklı bir değer alır.

%CPU: İşlemciyi yüzdelerik olarak ne kadar kullandığı yazıyor.

%MEM: Belleği yüzdelerik olarak ne kadar kullandığı yazıyor

VSZ(Virtual Memory Size): Sanal bellek boyutudur.Değiştirilen, tahsis edilen, ancak kullanılmayan ve paylaşılan kütüphanelerdeki bellek de dahil olmak üzere, işlemin erişebileceği tüm bellekleri içerir.

RSS(Resident Set Size): Bu işleme ne kadar bellek ayrıldığını ve RAM’de olduğunu göstermek için kullanılır. Takılan belleği içermez. Bu kütüphanelerdeki sayfalar gerçekte bellekte olduğu sürece, paylaşılan kütüphanelerdeki belleği ve tüm yığın, yığın belleğini içerir.

TTY(TeleTyPewriter): Standart girişe bağlanan dosyanın standart çıkışını yazan bir komuttur.

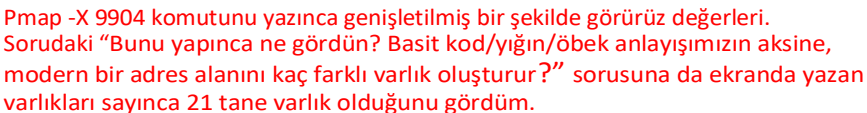
Stat: Dosya ve dosya sistemi hakkında bilgi veren bir komuttur.

Start: İşlemin ne zaman başladığını belirtir. Örneğin yukarıda Ara15 yazıyor yani aralık 15’te başladığını belirtiyor.

TIME: Kullanıcının CPU kullanım süresidir.

COMMAND: Komut bloğu olarak hangisinde olduğunu gösterir. Örnek olarak yukarıda /sbin/init auto noprompt splash komutunda olduğunu anlıyoruz.

İlk olarak PID değerini bilmemiz lazım onun için de pmap \$\$ komutunu yazdım ve ekran görüntüsünde gösterdiğim gibi değeri 9904 olarak gördüm.



```

umut@umut-virtual-machine: ~$ pmap -d 9904
pmap: bash
Address      Kbytes Mode Offset      Device Mapping
0000000000000000 188 r---- bash
0000000000000000 924 r-x-- bash
0000000000000000 208 r-x-- bash
0000000000000000 16 r----- bash
0000000000000000 48 r----- bash
0000000000000000 44 r----- [ anon ]
0000000000000000 1600 r----- local-archive
0000000000000000 136 r----- libc.so.6
0000000000000000 1508 r-x-- libc.so.6
0000000000000000 348 r----- libc.so.6
0000000000000000 16 r----- libc.so.6
0000000000000000 8 r----- libc.so.6
0000000000000000 12 r----- [ anon ]
0000000000000000 12 r----- [ anon ]
0000000000000000 68 r-x-- libc.so.6.3
0000000000000000 68 r-x-- libc.so.6.3
0000000000000000 16 r----- libc.so.6.3
0000000000000000 16 r----- libc.so.6.3
0000000000000000 4 r----- libc.so.6.3
0000000000000000 28 r-x-- /usr/lib/ld.so.cache
0000000000000000 8 r----- [ anon ]
0000000000000000 4 r----- ld-linux-x86-64.so.2
0000000000000000 164 r-x-- ld-linux-x86-64.so.2
0000000000000000 68 r-x-- ld-linux-x86-64.so.2
0000000000000000 8 r----- ld-linux-x86-64.so.2
0000000000000000 8 r----- ld-linux-x86-64.so.2
0000000000000000 112 r----- [ stack ]
0000000000000000 16 r----- [ anon ]
0000000000000000 8 r-x-- [ anon ]
0000000000000000 4 r-x-- [ anon ]
0000000000000000 4 r-x-- [ anon ]
mapped: 232K, writeable: 176K, shared: 20K
umut@umut-virtual-machine: ~$

```

Pmap -d 9904 yazınca, pmap -x 9904'ten farklı olarak Kbytes ve Mode bilgisine de ulaşabildik. Ama -X kadar geniş bir bilgi vermediği için birkaç değer de eksik örneğin: Inode, Size, Rss, Pss, Referenced, Anonymous, LazyFree, ShmemPmdMapped, FilePmdMapped, Shared_Hugetlb, Private_Hugetlb, Swap, SwapPss Locked, THPEligible, ProtectionKey gibi değerleri göstermiyor.

```

umut@umut-virtual-machine: ~$ pmap -q 9904
pmap: bash
0000000000000000 188K r---- bash
0000000000000000 924K r-x-- bash
0000000000000000 208K r-x-- bash
0000000000000000 16K r----- bash
0000000000000000 48K r----- bash
0000000000000000 44K r----- [ anon ]
0000000000000000 1600K r----- local-archive
0000000000000000 136K r----- libc.so.6
0000000000000000 1508K r-x-- libc.so.6
0000000000000000 348K r----- libc.so.6
0000000000000000 16K r----- libc.so.6
0000000000000000 8K r----- libc.so.6
0000000000000000 12K r----- [ anon ]
0000000000000000 12K r----- [ anon ]
0000000000000000 68K r-x-- libc.so.6.3
0000000000000000 68K r-x-- libc.so.6.3
0000000000000000 16K r----- libc.so.6.3
0000000000000000 16K r----- libc.so.6.3
0000000000000000 4K r----- libc.so.6.3
0000000000000000 28K r-x-- /usr/lib/ld.so.cache
0000000000000000 8K r----- [ anon ]
0000000000000000 4K r----- ld-linux-x86-64.so.2
0000000000000000 164K r-x-- ld-linux-x86-64.so.2
0000000000000000 68K r-x-- ld-linux-x86-64.so.2
0000000000000000 8K r----- ld-linux-x86-64.so.2
0000000000000000 8K r----- ld-linux-x86-64.so.2
0000000000000000 112K r----- [ stack ]
0000000000000000 16K r----- [ anon ]
0000000000000000 8K r-x-- [ anon ]
0000000000000000 4K r-x-- [ anon ]
0000000000000000 4K r-x-- [ anon ]
umut@umut-virtual-machine: ~$

```

Pmap -q 9904 yazdığımızda da Address, Kbytes, Mode ve Mapping değerleri çıkıyor ama bu değerlerin üstünde ne oldukları yazmaz. Pmap -d 9904'deki Offset ve Device değerleri de ekranda göstermez.

8. Son olarak, bellek-kullanıcı programınızda farklı miktarlarda kullanılan bellekle pmap'i çalıştıralım. Burada ne görüyorsunuz? pmap çıktısı beklentilerinizi karşılıyor mu?

```

umut@umut-virtual-machine: ~/Desktop
umut@umut-virtual-machine: ~/Desktop$ pmap memory-user.c
Usage:

umut@umut-virtual-machine: ~
umut@umut-virtual-machine: $ free --mega
total        used        free      shared  buff/cache   available
Mem:         4077       1596       1492         20       989       2209
Swap:        3376         496       2879

umut@umut-virtual-machine: $ free --mega
total        used        free      shared  buff/cache   available
Mem:         4077       1584       1504         19       988       2221
Swap:        3376         496       2879

umut@umut-virtual-machine: $ free --mega
total        used        free      shared  buff/cache   available
Mem:         4077       2429         659         19       988       1376
Swap:        3376         496       2880
umut@umut-virtual-machine: $

```

pmap memory-user.c komutu ile çalıştırmadan önce bellek değerlerimi ekrana yazdırdım. Kullanılan 1584MB'den 2429MB'ye çıktı. Boş alan da 1504MB'den 659MB'ye düştü. En son da Mevcut bellek değeri 2221MB'den 1376MB'ye geldi.