

VILDIZ TEKNİK ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

Kişisel Bilg. Çok. İşl. (Multitasking)

YÜKSEK LİSANS TEZİ

A. Tefik İnan

1992

150100
YILDIZ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

KİŞİSEL BİLGİSAYARLARDA
ÇOKLU İŞLEME
(MULTITASKING)

YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜH. AHMET TEVFİK İNAN

TEZ YÖNETİCİSİ
Prof. MEHMET YAHYA KARSLIGİL

İSTANBUL 1991

YILDIZ TEKNİK ÜNİVERSİTESİ
KÜTÜPHANE DOKÜMANTASYON
DAİRE BAŞKANLIĞI

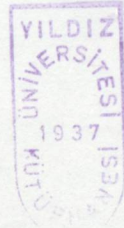
Kot : R 368
60
Alındığı Yer : Fen Bil. Ens.
Tarih : 13.4.95
Fatura :
Fiyatı : 130.000.-
Ayniyat No : 1-4
Kayıt No : 51039
UDC :
Ek :

YILDIZ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YILDIZ ÜNİVERSİTESİ

D.E. No 49898

**KİŞİSEL BİLGİSAYARLARDA
ÇOKLU İŞLEME
(MULTITASKING)**



YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜH. AHMET TEVFİK İNAN

TEZ YÖNETİCİSİ
Prof. MEHMET YAHYA KARSLIGİL

İSTANBUL 1991

1. GİRİŞ	4
2. KURAM VE YÖNTEM	5
2.1. Kuram	5
2.2. Yöntem	5
2.2.1. Veri toplama	5
2.2.2. Veri analizi	5
2.2.3. Veri yorumlama	5
2.2.4. Veri sunma	5
2.2.5. Veri değerlendirme	5
2.2.6. Veri raporlama	5
2.2.7. Veri depolama	5
2.2.8. Veri koruma	5
2.2.9. Veri yedekleme	5
2.2.10. Veri geri yükleme	5
2.2.11. Veri paylaşma	5
2.2.12. Veri izleme	5
2.2.13. Veri kontrol	5
2.2.14. Veri güncelleme	5
2.2.15. Veri silme	5
2.2.16. Veri kurtarma	5
2.2.17. Veri yedekleme	5
2.2.18. Veri geri yükleme	5
2.2.19. Veri paylaşma	5
2.2.20. Veri izleme	5
2.2.21. Veri kontrol	5
2.2.22. Veri güncelleme	5
2.2.23. Veri silme	5
2.2.24. Veri kurtarma	5
2.2.25. Veri yedekleme	5
2.2.26. Veri geri yükleme	5
2.2.27. Veri paylaşma	5
2.2.28. Veri izleme	5
2.2.29. Veri kontrol	5
2.2.30. Veri güncelleme	5
2.2.31. Veri silme	5
2.2.32. Veri kurtarma	5
2.2.33. Veri yedekleme	5
2.2.34. Veri geri yükleme	5
2.2.35. Veri paylaşma	5
2.2.36. Veri izleme	5
2.2.37. Veri kontrol	5
2.2.38. Veri güncelleme	5
2.2.39. Veri silme	5
2.2.40. Veri kurtarma	5
2.2.41. Veri yedekleme	5
2.2.42. Veri geri yükleme	5
2.2.43. Veri paylaşma	5
2.2.44. Veri izleme	5
2.2.45. Veri kontrol	5
2.2.46. Veri güncelleme	5
2.2.47. Veri silme	5
2.2.48. Veri kurtarma	5
2.2.49. Veri yedekleme	5
2.2.50. Veri geri yükleme	5
2.2.51. Veri paylaşma	5
2.2.52. Veri izleme	5
2.2.53. Veri kontrol	5
2.2.54. Veri güncelleme	5
2.2.55. Veri silme	5
2.2.56. Veri kurtarma	5
2.2.57. Veri yedekleme	5
2.2.58. Veri geri yükleme	5
2.2.59. Veri paylaşma	5
2.2.60. Veri izleme	5
2.2.61. Veri kontrol	5
2.2.62. Veri güncelleme	5
2.2.63. Veri silme	5
2.2.64. Veri kurtarma	5
2.2.65. Veri yedekleme	5
2.2.66. Veri geri yükleme	5
2.2.67. Veri paylaşma	5
2.2.68. Veri izleme	5
2.2.69. Veri kontrol	5
2.2.70. Veri güncelleme	5
2.2.71. Veri silme	5
2.2.72. Veri kurtarma	5
2.2.73. Veri yedekleme	5
2.2.74. Veri geri yükleme	5
2.2.75. Veri paylaşma	5
2.2.76. Veri izleme	5
2.2.77. Veri kontrol	5
2.2.78. Veri güncelleme	5
2.2.79. Veri silme	5
2.2.80. Veri kurtarma	5
2.2.81. Veri yedekleme	5
2.2.82. Veri geri yükleme	5
2.2.83. Veri paylaşma	5
2.2.84. Veri izleme	5
2.2.85. Veri kontrol	5
2.2.86. Veri güncelleme	5
2.2.87. Veri silme	5
2.2.88. Veri kurtarma	5
2.2.89. Veri yedekleme	5
2.2.90. Veri geri yükleme	5
2.2.91. Veri paylaşma	5
2.2.92. Veri izleme	5
2.2.93. Veri kontrol	5
2.2.94. Veri güncelleme	5
2.2.95. Veri silme	5
2.2.96. Veri kurtarma	5
2.2.97. Veri yedekleme	5
2.2.98. Veri geri yükleme	5
2.2.99. Veri paylaşma	5
2.2.100. Veri izleme	5

Yetiştirmemde en büyük desteğim olan ailem, hocalarım ve arkadaşlarıma.

İçindekiler

Özet

Abstract

1. GİRİŞ	4
2. MATERYAL VE YÖNTEM	5
2.1 Terminoloji	5
2.1.1 Multiprogramming	5
2.1.2 Multiprocessing	5
2.1.3 Multitasking	5
2.1.3.1 Multitasking tanımı	5
2.1.3.2 Multitasking'in püf noktası	6
2.2 Multitasking tipleri	6
2.2.1 Preemptive context switching	6
2.2.2 Nonpreemptive context switching	8
2.3 Multitasking de kaynakların yönetilmesi	8
2.3.1 İşlemcinin iş yükünün planlanması.....	9
2.3.1.1 İş plamlayıcısının görevi nedir?	11
2.3.1.2 Preemptive & Nonpreemptive iş planlaması	11
2.3.1.3 FIFO (First In First Out)	12
2.3.1.4 RR (Round Robin)	12
2.3.1.5 SJF (Shortest Job First)	13
2.3.1.6 STR (Shortest Remaining Time)	14
2.3.1.7 HRN (Higher Response Ratio Next)	14
2.3.1.8 MFQ (Multilevel feedback queues)	14
2.3.2 Hafıza yönetimi	17
2.3.2.1 Single contiguous memory management ..	18
(Bağlantılı tek hafıza yönetimi)	
2.3.2.2 Partitioned allocation	20
(Bölünmüş hafıza yönetimi)	
2.3.2.2.1 First fit	20
2.3.2.2.2 Best fit	21
2.3.2.2.3 Last fit	22
2.3.2.3 Paged memory management	22
(Sayfalı hafıza yönetimi)	
2.3.3 Cihaz yönetimi	25

2.3.3.1 Tahsis edilmiş cihazlar	25
(Dedicated devices)	
2.3.3.1 Paylaşılan cihazlar	26
(Shared devices)	
2.3.3.1 Zahiri cihazlar	26
(Virtual devices)	
2.3.4 Doğabilecek bazı problemler neler olabilir ? .	26
2.3.4.1 Deadly embrace	26
2.3.4.2 Race conditions.....	28
2.3.4.3 Kritik bölgeler.....	29
2.3.4.3.1 Kesmelerin engellenmesi	30
2.3.4.3.2 Kitlenmiş değişkenler.....	30
2.3.4.4 Semaphore	30
2.4 DOS işletim sisteminde çoklu iş yürütme	31
2.4.1 8086 Ailesi kesmeler ve kesme vektörleri.....	32
2.4.1.1 İç donanım kesmeleri	32
2.4.1.2 Dış donanım kesmeleri.....	32
2.4.1.3 Yazılım kesmeleri	33
2.4.1.4 Kesme vektörü	33
2.4.1.5 Kesme vektörü değiştirilmesi	33
2.4.2 INDOS bayrağı	35
2.4.3 Zamanlayıcı kesmesi	35
2.4.4 DOS programlarına tekrar girme	35
3. DENEYSEL ÇALIŞMA VE BULGULAR	37
3.1 Uygulanan metodun tanıtımı	37
3.1.1 İlk değer atama	37
3.1.1.1 Argümanların alınması	38
3.1.1.2 Kontrol programının kullandığı	40
hafıza alanının ayarlanması	
3.1.1.3 Kesme vektörünün değiştirilmesi	41
3.1.1.4 08h kesmesinin beklenmesi	41
3.1.2 İşin yüklenmesi	42
3.1.2.1 Hafızanın tahsis edilmesi	42
3.1.2.2 Alt işlemlerin yaratılması	43
3.1.2.3 Çalıştırılacak modülün yüklenmesi	43

3.1.3 İşin çalıştırılması	44
3.1.3.1 Bir işten diğerine geçme	45
3.1.3.2 İşin bitirilmesi	45
3.1.4 Programın sona erdirilmesi	46
3.1.4.1 Normal bitiş	46
3.1.4.2 Ctrl-Break ile bitirme	46
4. TARTIŞMA VE SONUÇ	48
4.1 İşlerin işlem zamanları	48
4.2 Zaman istatistikleri	48
4.3 Özel Zaman istekleri.....	48
4.3.1 Turn around time	48
(Geri dönüş süresi)	
4.3.2 Elapsed time	49
(Kayıp edilen süre)	
4.3.3 Weighted turn around time	49
(Ağırlıklı geri dönüş süresi)	
5. EKLER	50
5.1 PSP (program Segment Prefix) Yapısı	50
5.2 Com program yapısı	52
5.3 Exe program yapısı	53
5.3.1 Exe header	53
5.4 Bios/Dos Kesmeleri	56
5.4.1 Bios kesmelerinin listesi	56
5.4.2 Dos kesmelerinin listesi	59
5.4.2.1 Dos fonksiyon listesi	60
5.4.3 Kullanılan kesmeler ve açıklamaları	65
5.5 Kullanılan programların çıkışları	76
5.5.1 Switch.Asm	76
5.5.2 Macro.Lib	103
5.5.3 Extra.Lib	110
5.5.4 Debug.Lib	114
5.5.5 Create.Pas	116
5.5.6 Programın Akışı	128
5.6 Üretiten raporlar	131
KAYNAKLAR	134
SONSÖZ	136

ÖZET

PC olarak bilinen kişisel bilgisayarlarda çoklu işleme (Multitasking) temel olarak tek bir işlemcide (CPU Central processing Unit) birden fazla işin aynı anda çalıştırılıyormuş izleniminin yaratılarak çalıştırılması prensibine dayanmaktadır.

Bu tez çalışmasında 'Kişisel Bilgisayarlarda Çoklu İşleme' konusunu ele alarak, yaptığımız çoklu işleme tanımına uyan bir sistem yarattım. Sistem ana hatlarıyla Dos (Disk Operating System - Disk İşletim Sistemi) da uygulanabilen bir kontrol programı, bu programın ihtiyaç duyduğu macroları içeren ve .Lib uzantılı dosyalardan ibarettir. Bunlara ek olarak pascal dilinde yazılmış bir program yardımıyla programın çalışması sırasında elde edilen sonuçları değerlendirme imkanı da buldum.

Sistem Batch (toplu işleme) olarak çalıştırılabilen .Exe yada .Com uzantılı çalıştırılabilir (executable) programları iş olarak çalıştırmak üzere yaratılmış sunni bir ortam yaratmaktadır.

ABSTRACT

Basically , multitasking on personal computers is to give the impression of running several tasks on a single CPU (Central Processing Unit).

The system that we created to realize a multitasking environment is similar to the description above. The system is simply a control program running under Dos (Disk Operating System) and several macros under the file extension .Lib and also a Pascal program which analyse the data file which is created during the execution of the program.

This system is an artificial environment where we can run the files which has .Exe and .Com extension as a batch process.

1.GİRİŞ

Yüzyılımızdaki son derece hızlı teknolojik gelişmeler ile performansları inanılmaz boyutlara ulaşan bilgisayarlar artık tek iş düzeninden uzaklaşıp çok iş düzenine ve çok kullanılcılığa hale gelmişlerdir. Bu nedenden dolayı Dos üzerinde yaptığımız çalışma ile daha fazla işi tek CPU da multitasking olarak çalışmayı amaçladık.

Ana hatlarıyla program biribirinden bağımsız işleri tek CPU' da aynı anda çalıştıyormuş izlenimini yaratarak çalıştırmaktır.

Tez çalışmasında 8086 assmebly dili kullanılmıştır. Hafızanın işlere bölünmesi ve diğer kontrol fonksiyonlarını yerine getirmek için Dos'un kesmelerinden yararlanılmıştır. Yaratılan raporlar 8086 ile yazılmış programın ürettiği bilgilerden oluşan bir bilgi dosyasının Pascal programlama dilinde yazılmış bir program yardımıyla değerlendirilmesiyle oluşturulmuştur.

Test etmek amacıyla kullanılan programlar ise C, Pascal ve Assembly dili ile yazılmış .Exe yada .Com tipi programlardır.

Bu tez çalışmasının bu konuya ilgi duyan arkadaşlara yardımcı olacağına inanıyorum.

Tezimin gelişmesinde katkılarından dolayı sayın Prof. M. Yahya Karsılgil'e, zaman zaman karşılaştığım problemleri çözmemde bilgi desteğini esirgemeyen meslektaşım A. Sedat Alış'a teşekkürü bir borç bilirim.

2.MATERYAL VE YÖNTEM

2.1 Terminoloji

Multitasking'in ne olduğunu anlatmadan önce birbiriyle çok sık karıştırılan Multiprogramming, Multiprocessing deyimlerinin ne anlama geldiklerini açıklamakta ve kavram kargaşasını ortadan kaldırmakta fayda vardır.

2.1.1 Multiprogramming (Çoklu programlama)

Multiprogramming de tek bir CPU (Central Processing Unit - Merkezi İşlem Birimi) mevcuttur. Bu CPU zaman paylaşımı (time sharing) olarak mevcut işlere (job) hizmet vermektedir.

2.1.2 Multiprocessing (Çoklu işlemci ile işleme)

Multiprocessing de birden fazla CPU mevcut olup birden fazla iş (job) aynı anda farklı işlemcilerde işlenebilmektedir. Bunun özel bir halide 'Paralel processing' olarakta adlandırılabilir.

2.1.3 Multitasking (Çoklu işleme)

Bazı kaynaklarda multitasking, multiprogramming olarak değerlendirilmektedir. Diğerleri ise multiprogramming, multiprocessing yada her ikisininide içeren bir kavram olarak değerlendirilmektedir.

2.1.3.1 Multitasking'in tanımı

Multitasking (Çoklu işleme) çoğu zaman 'birkaç program yada procedure'un aynı anda çalıştırılması' şeklinde yanlış olarak tarif edilmektedir. Bu tanım 'paralel processing'

yapabilen ve birden fazla CPU ya sahip sistemler için doğrudur. Zira işler ayrı ayrı işlemcilerle aktarılarak aynı anda birden fazla işin fiziksel olarak çalıştırılması mümkün olabilmektedir. Ancak elimizdeki sistemin tek CPU 'ya sahip olduğu ve bu CPU 'nun her seferinde ancak ve ancak bir iş yerine getirebileceği gerçeği göz önüne alınırsa bu tanımın multitasking'in tarifi olmadığı anlaşılacaktır.

2.1.3.2 Multitasking'in püf noktası

Burada amaç CPU'nun mümkün olduğunca çabuk bir şekilde mevcut işler (task) arasında gidip gelerek onlara hizmet vermesi ve kullanıcıya da sanki birden fazla iş aynı anda çalıştırıyormuş izlenimini vermesidir. İşte bu tarife uygun olarak çalışan sistemler multitasking sistem olarak adlandırılmaktadır.

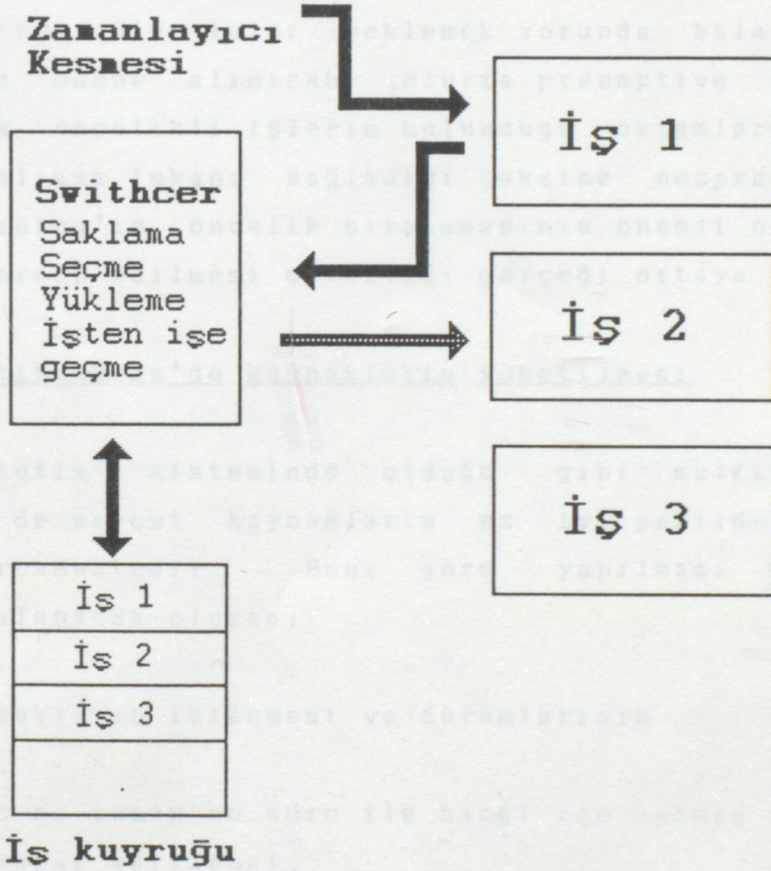
2.2 Multitasking tipleri

CPU'nun kısa aralıklarla mevcut işlerden bir birine bir diğerine hizmet vermesi 'task switching' yada 'context switching' (iş anahtarlama) olarak adlandırılmaktadır. Temel olarak iki ayrı tip iş anahtarlama (context switching) söz konusudur.

2.2.1 Preemptive context switching

Bu tip multitasking'de çalışmakta olan iş (task) durumdan habersiz olarak çalışmaktadır. Sistemin oluşturduğu bir zamanlayıcı kesmesi ile tetiklenen iş yerini bir diğer işe bırakmakta ancak bundan haberdar olmamakta ve halen çalıştığını zanetmektedir. (Bakınız Şekil-1) Oysa kontrol başka bir işe verilmektedir. CPU'nun bir işten alınarak diğer iş için tahsis edilmesi sırasında

daha önceden CPU'yu elinde tutan işe ait bilgiler saklanmakta (bilgilerin saklanması kasıt saklayıcı değerlerinin saklanmasıdır. Çalışan işin kodu hafızada yüklü bulunur.) ve bu sayede tekrar ilk iş (kendisine sıra geldiğinde) kaldığı yerden çalışabilmektedir. (Zamanlayıcı kesmesi Dos ortamında saniyede 18.2 defa oluşmaktadır. Ancak multitasking sistemlerde bu işaret her üretildiği zaman kullanılmaz. Ne zaman kullanacağını belirleyen ve adına quantum dediğimiz süreler ile kullanılırlar. Bu konu hakkında daha detaylı bilgiler veren başlıklar altında verilecektir.)



Sekil-1

2.2.2 Nonpreemptive context switching

Bu tip multitasking'de preemptive context switching 'de olduğu gibi işler (Task) olup bitenlerden habersiz değil aksine olaya hakim durumdadır. Preemptive'de iş (task) kaynakları belirlenen süre için kullanabilirken Nonpreemptive'de iş kaynağı elde ettikten sonra işini bitirene yada kaynağı bırakmayı isteyene kadar elinde tutar.

Bu bir takım problemleri de beraberinde getirmektedir. Eğer kaynağı elde eden iş oldukça uzun süren bir iş ise ve kaynağı işini bitirinceye kadar bırakmıyor ise çok daha kısa süre çalışması gereken ve öncelik itibariyle diğerlerinden üstün olan işler beklemek zorunda kalacaktır. Bu durum göz önüne alınacak ,olursa preemptive context switching 'in öncelikli işlerin bulunduğu ortamlarda daha etkin bir çalışma imkanı sağladığı ,aksine nonpreemptive context switching'in öncelik sıralamasının önemli olmadığı işlemlerde tercih edilmesi gerektiği gerçeği ortaya çıkar.

2.3 Multitasking'de kaynakların yönetilmesi

Her işletim sisteminde olduğu gibi multitasking sistemlerde de mevcut kaynakların en iyi şekilde idare edilmesi gerekmektedir. Buna göre yapılması gereken işlemler sıralanacak olursa;

- * Mevcut kaynakların izlenmesi ve durumlarının belirlenmesi,
- * Kaynakların ne zaman ne süre ile hangi işe tahsis edileceğine karar verilemesi,
- * Verilen karar doğrultusunda kaynakların tahsis edilmesi,
- * İşin sona ermesini takiben kaynakların geri alınması,

olmak üzere karşımıza 4 madde çıkar. Bunları en genel yönetim aşamaları olarak düşündüğümüzde işlemcinin, hafızanın ve diğer kaynakların yönetim kademelerinde karşımıza çıkacağı açıktır.

2.3.1 İşlemcinin iş yükünün planlanması

(Processor scheduling)

İşlemcinin iş çizelgesinin hazırlanması işletim sistemi tarafından yerine getirilen oldukça karmaşık bir işlemdir. Processor scheduler, işlemcinin hangi işe (process) tahsis edileceğini belirlemekle yükümlüdür. Şekilden de anlaşılacağı üzere 3 temel seviyede olduğu düşünülebilir. (bakınız Şekil-2)

a-) High level (üst seviye iş çizelgesi)

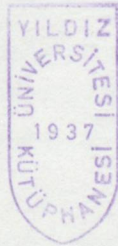
Bu işin kabul edilmesi aşamasıdır. Bir kere iş kabul edilmiş ise artık zaman içerisinde ihtiyaç duyduğu kaynaklar tahsis edilecek ve diğer kademelerinden geçerek sonuca ulaşacaktır.

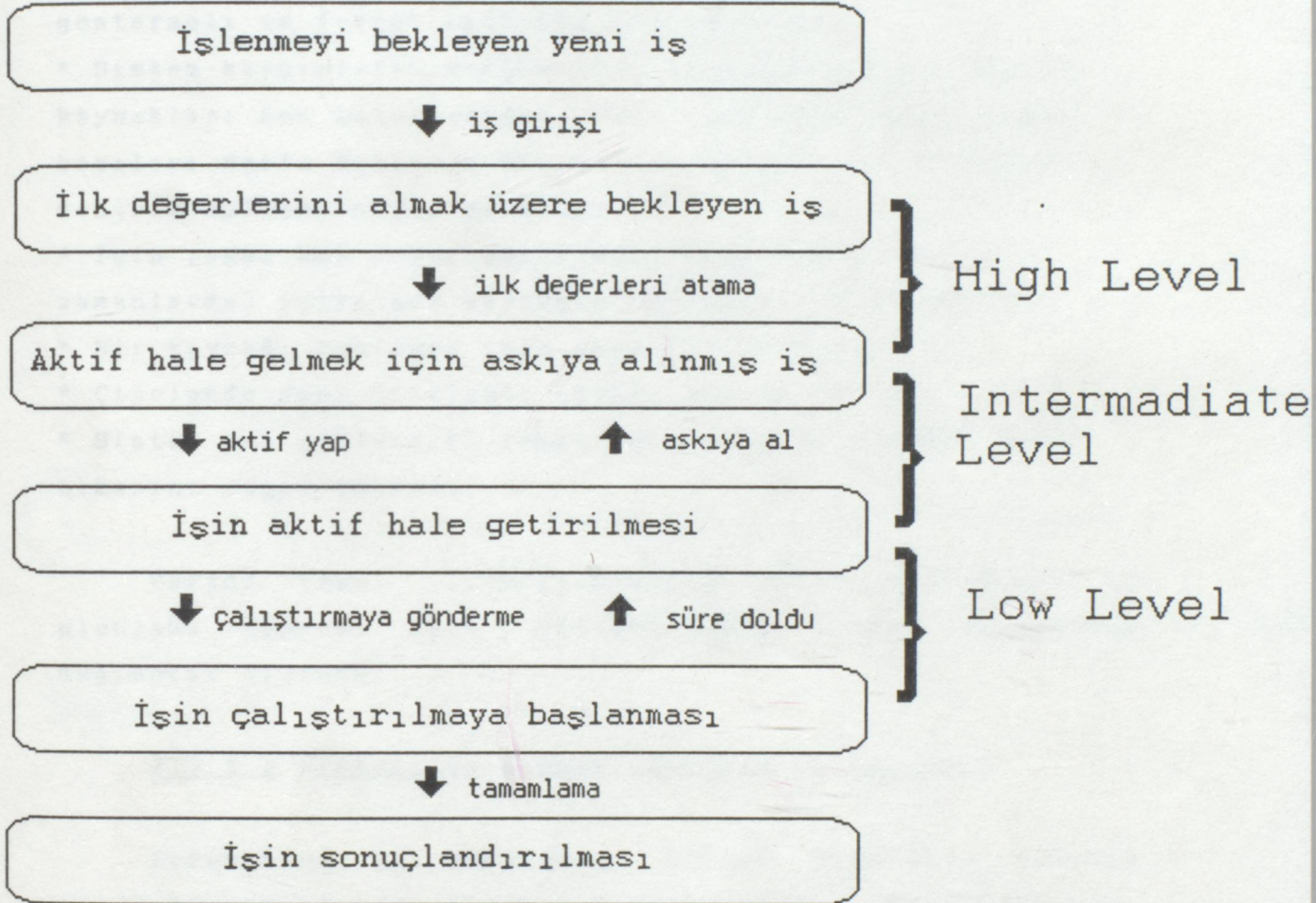
b-) Intermediate level (orta seviye iş çizelgesi)

Hangi işin (task) CPU'ya nasıl hakim olacağını belirlemekle yükümlüdür. Uygulanan politika doğrultusunda hangi işin durdurulacağını, hangisinin kontrolü alacağını belirlemekte kullanılmaktadır.

c-) Low level (alt seviye iş çizelgesi)

Hazır olan işlerden hangisine CPU'nun tahsis edileceğine karar verir.





Sekil-2

2.3.1.1 İş planlayıcısının görevi nedir?

- * Üretilen çıkış miktarı mümkün olduğunca fazla olmalı ve mümkün olduğunca çok işe hizmet verilmelidir.
- * Interactive (etkileşimli) kullanımı artırmalıdır.
- * Her işe (job) sistemin yüküne bakmaksızın aynı önem göstermeli ve fırsat eşitliği sağlanmalıdır.
- * Sistem kaynakların kullanımını dengelemelidir. (Sistem kaynakları boş kalmamalıdır. Eğer sistemde kaynaklardan bazıları fazla bazıları hiç kullanılmıyor ise kaynaklar dengeli kullanılmıyor demektir.)
- * İşin zamanında sonuçlanabilmesi için (gerekli olduğu zamanlarda) yeterince kaynağın bulunması gerekmektedir.
- * Bir kaynağı bekleyen işin önceliği artmalıdır.
- * Çizelgede daha öncelikli işlere yer vermelidir.
- * Sistem çok yüklendiği zaman yeni işlerin sisteme dahil olmasını engellemelidir.

Burada temel fonksiyonlardan bahis ettiğimiz iş planlama aşaması için geliştirilmiş birkaç stratejiye değinecek olursak;

2.3.1.2 Preemptive & Nonpreemptive iş planlama

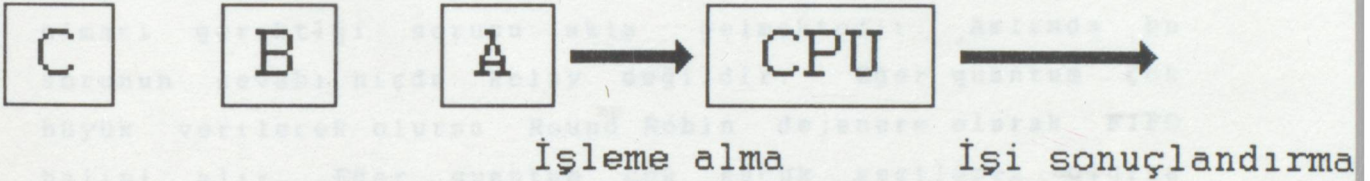
Preemptive iş planlama, yüksek öncelikli işlerin işlediği ortamlarda oldukça faydalı olur. (Bilindiği gibi preemptive scheduling'de CPU belirlenen quantum sonunda bir işten (task) diğerine aktarılabilmekte, bunun aksine nonpreemptive de CPU bir işe tahsis edildiği zaman o iş CPU yu bırakmadığı sürece geri alınamamaktadır.) Birkaç işin ana hafızada bulunması bir sonraki işin hazır durumda olması sağlayacağından dolayı etkinliği artırılmış olur. Nonpreemptive sistemlerde uzun süren iş CPU 'yu kısa süren işten önce elde etmiş ise kısa sürecek işin, uzun olanın bitmesini beklemesi yada uzun işin, CPU 'yu bırakması

gerekmektedir.

2.3.1.3 FIFO (First in first out)

Belkide en basit çizelge oluşturma metodudur. İşler kuyruğa geliş sıraları göz önüne alınarak işlenmekte ve tamamlanmaktadırlar. (bakınız Şekil-3) FIFO aslında nonpreemptive iş planlamaya iyi bir örnektir. Her defasında CPU'ya sahip olan iş yerine getirmesi gereken görevleri tamamlamadan diğer iş CPU'ya sahip olamayacaktır. Kısacası uzun süren önemsiz işler bitinceye kadar kısa süreli ancak önemli işler beklemek zorunda kalacaklardır.

İş kuyruğu



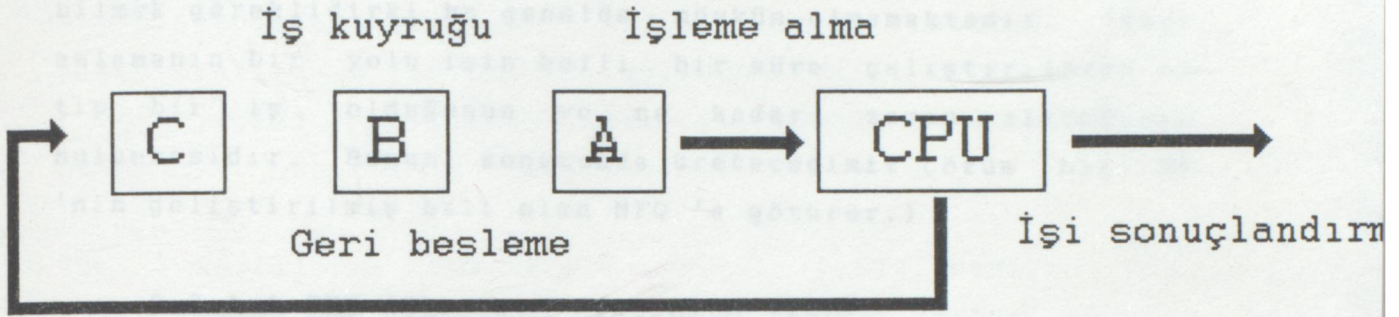
Sekil-3

2.3.1.4 RR (Round Robin)

Bu metoda da işler FIFO daki gibi CPU'ya sahip olur ancak quantum yada time-slice dediğimiz bir süre sonunda kontrol programı tarafından CPU elinden alınır ve sırada beklemekte olan diğer işe (task) verilir. Eğer bir önceki iş tamamlanmamışsa kuyruğun sonuna yerleşerek tekrar sıranın kendisine gelmesini bekleyecektir. (bakınız Şekil-4)

Round Robin oldukça etkili bir zaman paylaşımı (time-

sharing) ortamı sağlar ve tüm işlerin sonucu oldukça uygun bir zaman sonunda elde edilmiş olur.



Sekil-4

Burada quantum yada time-slice olarak ifade ettiğimiz ve her taskın işlemde kalma süresini belirleyen sabitin ne olması gerektiği sorusu akla gelmektedir. Aslında bu sorunun cevabı hiçde kolay değildir. Eğer quantum çok büyük verilecek olursa Round Robin dejenere olarak FIFO halini alır. Eğer quantum çok küçük seçilecek olursa zamanın çoğu işlemcinin bir işten diğer bir işe aktarılması sırasında kayıp edilecektir ki buda işin diğer negatif yönüdür.

Bu durumda en uygun quantum zamanın bir işten diğer işe geçmek ile harcanacak kadar küçük, ayrıca RR'yi dejenere ederek FIFO haline getirecek kadar da büyük olmamayacak şekilde seçilmelidir.

2.3.1.5 SJF (Shortest Job First)

(Kısa iş öncelikli)

Bu metod nonpreemptive iş planlamasına bir örnektir. Ancak kısa sürecek işlerin önce işleme alınması durumunda bu işler sistemi çabuk terk ederek daha uzun süreli işlere

yer açarlar. Bu arada weighted turn around time (bakınız 4.4.1 Ağırlıklı geri dönüş süresi) değeri düşecektir. Ancak bu metoda hangi işin ne kadar zaman süreceğini tam olarak bilmek gereklidirki bu genelde mümkün olmamaktadır. (Bunu anlamamanın bir yolu işin belli bir süre çalıştırılarak ne tip bir iş olduğunun ve ne kadar zaman alacağını bulundurmasıdır. Bunun sonucunda üreteceğimiz çözüm bizi RR'nin geliştirilmiş hali olan MFQ 'a götürür.)

2.3.1.6 STR (Shortest Remaining Time)

(Kalan zamanı az olan iş öncelikli)

STR, SJF'nin preemptive olan cinsi olarak adlandırılabilir. STR de çalışmakta olan iş daha kısa çalışma süresi olan iş tarafından bölünebilmekte ve kısa iş öncelikli olarak işlenmektedir. Bu durumda yeni gelen kısa süreli işler hemen işleme alınmakta ve işlem tamamlanmaktadır.

2.3.1.7 HRN (Highest Response Ratio Next)

(Cevap süresi oranı yüksek iş öncelikli)

HRN bir nonpreemptive iş planlamadır. Burada her işin önceliği bekleme ve servis süresinin bir fonksiyonu ile tespit edilmektedir. Bunun için belirlenen formül şu şekildedir;

öncelik = (beklenen süre + hizmet süresi) / hizmet süresi

Bu durumda uzun zaman bekleyen ve kısa zamanda servis verilebilecek tip program yada işlerin önceliği fonksiyon uyarınca artırılmaktadır.

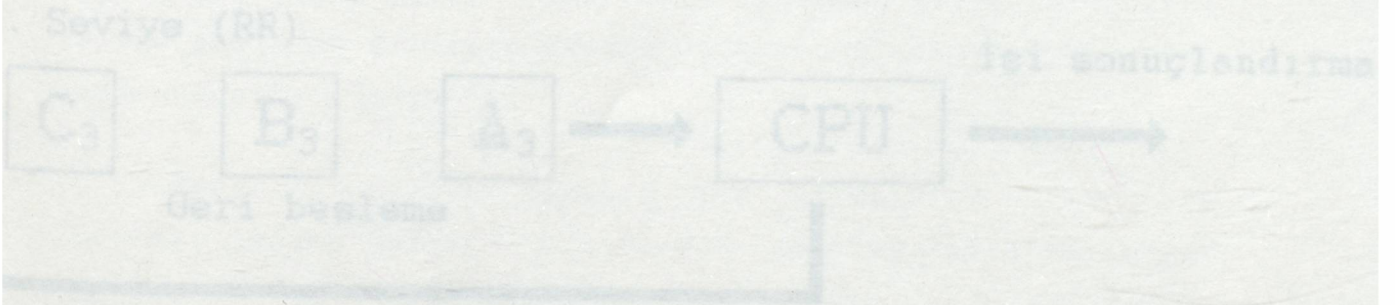
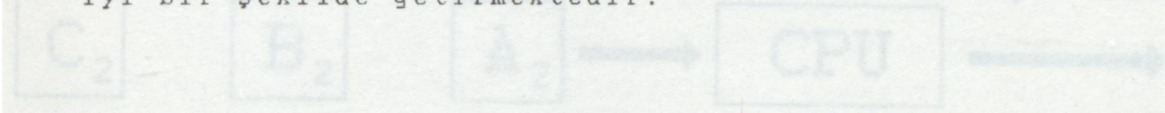
2.3.1.8 MFQ (Multilevel feedback queues)

(Çok seviyeli geri beslemeli kuyruklar)

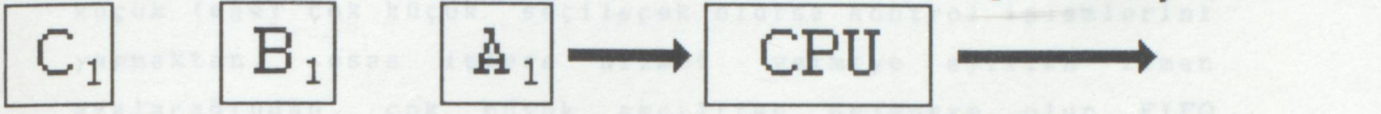
Bir iş çizelgesinde,

- * Kısa sürecek işlere öncelik vermelidir.
- * I/O ihtiyacı duyan işlere I/O birimlerini mümkün olduğunca çok ve kesintisiz kullanma imkanı vermelidir.
- * İşin yapısı, tipi ve ihtiyaçlarını mümkün olduğunca çabuk belirleyip ona göre çalıştırmalıdır.

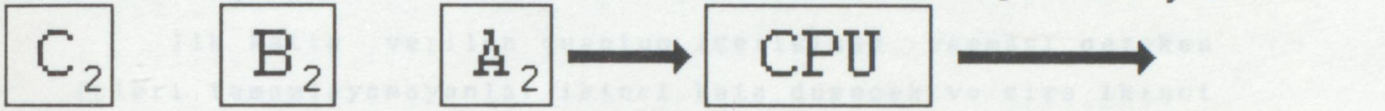
İşte multilevel feedback queues (yada benzer olan multilevel Round Robin) bu yükümlülükleri yerine oldukça iyi bir şekilde getirmektedir.



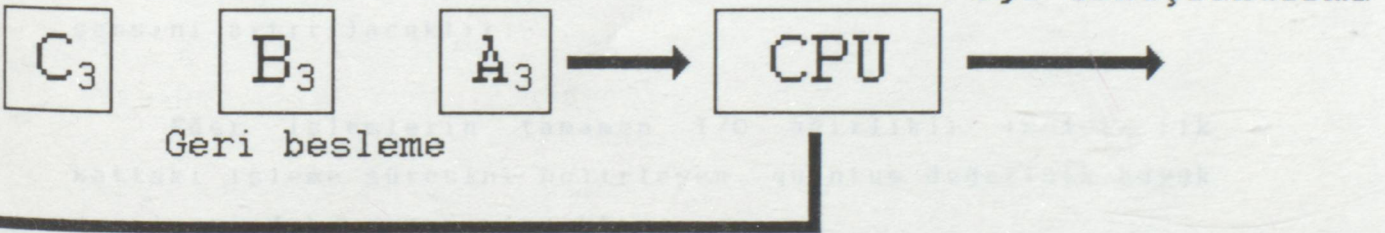
1. Seviye (FIFO)



2. Seviye (FIFO)



N. Seviye (RR)



Sekil-5

Buradaki yapıyı, sisteme alınan işlerin CPU işlem ağırlıklı, yada I/O işlem ağırlıklı olarak ayrı ayrı incelenmesinde yarar vardır. İşler CPU ağırlıklı iseler ilk kattaki quantum uygun ölçüler içerisinde kalmak üzere küçük (eğer çok küçük seçilecek olursa kontrol işlemlerini yapmaktan, esas işlere hizmet vermeye ayrılan zaman azalacağından, çok büyük seçilirse dejenere olup FIFO halini alacağından dolayı sistem performansında relatif olarak düşecektir.) daha sonraki katlara doğru artacak şekilde düzenlenmelidir.

İlk katta verilen quantum içerisinde yapması gereken işleri tamamlayamayanlar ikinci kata düşecek ve sıra ikinci kata geldiğinde ilk kattakine nazaran daha büyük bir quantum değeri ile işe kaldıkları yerden devam edeceklerdir.

Bu katların özellikleri FIFO olarak çalışmalarıdır. Aksine en son kat RR olarak düzenlenmiş ve en fazla quantum ile işlenecek işlerin bulunduğu kattır. Burada işlemi verilen quantum'da tamamlayamayan işler aynı katta kalacaklar ancak kuyruğun sonunda yer alacaklardır. Bu şekilde az işlem zamanına ihtiyaç duyan işler doğal bir öncelik kazanırken uzun sürebilecek hatta uzun I/O işlemleri yapan işler daha sonra işlenmek üzere daha geri plana atılarak sistemin mümkün olduğunca çabuk cevap verme şansını artırılacaktır.

Eğer işlemlerin tamamen I/O ağırlıklı iseler ilk kattaki işleme süresini belirleyen quantum değerinin büyük verilmesi daha uygun olacaktır.

2.3.2 Hafıza yönetimi

İşlemcinin mevcut işlerin (task) ihtiyaçlarına cevap verecek şekilde belirli metodlar uyarınca nasıl hizmet

verdiğini incelemiştik. Şimdi hafızanın mevcut olan bu işlerin ihtiyaçlarını karşılamak üzere nasıl yönetilmesi gerektiğini ve bunun için izlenmesi gereken stratejilerin neler olduğunu inceleyeceğiz. Hemem belirtelim ki tüm yönetim sistemlerinde olduğu gibi hafıza yönetimi yapılırkende şu dört temel işlem gerçekleşmelidir.

- * Hafızanın izlenmesi (boş ve dolu olan hafıza alanlarını gösteren bilgilerin tutulması)
- * Hafızaya ihtiyaç duyulduğunda hangi işin ne süre ile, ne kadar hafızayı kullanması gerektiğine karar verilmesi (buna belirlenen strateji doğrultusunda karar verilecektir.)
- * Hafızanın iş için tahsis edilmesi (allocation)
- * İşin bitiminde işe verilen hafızanın geri alınarak daha sonraki işlerin ihtiyaçlarını karşılamak üzere kullanılması. (deallocation)

Şimdi sırası ile birkaç hafıza yönetim tekniğine (stratejisine) değinelim ve bunların sağladıkları kolaylıkların ve eksik yönlerinin neler olduğunu görmeye çalışalım.

2.3.2.1 Bağılantılı tek hafıza yönetimi

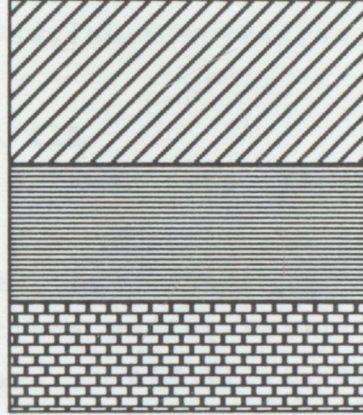
(Single contiguous memory management)

Hafızanın bir seferde ancak bir iş için kullanıldığı bir yönetim şeklidir. Burada hafızanın temel olarak üç parçadan oluştuğunu söylemek pek hatalı olmaz. Bunlardan ilki işletim sisteminin bulunduğu bölümdür. Geri kalan parça tamamen iş için ayrılmış olan hafıza alanıdır. Ancak her işin artakalan bu hafızayı kullanması düşünülemez. Dolayısıyla, geri kalan hafızayı da iki bölüm halinde görürüz. Bunlardan ilki işin yüklendiği ve gerçekten kullanılan hafızadır. İkincisi ise iş için ayrılmasına rağmen iş tarafında kullanılmayan (yada kullanılamayan) ve

tamamen atıl bir kapasite oluşturan hafıza alanıdır. Bu tip basit hafıza yönetimi microcomputer olarak bilinen ve yaygın olarak PC denilen kişisel bilgisayarlarda kullanılmaktadır. (bakınız Şekil-6)

Kullanıcının
iş i için
ayrılmış
olan hafıza
alanı

İşletim
Sistemi



Kullanıcı programı i
ayrılmasına rağmen
kullanılmayan hafıza
alanı

Kullanıcının program
tarafından kullanılan
hafıza alanı

Sekil-6

Bu hafıza yönetim sistemini avantajı hafızada neresinin boş, nersinin dolu olduğunu takip etmenin büyük bir zorluk olmamasıdır. Ayrıca çalışması için ek bir donanım ihtiyacı da yoktur. Ancak hafızanın efektif kullanılamaması en büyük problemidir. Genelde hafıza çalıştırılan (yada çalıştırılması düşünülen) en büyük programın sığabileceği şekilde seçilir. Buna rağmen çalıştırılan programlar bu hafızanın tamamını çok nadir olarak kullanırlar. Dolayısıyla çok nadir kullanılacak (belkide hiç kullanılmayacak) programları çalıştıracığımızı düşünerek büyük hafıza için yatırım yapılmaktadır.

2.3.2.2 Bölünmüş hafıza yönetimi

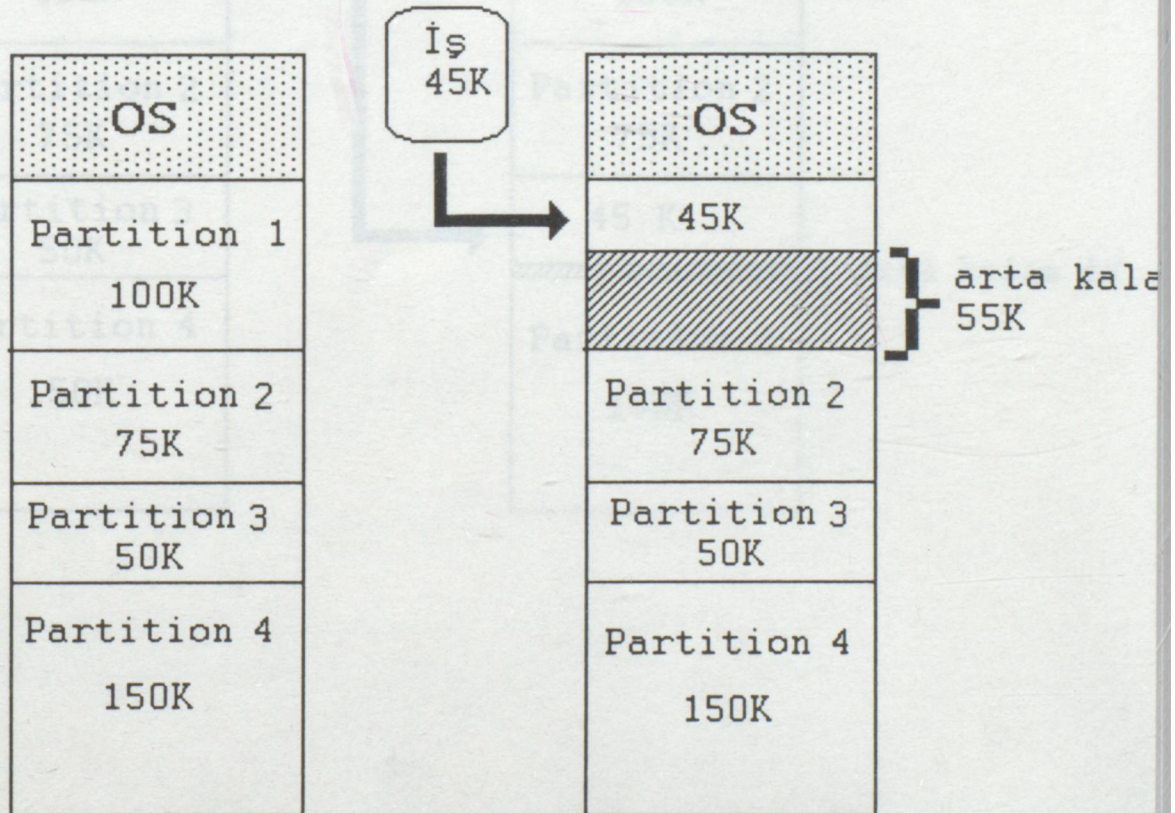
(Partitioned allocation)

Bu tip hafıza yönetiminde mevcut hafıza önceden belirlenmiş sabit büyüklüklerde, partition adı verilen parçalara ayrılmıştır. İşler, partition adı verdiğimiz bu hafıza alanlarına yüklenir. İşlerin hafızaya yüklenmeside belirlenecek strateji doğrultusunda olacaktır. Bu stratejilere kısaca değinmek gerekirse;

2.3.2.2.1 First fit

(İlk bulunan uygun alana yerleştirme)

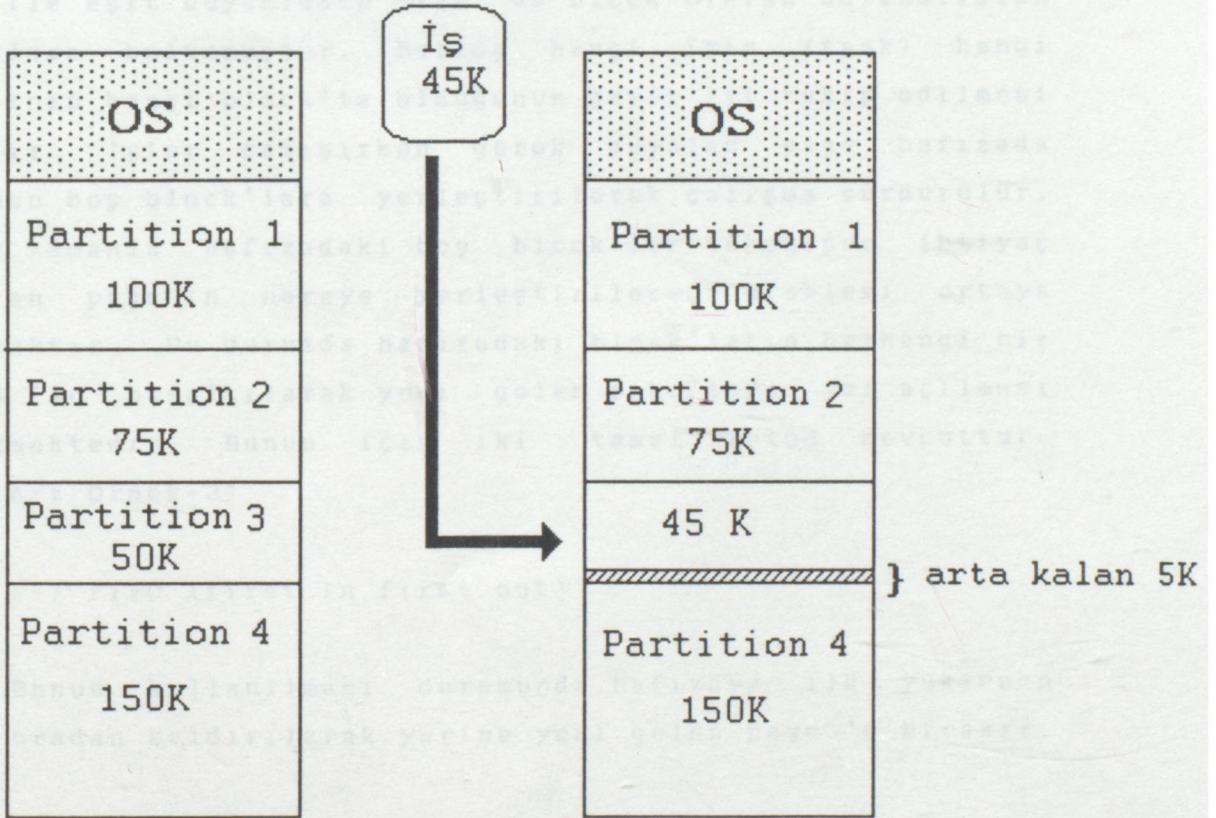
Bu stratejide iş (task) bulunan ilk (işin sığabileceği) partitiona yerleştirilir. Bu stratejinin bir takım sıkıntıları vardır. 45K hafıza ihtiyacı duyan iş 100K lık bir partition'a yerleştirilebilir. Bu durumda hafızanın 55K'lık kısmı ziyan olurken o alanda çalıştırılabilabilecek 100k'lık bir işi çalıştıramama gibi bir olumsuzlukta mevcuttur. (bakınız örnek-1)



2.3.2.2.2 Best fit

(Bulunan en uygun alana yerleştirme)

Best fit stratejisi, iş hafızadaki partition'lardan en uygun olanına yerleştirmeye çalışır. Bu sayede first fit 'te karşımıza çıkan ve ziyan edilen hafıza olarak bahis ettiğimiz hafıza miktarı asgari değere indirilmeye çalışılmıştır. Gerçi her iki strateji içinde hangi işin nereye yerleştirildiğinin izlenmesi gerekmektedir ancak best fit de en uygun olan partition'ın bulunması için bir takım fazladan kontrol ve işlemlerin yapılması gerekmektedir. (bakınız Örnek-2)



Örnek-2

2.3.2.2.3 Last fit

(Bulunan en son uygun alana yerleştirme)

Bu tamamen First fit'in aynısı olmasına rağmen farklı olarak ilk bulunan uygun alana koymak yerine sondaki uygun alana işi yerleştirir.

2.3.2.3 Sayfalı hafıza yönetimi

(Paged memory management)

Bu yöntemde her iş, page adı verilen eşit büyüklüklerdeki (bu sistemden sisteme değişiklik göstermektedir. Örnek olarak IBM 370 de page büyüklüğü 4k dır) parçalara ayrılmıştır. Buna benzer olarak hafıza da page ile eşit büyüklükte olan ve block olarak adlandırılan parçalara bölünmüştür. Burada hangi işin (task) hangi page'inin hangi block'ta olduğunu gayet iyi takip edilmesi şarttır. İşler çalışırken gerek duyulan page hafızada bulunan boş block'lara yerleştirilerek çalışma sürdürülür. Ancak zamanla hafızadaki boş block'lar tükenince ihtiyaç duyulan page in nereye yerleştirileceği problemi ortaya çıkacaktır. Bu durumda hafızadaki block'ların herhangi bir metod ile boşaltılarak yeni gelen page'lere yer açılması gerekmektedir. Bunun için iki temel metod mevcuttur. (bakınız Örnek-3)

a-) FIFO (first in first out)

Bunun kullanılması durumunda hafızaya ilk yüklenen page oradan kaldırılarak yerine yeni gelen page 'e bırakır.

b-) LRU (Least recently used)

Bu metod da son zamanda en az kullanılan block'lar, (en az referans edilen block'lar) bundan sonrada ihtiyaç

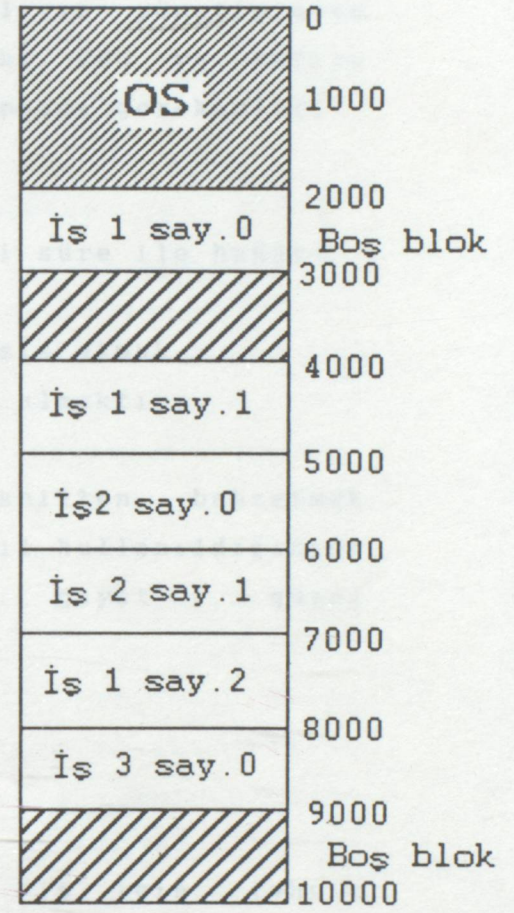
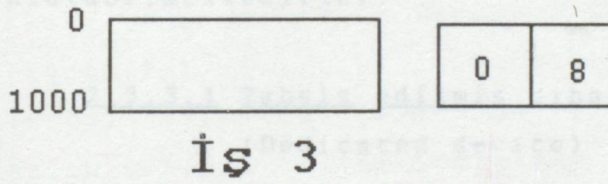
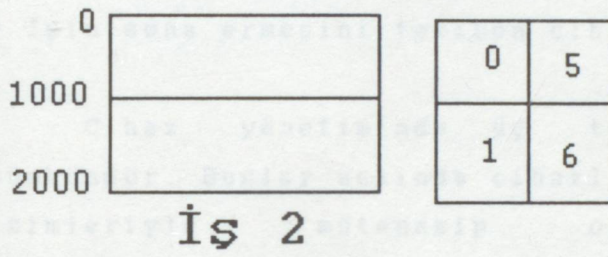
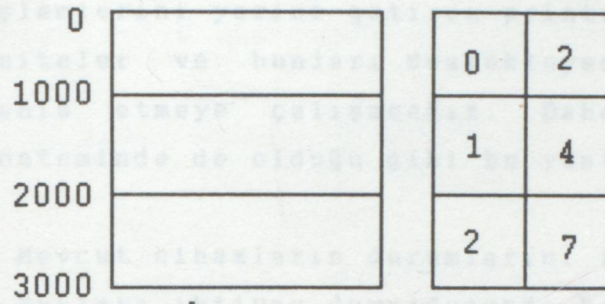
duyulmayacağı düşünülerek kaldırılır ve yerine yeni page yerleştirilir.

Bu metodlarla ufak tefek farklılıkları olan iki ayrı metod daha mevcuttur. Bunlar LFU (Least frequently used) ve NUR (Not used recently) dir.

Sayfalı hafıza yönetiminde önemli nokta kontrolün zorluğu ve karmaşıklığıdır. Ancak bu ek donanım elemanlarının sisteme dahil edilmesi ile aşılabilecek bir problemdir. Sayfalı hafıza yönetiminin en büyük avantajı ufak bir hafıza alanı kullanarak çok büyük işleri çalıştırılabilmesidir. Dolayısıyla bu yönetim şeklinde hafıza efektif kullanılmaktadır. Ayrıca az hafıza ile çok büyük işleri çalıştırabilmesi hafıza için yapılacak yatırımın atıl kalmasını önlemektedir.

Örnek-3

Sayfa Blok



Örnek-3

2.3.3 Cihaz yönetimi

Bu başlık altında CPU ve hafıza dışında kalan ve I/O işlemlerini yerine getiren printer (yazıcı), tape, disk vs. üniteler ve bunları destekleyen birimlerin yönetiminden bahis etmeye çalışacağız. Daha önceki CPU ve hafıza yönteminde de olduğu gibi bu yöntemin yapması gerekenler;

- * Mevcut cihazların durumlarını izlemek
- * Bunlara ihtiyaç duyulduğunda kime hangi süre ile hangi cihazın tahsis edileceğine karar vermek
- * Verilen karar doğrultusunda cihazı tahsis etmek
- * İşin sona ermesini takiben cihazı geri almaktır.

Cihaz yönetiminde üç temel teknikten bahsetmek mümkündür. Bunlar aslında cihazların nasıl kullanıldığını da isimleriyle mütenasip olarak gayet güzel anlatabilmektedirler.

2.3.3.1 Tahsis edilmiş cihaz

(Dedicated device)

Bu tip cihazlar tamamen bir iş için tahsis edilmişlerdir. Dolayısıyla, o iş sona erene kadar onun kontrolü altında çalışacaklardır. Bunlara bir örnek vermek gerekirse Card reader (kart okuyucu), tape yada printer (yazıcı) sayılabilir. Bu cihazlar bağlı bulundukları sistem içerisinde ancak bir işe hizmet verebilirler.

2.3.3.2 Paylaşılan cihazlar

(Shared device)

Disk tipindeki cihazlar birden fazla iş tarafından aynı anda kullanılmaktadır. Ancak bu durumda bazı kontrol mekanizmalarına ihtiyaç duyulacaktır. Örneğin iki işte A

diskinden okumaya kalkıştığında önce hangi işin okuma yapacağına karar vermelidir. Eğer bir iş diğerinden daha öncelikli değil ise birtakım problemlerin ortaya çıkacağı açıktır. (Bunlara ileriki başlıklarda değineceğiz ve problemlerden bahis etmeye çalışacağız.)

2.3.3.3 Zahirî cihazlar

(Virtual devices)

Bazı tahsis edilmiş cihazlar bu yöntem yardımıyla paylaşılabilir cihazlar haline gelebilirler. Bu işlem 'spooling' olarak bilinmektedir. Yazıcının kullanımı buna en iyi örneği teşkil etmektedir. Herhangi bir iş yazıcıya ihtiyaç duyduğunda eğer printer meşgul ise onun yazıcıya yollayacağı bilgi disk üzerinde saklanır. (disk paylaşılabilir cihaz olduğu için erişmede herhangi bir problem çıkmayacaktır.) Yazıcı yapmakta olduğu işi bitirdikten sonra diske kaydedilmiş olan bu bilgileri geliş sırası ile alarak işleme devam eder. Bu şekilde kullanıcılar yazıcıyı ikendisine tahsis edilmiş gibi görünür. Gerçekte, yazıcı bu iş için tahsis edilmemiştir. İşte bu şekilde kullanılan cihaz tek olmasına rağmen pek çok işe aynı anda servis veriyormuş gibi görünmektedirler. Buda zahirî olarak adlandırılmalarındaki en büyük etkendir.

2.3.4 Doğabilecek bir takım problemler neler olabilir?

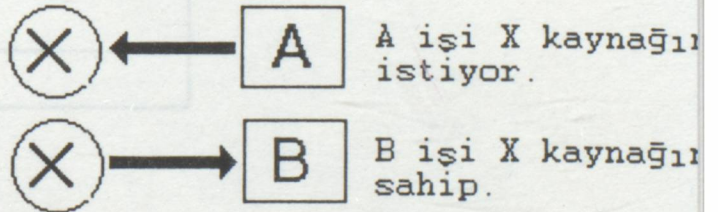
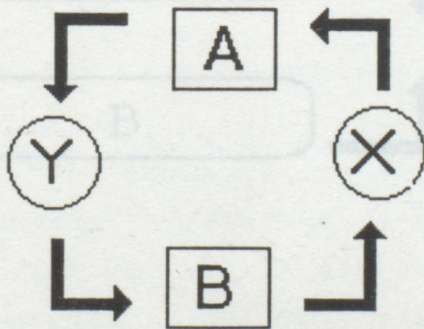
2.3.4.1 Deadly embrace

(Ölümcul kucaklaşma)

Hepimizin bildiği gibi bir iş çalışması sırasında hafıza, disk, printer ve diğer kaynaklara ihtiyaç duymaktadır. İşletim sistemi de uyguladığı strateji uyarınca bu kaynakları iş için tahsis etmekte, iş ise onlara ihtiyaç duyduğu sürece elinde tutmakta ve

kullanılmaktadır. Zaman zaman kaynakların kullandırılması sırasında işlerin bu kaynaklara ulaşması mümkün olamamaktadır. Bu durum sistemin kitlenmesine neden olmaktadır (dealy embrace). Bu durumu basit bir örnekle (bakınız Şekil-7) açıklamak istersek ;

Scanner'den (tarayıcı) bilgi okuyacak ve bunları diske yazacak olan iki ayrı iş düşünelim. Okuma işleminin gerçekleşmesi için önce scanner'e ardındanda diske erişilmesi gerekmektedir. Şimdi işin birincisinin scanner'e sahip olduğunu, diğerinin ise diske sahip olduğunu düşünelim. Bu durumda ilk işin tamamlanabilmesi için diske ihtiyaç duyulacak, ancak disk diğer işin elinde olduğundan ona yazamayacak, dolayısıyla işlemi tamamlanamıyacaktır. Benzer şekilde ikinci iş de okuma işlemini yapabilmesi için scanner'e ihtiyaç duyulacak, ancak scanner (tarayıcı) başka bir işe tahsis edildiği için ikinci iş de okumayı yapamayacak ve bağlı olarak işlemi tamamlayamayacaktır. Her iki iş de diğerinin elinde bulunan kaynağı bırakmasını bekleyecek, fakat bu arada kendi sahip olduğu kaynağı da bırakmayacaktır. Bunu basit bir şema ile ifade edersek;

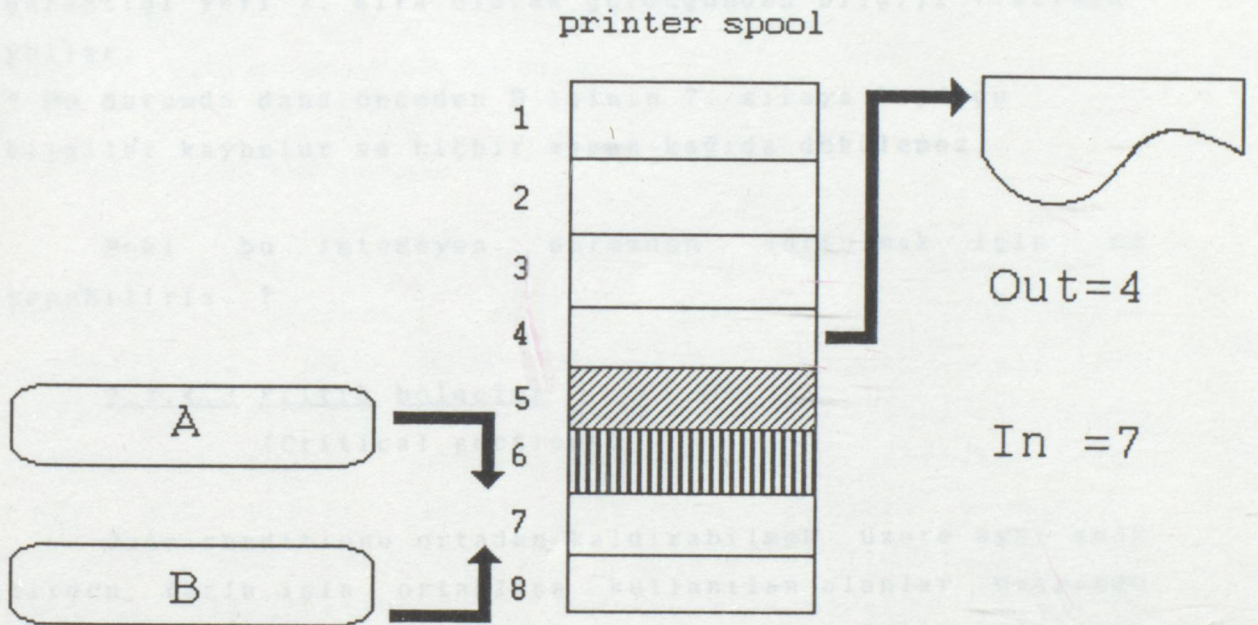


Sekil-7

Bu durum başka bir işe tahsis edilmiş kaynağı bekleyen işin kendisine tahsis edilmiş kaynağı tutması durumunda meydana gelecektir. Eğer bu tip durumlar oldukça seyrek meydana geliyor ise insanlar deve kuşu algoritmasını (ostrich algorithm) kullanarak kafalarını toprağa gömer ve problemi görmemezlikten gelirler. Bu durum multitasking yapan microcomputer'lerde çok karşılaşılan bir durumdur. Ancak makinalar donanım olarak bu engeli aşmak üzere tasarlanmıştır. Bu donanım makinenizin üzerindeki kırmızı büyük açma kapama düğmesinden başka bir şey değildir.

2.3.4.2 Race condition

(Yarış koşulu)



Sekil-8

Bu problemi ortaya koymak üzere şu örnek üzerinde dikkatimizi toplayalım. Sistemimizde 10 file kapasiteli bir printer spool'u olsun. Bu spooler'in boş olan gözünü IN isimli değişkenin gösterdiğini düşünelim. Aynı anda A ve B isimli iki işin printer spooluna bir takım bilgiler yollamaya çalıştığını düşünelim. Bu koşullar altında hangi işin printer spoolunu önce kullanacağı bir bilinmez olmasına rağmen şunların sıra ile oluştuğunu farzedelim.

- * A işi IN değişkenini okur.
- * Bir saat darbesi gelir ve CPU B işine tahsis edilir.
- * B işi IN değerini okur ve 7. sıraya bilgiyi yollar.
- * B işi IN değerini bir artırarak bir sonraki boş spool alanını gösterir hale getirilir.
- * Sıra A işine geldiği zaman spool file '1'ı koymasına gerektiği yeri 7. sıra olarak gördüğünden bilgiyi 7.sıraya yollar.
- * Bu durumda daha önceden B işinin 7. sıraya koyduğu bilgiler kaybolur ve hiçbir zaman kağıda dökülemez.

Peki bu istemeyen durumdan kurtulmak için ne yapabiliriz. ?

2.3.4.3 Kritik bölgeler

(Critical sections)

Race conditionu ortadan kaldıracılabilmek üzere aynı anda birden fazla işin ortaklaşa kullanılan alanlar üzerinde izinsiz okuma ve yazma yapmaması ve buradaki bilgileri bozmaması gerekmektedir. İşlerin içerisinde ortak kullanılan hafıza alanlarına erişilmesini sağlayan kodun bulunduğu bölümlere 'critical section' adı verilir. İşte bu alanlara aynı anda iki işinde girmesi engellenmelidir. Kritik bölgeden çıkan işler diğerlerinin herhangi bir şekilde çalışmasını engellememelidirler.

2.3.4.3.1 Kescmelerin engellenmesi

Burada cözümlerden biri de herhangi bir iş critical section'a girince kescmelerin etkisiz kılması ve çıkmadan hemen öncede kescmeleri eski haline getirilmesi gerekmektedir. Bu durumda CPU saat darbesine bağılı olarak bir işten diğetine hizmet vermek üzere tetiklenmelidir.

2.3.4.3.2 Kitlenmiş değışkenler

Bu kritik alanlara girme problemini ilk değıerini sıfır kabul ettiğimiz bir değışkeni kontrol ederek önleme yoluna gidebiliriz. Bu sayede kritik bölgeye giren iş bu değışkeni bir yapar. Eğer diğeri bir iş kritik alana girecek olursa değışken bir durumunda olduğu için giremez ve sıfır olana kadar bekler. Yinede burada printer (yazıcı) spool'u için bahsettiğimiz kör nokta mevcuttur.

2.3.4.4 Semaphore

Bu metod 1965 yılında Disjkstra tarafından ortaya atılmıştır. Disjkstra Up ve Down olarak iki işlem tanımlamıştır. Down işlemi semaphore'un değıerinin sıfırdan büyük olup olmadığını kontrol eder. Eğer değıer sıfırdan büyük ise azaltır ve işlemeye devam eder. Aksine değıer sıfır ise işlemi durdurur (Bundan, işlemin uyutulması olarakda bahis edilebilir.). Yapılan işlemler (kontrol değıerin değıştirilmesi ve gerekiyorsa işlemin uyutulması 'atomic action' olarak isimlendirilen bölünmez bir bütündür.) Semaphore işlemi başladığında o işlem bitmeden hiçbir şekilde semaphore'a erişilemez. Yapılması gereken iş tamamlandıktan sonra Up işlemi semaphore'un değıerini artırır. Eğer bir yada birden fazla iş daha önceki Down işlemini bitirmemiş ve halen uyumakta ise rasgele (random) olarak bir tanesi sistem tarafından seçilir ve Down

işleminin tamamlanması sağlanır. Semaphore artırılarak tekrar sıfır durumuna getirilir.

Bu şekilde aynı ortamları birden fazla işin kullanılmasını engellenerek çıkabilecek sorunlar ortadan kaldırılmış olur.

2.4.1 8086 ailesi kesmeler ve kesme vektörleri

8086 ailesi kesmeler temel olarak 3 grupta toplanmaktadır.

- * İç donanım kesmeleri (internal interrupts)
- * Dış donanım kesmeleri (external interrupts)
- * Yazılım kesmeleri (software interrupts)

Şimdi bunlara kısaca göz atalım;

2.4.1.1 İç donanım kesmeleri

Bunlar işlemciye donanım olarak bağlanmış ve özel durumlarda oluşan kesmelerdir. Bunların oluşması durumunda işlemci hatayı sezer. Bunlardan sık karşılaşılanları;

00h	Sıfıra bölme (divide by zero)
01h	Tek adım çalıştırma (single step)
02h	Maskelenemez kesme (nonmaskable interrupt)
03h	Break point (durma noktası)
04h	Taşma (overflow)

Bunlardan hiç birini doğrudan kullanmayız. (Ancak bazı paket yazılımlar programın işleyişinin bozulmasını engellemek ve kullanıcının kendi yarattıkları sunni ortamdan çıkmasını engellemek amacıyla bu tip kesmelerin vektörlerini değiştirerek kendi yazdıkları programları bunların yerine çalıştırır ve kullanıcılarını uyarırlar. Bu sayede makinada doğan bu maskelenmez kesmeler sonucunda kitlenmemiş olur. Kesme vektörlerinin nasıl değiştirildiğine ilerideki başlıklarda ayrıntısı ile değinmeye çalışacağız.)

2.4.1.2 Dış donanım kesmeler

Bunlar herhangi bir birimin/cihazın kesme üreterek işlemciyi uyarması için donanım olarak işlemciye bağlanmıştır. İki tür bağlantı söz konusudur. Maskelenemez (NMI) ve maskelenemez (INTR)

INTR 'nin oluşması engellenememekte ancak NMI 'ın oluşması engellenememektedir.

2.4.1.3 Yazılım kesmeleri

Bu program tarafından kullanılan kesme komutunun sonucunda oluşur. Bu durumda işlemci sanki bir donanım kesmesi gelmiş gibi hareket eder. Bu metod yardımıyla Dos ve Bios servislerine bağımsız olarak erişmek mümkün olmakta hatta bu kesme vektörleri program tarafından değiştirilebilmektedir. (Bu işlem ayrıntıları ile 2.4.1.4 başlığı altında açıklanmıştır.)

2.4.1.4 Kesme vektörü

Bu vektör sistemin hafızasının en altındaki 1024 byte'ı kapsar. Her kesme 4 byte kullanmakta, buda 256 adet kesme bilgisinin saklanması için imkan sağlamaktadır. Her 4 byte'lık bilginin 2 byte'ı segment, kalan 2 byte'ıda offset olarak kullanılmaktadır.

2.4.1.5 Kesme vektörünün değiştirilmesi

Kesme vektörü eğer kurallara uygun olarak değiştirilmez ise sistemin çalışmasını etkileyecek hatta kitlenmesine neden olacaktır.

Zaman zaman ticari yazılımlar (özelliktede TSR

olanlar) kesme vektörünü değiştirmektedir. Ancak program işlenmesi bittikten hemen sonra değiştirdiği kesme vektörünün orijinalini yerine koyarak sistemi terk etmektedir. Kısaca iş kullandığı herşeyi işlem bittikten sonra yerli yerine koymalıdır.

Kesme vektörünü değiştirmenin iki yolu vardır. Bunlardan bir tanesi kesme vektörünün bulunduğu hafıza alanına erişilerek oradaki bilginin değiştirilmesidir. Ancak bu yöntemin bir takım tehlikeli yönleri de mevcuttur. Değiştirmenin tamamlanabilmesi için 4 byte bilginin o hafıza alanına yazılması gerekmektedir. Bu 4 byte 'tan 2 tanesi yazıldıktan hemen sonra değiştirilmesi istenen kesme herhengi bir şekilde oluşursa vektörü tam olarak değiştirilmemiş olduğu için yanlış bir adrese gidecek ve muhtemelen sistemin kitlenmesine neden olacaktır.

Bunu engellemenin bir yolu bu 4 byte o hafıza alanına yazılmadan hemen önce CLI ile kesmelerin engellenmesi ve hemen sonunda da STI ile kesmelere izin verilmesidir. Bu şekilde en azından vektörün değiştirilmesinden önce kesme gelmemiş olacaktır.

Bu işi yapmanın ikinci ve en iyi yolu ise Dos'un sağladığı kolaylıklardan yararlanmaktır. Buna göre kesme vektörünün değiştirilmesi için şu işlemlerin sırası ile yapılmasında fayda vardır.

1-) INT 21h Fonksiyon 35h kullanılarak istenilen kesmenin vektör değeri alınır ve daha sonra işlemlerin tamamlanmasını takiben yerine konmak üzere hafızada saklanır.

2-) INT 21h Fonksiyon 25h kullanılarak istenilen kesme vektör değeri değiştirilir.

3-) İşlemler bittikten sonra hafızaya saklanan orijinal kesme vektörü yine INT 21h fonksiyon 25h yardımıyla yerine konulur ve işlem tamamlanır. (Program içinde kullandığımız kesmeler hakkında ayrıntılı bilgiler 5.4 Dos kesmeleri başlığı altında bulunabilir.)

2.4.2 Indos bayrağı

INT 21h, fonksiyon 34h'i çağırarak InDos bayrağına bir işaretçi elde edebiliriz. (Bu fonksiyon Dos'un özel kesmelerinden birisidir.) Bu işaretçinin gösterdiği değer eğer 1 ise Dos herhangi bir fonksiyonu yerine getirmek üzere kesilmiş ve iş yaptığı için meşgul durumdadır. Bu durumda Dos'un işini bitirmesini beklemek gerekmektedir. Dos meşgul değil ise yapılması gereken iş için Dos'un sağladığı kolaylıkları kullanabiliriz.

2.4.3 Zamanlayıcı kesmesi (INT 08h)

INT 08h sistem saati tarafından saniyede 18.2 defa çağırılmaktadır. (Buda 55ms de bir ddefa üremekte olduğunu gösterir.) INT 08h aynı zamanda INT 1Ch (timer tick) 'de çağırır. INT 08h, IRQ0 bacağına bağlı olduğu için en yüksek öncelikli donanım kesmesidir ve diğer tüm kesmelerden daha önce servis verilir.

Bu sistemin gün/saat fonksiyonlarını yerine getirebilmesi için gerekli bir kesmedir. Ayrıca disk biriminin motor çalıştırma/durdurma fonksiyonunu gerçekleştirmesi için zaruridir.

2.4.3 Dos a tekrar girme (Reentrancy)

Dos servisleri, tek kullanıcı, tek iş düzenine göre

2. DENEYSEL ÇALIŞMA VE BULGULAR

çalışan bir sistem olarak tasarlanmıştır. Bilgisayar dünyasında çoklu iş düzeni ve çok kullanıcıli sistemler oldukça fazla konuşulan konulardandır. Çoklu iş düzenini yapabilmek için Dos'a alternatif olarak OS/2 işletim sistemi yazılmıştır.

Dos, tek kullanıcıli bir sistem olarak tasarlandığından, Dos servislerinin kullanımına bir takım kısıtlamalar getirmektedir. Dos servis programları global bazı veri alanları kullanırlar. Şayet kullanılan kesme yordamı bir Dos servisini tekrar çağırarak olursa, Bu taktirde kesme oluşmadan önce yürütölmekte olan servise ait veriler bozulacaktır. Bu nedenden dolayı Dos servis programının içine tekrar girilmez. Bu nedenden dolayı Dos servisleri çağırılmak istendiğinde Dos'un bu servis programlarının içinde olmadığından emin olunmalıdır.

3. DENEYSEL ÇALIŞMA VE BULGULAR

3.1 Uygulanan metodun tanıtımı

Bu projede preemptive context switching ile multitasking'i gerçeklemeye çalıştık. Amaç multitasking tanımında da belirttiğimiz gibi birden fazla işi aynı anda çalıştırıyormuş izlenimini yaratmaktır. Projeye başlarken kendimizce çalışmamızın sınırlarını kaba hatlarıyla belirlemiş ve bir takım kısıtlar koymuştuk. Bunlardan bir tanesi batch tipi çalışabilecek işlerle multitasking yapmaktır. Ayrıca zaman içerisinde uygulamadan kaynaklanan kısıtlar da ortaya çıktı. (Bunlar da yeri geldikçe belirtmeye çalışacağız.) Temel hatlarıyla yazdığımız program kendisine argüman olarak verilen .EXE veya .COM tipindeki çalıştırılabilir (executable) file 'ları işlemek üzere tasarlanmıştır. Sistem tarafından üretilen zamanlama işaret ile (daha öncedende söylediğimiz gibi) preemptive context switching yapmaktadır. Yani çalışmakta olan iş (task) ne olduğunu anlamadan devreden çıkarılmakta ve sırada çalışmayı bekleyen diğer iş faaliyete geçirilmektedir.

İleride doğabileceğini düşündüğümüz bir takım hafıza yönetim problemlerini ortadan kaldırabilmek için bir anda hafızada en fazla iki işe (task) yervermeyi uygun bulduk.

Program hakkında daha detaylı bilgi takip eden başlıklar altında verilmeye çalışacaktır.

3.1.1 İlk değerlerin belirlenmesi

Swicthe.asm adını verdiğimiz program sırası ile şu fonksiyonları gerçeklemektedir.

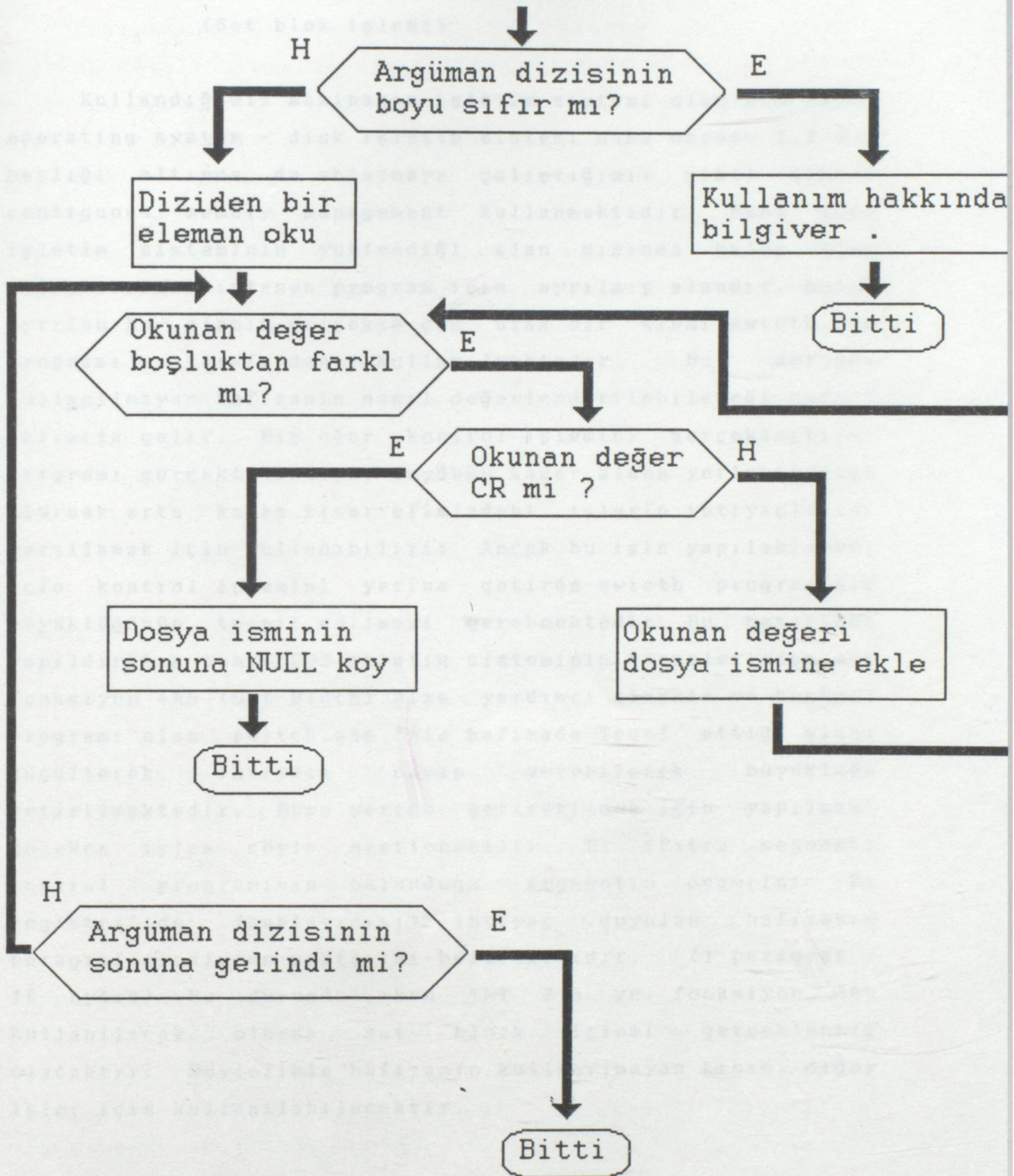
3.1.1.1 Argümanların alınması

Switch.asm programı çalıştırması gereken programların isimlerini argüman olarak kabul etmektedir. İlk yerine getirilen fonksiyon bu argümanların alınması böylece çalıştırılacak program isimleri ve bunların bulundukları yeri belirleyen Path'lerin tespit edilmesidir. Daha önce söylediğimiz gibi switch.asm, uzantısı .EXE ve .COM olan programları çalıştırmak üzere tasarlanmıştır. Buna göre çalıştırılması düşünülen programların (task) isimleri uzantıları ve path'larda dahil olmak üzere verilmelidir. (Exe ve Com tipi programlar yapı itibari ile farklılıklar göstermektedir. Bunları 5.2 ve 5.3 numaralı başlıklar altında ayrıntıları ile açıklanmıştır.) Program argümanları (DOS işletim sisteminin özelliğinden kaynaklanmak üzere) programın PSP'sinde (program segment prefix - açıklama için 5.1 numaralı başlığa bakınız) saklıdır. Bu bilgiye ulaşabilmek için öncelikle 80h adresinden bu argüman dizisinin uzunluğunu belirten bilgi bulunmakta bunu takiben (81h adresinden itibaren) argüman dizisi bulunmaktadır. Argüman dizisi PSP'nin sonu olan FFh adresine kadar devam etmektedir. Burada yeri gelmişken kendiliğinden ortaya çıkan bir kısıttan bahis etmek istiyoruz.

Bu program kaç tane işi parametre olarak kabul edebilir ?

Bunu aslında argüman dizisinin boyutu ve bu dizide bulunan programların path ve program isimlerinin ne kadar uzun olduğuna bağlıdır. Bu basit olarak 127 karakter kabul edebilecek bir dizidir. Ancak buraya yazılan path name ve program isim bilgileri uzun ise az sayıda task işlenebilecektir. Ancak azami olarak 20 kadar işi bir seferde argüman olarak kabul etmesi teorik olarak mümkündür. Bununla birlikte argüman olarak kabul edilebilecek task sayısı program içerisinde de

tanımlanabilmekte ve gerekirse değiştirilebilmektedir.



Akış (PARSE)

3.1.1.2 Kontrol programının kullandığı

hafıza alanının ayarlanması

(Set blok işlemi)

Kullandığımız makinanın işletim sistemi olan DOS (disk operating system - disk işletim sistemi daha öncede 2.3.2.1 başlığı altında da anlatmaya çalıştığımız gibi) single contiguous memory management kullanılmaktadır. Buna göre işletim sisteminin yüklendiği alan dışında kalan alan çalıştırılmak istenen program için ayrılmış alandır. Ancak ayrılan bu alanın gerçekte çok ufak bir kısmı swich.asm programı tarafından kullanılmaktadır. Bu durumda kullanılmayan hafızanın nasıl değerlendirilebileceği sorusu aklımıza gelir. Biz eğer kontrol işlemini gerçekleştiren programı gerçekte ihtiyaç duyduğu kadar alana yerleştirecek olursak arta kalan kısmı elimizdeki işlerin ihtiyaçlarını karşılamak için kullanabiliriz. Ancak bu işin yapılabilmesi için kontrol işlemini yerine getiren swich programının büyüklüğünün tespit edilmesi gerekmektedir. Bu hesaplama yapıldıktan sonra DOS işletim sisteminin kesmelerinden 21h fonksiyon 4Ah (Set Block) bize yardımcı olmakta ve kontrol programı olan switch.asm 'nin hafızada işgal ettiği alanı küçülterek ihtiyaca cevap verebilecek büyüklüğe getirilmektedir. Bunu yerine getirebilmek için yapılması gereken işlem şöyle özetlenebilir. Es (Extra segment) kontrol programının bulunduğu segmentin değerini Bx register'ide (saklayıcısı) ihtiyaç duyulan hafızanın paragraf cinsinden miktarını belirtmelidir. (1 paragraf = 16 bytes) Bu durumda eğer INT 21h ve fonksiyon 4Ah kullanılacak olursa set block işlemi gerçekleşmiş olacaktır. Böylelikle hafızanın kullanılmayan kısmı diğer işler için kullanılabilecektir.

3.1.1.3 Kesme vektörlerinin değıştirilmesi

Program esas olarak INT 08h sistem saat kesmesi üzerine dayalı , olarak çalışmaktadır. Amaç sistem saatinin ürettiğı darbelerden yararlanarak CPU'nun bir işden (task) diğetine geçerek hizmet vermesini sağlamaktır. Dolayısıyla INT 08h'in yerini alacak bir program dilimi (handler) yazılmış ve kontrol saniyede 18.2 defa bu program dilimine aktarılmıştır. Bu program dilimi aslında tüm kontrol fonksiyonlarını da yerine getiren modülleri içermektedir. Bu modüller ilerideki başlıklar altında detaylı olarak incelenecektir. INT 08h yanı sıra INT 22h (terminate address) ve INT 23h (Ctrl-Break) içinde ayrı ayrı program dilimleri de yazılmıştır. Bunlar da ileriki başlıklar atında yeri geldikçe incelenecektir.

3.1.1.4 08h kesmesinin beklenmesi

Gerekli işlemler tamamlandıktan sonra (argümanlar alınıp gerekli kesme vektörleri değıştirildikten sonra) artık işlemi başlatacak, yada tabir doğru ise tetiğı çekecek bir şeye ihtiyacımız vardır. İşte saniyede 18.2 defa oluşan INT 08h bu tetikleme vazifesini yerine getirmek üzere kullanılmaktadır. Sistem INT 08h 'i ürettiğı zaman kontrol bizim yazdığımız ve orijinal 08h yerine görev yapmak üzere tasarlanmış program dilimine gitmekte ve kontrol işlemleri bu program dilimi tarafından yönetilmektedir. Kontrolü ele geçiren bu program vakit kaybetmeden orijinal INT 08h koduna dallanarak yürütmesi gereken temel fonksiyonları yerine getirmektedir. (interrupt 08h sistem saat ve zamanının yanı sıra disket motorunu da kontrol fonksiyonları için gerekli sinyalleri üretmektedir.) Bu işlemi takiben kendisinin yerine getirmekle yükümlü olduğı adımları yapmaktadır.

3.1.2 İşin yüklenmesi

Bir taskın yüklenip çalıştırılabilmesi için herşeyden önce DOS'un buna izin vermesi gerekmektedir. Yani ilk iş olarak INDOS flag kontrol edilerek bu işlemin yapılıp yapılamayacağına karar vermesi gerekmektedir. Eğer DOS meşgul ise INT 08h geldiği yere dönecek aksi halde iş çalışırken işin saklayıcı (register) değerleri bozulmadan saklanacaktır. Ancak bundan sonra işin hafızaya yüklenmesi aşamasına geçilebilir. İş aşağıda bahis eden kademelerden geçerek hafızada yerini almaktadır.

3.1.2.1 Hafızanın tahsis edilmesi

Bir işin çalıştırılabilmesi için öncelikle hafızada yerini alması gerekmektedir.

Tez çalışması sırasında bu hafızaya yerleştirme işlemini ile ilgili program dilimini yazarken karşılaştığım problemler ve bunun sonucunda uygulamaya karar verdiğim metodları şu şekilde özetleyebilirim;

1-) Kontrol programı (switch.asm) hafızanın tümüne hakim iken (3.1.1.2 başlığı altında da anlatıldığı gibi) işlerin yüklenmesi için hafıza alanını gerekecektir. Bu amaçla kontrol programının kendine yetecek kadar hafıza alanına yerleştirdik. Bu sayede elde ettiğimiz hafıza alanını, projenin planlama safhasında koyduğumuz kısıtlara uygun şekilde, iki işi yükleyebileceğimiz şekilde organize ettik. (Bu işlemi yaparken müsait olan hafıza miktarını PSP 'nin 02h byte'ından aldık ve kontrol programının işgal ettiği miktarı bundan çıkartarak kalan hafızayı iki işe eşit olarak pay ettik.)

2-) Gelen işlerin ihtiyaç duyduğu hafıza miktarını

hesaplayarak ihtiyaç duyulan kadar hafızayı işe tahsis etmeye karar verdik. Ancak burada bir çıkmazla karşılaştık. C programlama dilinde yazılmış olan örneği test ederken 'allocation error' ile karşılaştık. (Bunun nedeni C programlama dilinin allocation'u kendisinin apmasıdır. Dolayısıyla C de yazılan programlara kod kadar hafıza yeterli olmamakta ve hata ile karşılaşılmaktadır.)

3-) Hatayı ortadan kaldırmak üzere işin ihtiyaç duyacağı miktarı hesaplamak yerine hafızanın yarısını bu işe tahsis ettik. (Burada hemen belirtmek gerekirken çalıştırılabilir programın büyüklüğü tamamen partitionun büyüklüğüne bağlıdır. Partitioned memory managementda da olduğu gibi hiçbir zaman partitionun büyüklüğünden daha büyük bir program çalıştırmak mümkün değildir.) Bu şekilde hafıza yönetim sistemi partitioned memory management halini almış oldu.

3.1.2.2 Alt işlemler yaratılması

Herhangi bir iş için hafızada yer ayarlandıktan sonra yapılması gereken iş bu iş için bir PSP (program Segment Prefix) yaratmaktır. Bu amaçla Dos'un özel kesmelerinden (reserved interrupt) INT 21h fonksiyon 55h kullanılmıştır. Bu fonksiyon aslında fonksiyon 26h'e benzemektedir. Ancak 26h gibi sadece ana programın PSP'sini kopyalamak yerine tamamen farklı bir PSP yaratmaktadır. (Bu aslında INT 21h fonksiyon 4Bh exec fonksiyonu için yapılması gerekli bir işlemdir.)

3.1.2.3 Çalışılacak modülün yüklenmesi

İş için hafıza tahsis edilip PSP yaratıldıktan sonra yapılması gereken şey artık çalıştırılması istenen işin kodunun hafızaya yüklenmesidir. Bu amaçla Dos INT 21h

fonksiyon 4Bh kullanılmaktadır. Bu fonksiyonun iki tane alt fonksiyonu da vardır. Bunlardan bir tanesi programı hafızaya yükler ve çalıştırır, diğeri ise sadece yükleme işlemini gerçekleştirir. İşte biz bu alt fonksiyonlardan ikincisini kullanarak çalıştırılacak programın kodunu hafızaya yükledik. Bu sayede kontrol tamamen switch.asm isimli programda kalmış oldu. (Böylelikle programın ne zaman çalıştırılıp durdurulacağına kararı ana programın vermesini sağlamış olduk.)

Burada yeri gelmişken bir farklılığı vurgulamakta fayda vardır. Daha önceki başlıklarda Com programın 100h adresinden başlaması gerektiğinden söz etmiş, (detay için bakınız 5.2 numaralı başlık) azami 64K'lık (FFFFh) alanda bulunması gerektiğine değinmiştik. Dolayısıyla programın offset değerinin bu 100h de göz önünde bulundurularak belirlenmesi gerekmektedir. Aynı 64k'lık alan hem CS, hem DS, hemde SS olarak kullanıldığına göre SP 'ninde bu dilimde kalması yani FFFFh degerini almasını gerekmektedir. (Stack yüksek adresden alçak adrese doğru ilerlemektedir.)

3.1.3 Programın çalıştırılması

İş hafızada yerini eksiksiz aldıktan sonra artık çalıştırılmaya hazır durumdadır. İşin çalıştırılıp çalıştırılmayacağına kontrolü elinde tutan program (switch.asm) karar verecek ve çalıştırmak istediği programın CS:IP değerine JMP (koşulsuz dallanma - intersegment jump) konutu ile giderek belirlenen quantum süresince işi çalıştıracaktır. Ancak bundan önce yapılması gereken bir iş daha vardır. Bu da o işin PSP'sinin aktif hale getirilmesidir. Bu amaçla yine Dos'un özel kesmelerinden olan (reserved interrupt) INT 21h fonksiyon 50h kullanılmaktadır. (PSP o işe ait birtakım bilgiler bulundurmaktadır. Daha detaylı bilgi için 5.1 numaralı

başlığa bakınız. Bu bilgiler her program için farklı değerlerdir. Bu amaçla her farklı program aktif hale getirildiğinde o programa ait PSP de aktif hale getirilmelidir. Bu INT 21h fonksiyon 50h ile gerçekleştirilmiştir.)

3.1.3.1 İsten işe geçilmesi

Bir işi yükleyip, belirlenen quantum süresince çalıştırıldıktan sonra sıra birdiğer işin (task/job) işlenmesine gelir. Bu durumda üç olasılık mevcuttur.

1-) Hafızada yüklü bir işin olması hali. Bu durumda mevcut iş çalıştırılmaya başlanır.

2-) Hafızada yüklü bir iş olmamasına rağmen kuyrukta işlenmesi gereken işin olması hali. Bu durumda daha önce belirlediğimiz aşamalardan geçmeli ve kuyruktan hafızaya yeni bir iş alınarak çalıştırılmalıdır.

3-) Hafızada başka bir iş yüklü olmadığı gibi kuyrukta da iş olmaması hali. Burada yüklenecek yeni bir iş olmadığı için işlenmesi quantum bittiği için kesilen işe tekrar yol verilir ve tamamlanması sağlanır.

3.1.3.2 İşin bitirilmesi

Eğer bir iş bitmiş ise (bu INT 20h yada INT 21h fonksiyon 00 yada 4Ch ile mümkün olabilir.) kontrol ana programa (switch.asm) geçmelidir. Bu amaçla INT 22h (Terminate Address) için bir program dilimi yazılarak orijinal INT 22h yerine konmuştur. Bu sayede çalıştırılmakta olan iş nasıl biterse bitsin kontrol INT 22h aktarılacak buda bizim yazdığımız programın devreye girmesini sağlayarak kontrolü ana programa aktaracaktır. Ana programda gerekli değişkenler değiştirilecek bunun yanı sıra tahsis edilen hafıza boşaltılacaktır. Daha sonra sıra

3.1.3.1 başlığında anlattığımız şekilde bekleyen işlere gelecek ve çalışma devam edecektir.

3.1.4 Programın bitirilmesi

Program, eğer kuyrukta işlenmek üzere bekleyen yada hafızaya yüklenmiş olan işler bitmiş ise sona ermelidir. Ancak bunun dışında kullanıcı da programın çalışmasını kesebilmelidir. Buna göre;

3.1.4.1 Normal şekilde bitirme

Yukarıda belirttiğimiz gibi eğer kuyruk ve hafızada işlenmek için bekleyen iş kalmamış ise program sona ermelidir. Ancak burada dikkat edilmesi gereken konu her şeyin yerli yerinde bırakılarak programın bitirilmesidir. Bu amaçla eğer kesme vektörleri değiştirilmiş ise orijinal değerleri yerine konmalıdır. Aksi yapılacak olursa sistemin çalışması aksayacak hatta sistem kitlenecektir.

Bu durumu göz önünde tutarak ana programın değiştirdiği kesme vektörlerini eski hallerine getirdik. (INT 22h ve INT 23h zaten programın sona ermesi ile otomatik olarak anlamlarını yitirmekteler. Dolayısıyla artık bunların eski haline getirilmesine gerek yoktur.) Bunu takiben ana programın PSP değerini INT 21h fonksiyon 50h ile aktif hale getirerek ana programı da sona erdirdik.

3.1.4.2 Ctrl-Break ile birtirme

Kullanıcı herhangi bir nedenden dolayı programı bitirmek isterse Ctrl-Break tuşlarına basarak programın çalışmasını kırarak Dos ortamına geri dönebilecektir.

Bu durumda hafızada bulunan tüm işlerden kendilerine

tahsis edilen hafıza geri alınacaktır. Tahsis edilen hafızanın geri alınabilmesi için INT 23h'e (Ctrl-Break) yerini alacak bir program dilimi yazılmış ve bu program yardımı ile kontrol ana programa aktarılmıştır. (Kullanıcı eğer Ctrl-Break ile programı kırarak çıkacak olursa programın işleyişi hakkında tutulmakta olan raporlar o ana kadar olan işleyişi belirleyeceklerdir. Ctrl-Break sırasında programların tamamlanmamış durumda oldukları göz önünde bulundurulursa üretilen data file'lı kullanılarak yapılacak istatistiklerin doğruluklarından şüphe etmek hiçde yanlış olmayacaktır. Üretilen istatistikler hakkında ayrıntılı bilgi 4. numaralı başlık altında verilmektedir.)

4.2 Zaman istatistikleri

Bu bölümde her ayın 1. günü için hesaplanan zaman istatistikleri verilmektedir. Bu istatistikler, ayın başında ve sonunda hesaplanmaktadır. Ayın başında hesaplanan istatistikler, ayın sonuna kadar geçerlidir. Ayın sonuna hesaplanan istatistikler, ayın başından itibaren geçerlidir.

4.3 Zaman istatistikleri

Bu bölümde her ayın 1. günü için hesaplanan zaman istatistikleri verilmektedir. Bu istatistikler, ayın başında ve sonunda hesaplanmaktadır. Ayın başında hesaplanan istatistikler, ayın sonuna kadar geçerlidir. Ayın sonuna hesaplanan istatistikler, ayın başından itibaren geçerlidir.

4.4 Zaman istatistikleri

Bu bölümde her ayın 1. günü için hesaplanan zaman istatistikleri verilmektedir. Bu istatistikler, ayın başında ve sonunda hesaplanmaktadır. Ayın başında hesaplanan istatistikler, ayın sonuna kadar geçerlidir. Ayın sonuna hesaplanan istatistikler, ayın başından itibaren geçerlidir.

4. TARTISMA VE SONUC

Programın nasıl çalıştığını anlamak ve bir takım istatistik değerleri elde etmek amacıyla kontrol programının içine gerekli bilgileri bir data file 'lına saklayan komutlar da ilave edilmiştir. Kontrol programı tarafından oluşturulan bu data file'li daha sonra Pascal programlama dilinde yazılmış bir programın yardımıyla çözümlenmekte ve raporlar üretilmektedir.

4.1 İşlerin işlem zamanlarının belirlenmesi

Yapılan değerlendirmelerden ilki hangi işin ne zaman işe başladığı ve işini ne zaman bitirdiğinin ve buna bağlı olarak ne süre ile işlemde tutulduğunun tespit edilmesidir.

4.2 Zaman istatistikleri

Burada her işin ne kadar süre ile işlendiği belirlenmekte ve harcanan toplam zamanın yüzde kaçının bu iş için harcandığı hesaplanmaktadır. Aynı zamanda o işin tamamlanması için kaç quantum ayrıldığı da belirlenmektedir.

4.3 Özel zaman istatistikleri

Yapılan basit zaman istatistiklerinin yanı sıra daha detaylı olarak şu değerlerde hesaplanmıştır.

4.3.1 Turn around Time

(Geri dönüş süresi)

Bu bir işin başlaması ile bitmesi arasındaki zaman değeridir. İşin beklediği sürede (yada kaybettiği süre elapsed time) buna dahildir.

4.3.2 Elapsed time

(Kayıp edilen süre)

Bir iş tek quantum'da yapması gerekenleri yapamıyor ve kendisine tekrar sıra gelmesini bekliyor ise belli bir zaman kaybediyor demektir. Bu süre içerisinde iş sadece işlemciyi kullanma sırasının kendisine gelmesini beklemekte yada diğer değişle bekleyerek zaman kaybetmektedir. İşte beklemekle kayıp edilen bu zamana elapsed time denir.

4.3.3 Weighted Turn around time

(Ağırlıklı geri dönüş süresi)

Bu değer işin gerçek çalışma süresinin kaç katı süre beklediğini gösteren bir değerdir. Bu değer mümkün olduğunca küçük olması gerekmektedir. (Hiç bir işin weighted turn around time değeri 1 den küçük olamaz.) Bunu sağlamanın bir takım yolları vardır. Bunlardan en etkin olanı kısa süreceği düşünülen işlerin uzun olacağı zannedilen işlerden önce işleme sokularak bekleme ile kayıp ettikleri zamanlarının azaltılmasıdır. Gerçi hangi işin ne kadar zaman alacağını tahmin etmek yada önceden belirlemek mümkün değildir. Ancak işler belli sürelerde çalıştırılıp herhangi bir şekilde işin ne kadar süreceği yada hangi işin daha kısa olduğu belirlenebilir. (Bu tip tekniklerden 2.3.1 başlığı altında bahis etmiştik)

5.1 PSP (Program Segment Prefix)

Her programın başında bulunan bu 256 bytelık bilgide Terminate address, Ctrl-Break Address gibi çıkış adreslerinin yanı sıra program argümanlarına ilişkin bilgilerde bulunmaktadır. Aşağıda PSP ve içerdiği bilgiler bulunmaktadır.

PSP (Program Segment Prefix)

0	1	2	3	4	5	6	7
INT 20h		Top of mem		Res.	Long Call to Dos		
8	9	A	B	C	D	E	F
Cont.		Term. add. IP		Term. add. CS		Ctrl-C IP	
10	11	12	13	14	15	16	17
Ctrl-C CS		Critical error exit add				Reserved	
18	19	2A	2B	2C	2D	2E	2F
Reserved				Segment add of Envir.		Reserved	
30 to 4F							
Reserved							
50	51	52	53	54	55	56	57
Dos Call		Reserved					
58	59	5A	5B	5C	5D	5E	5F
Reserved				Unopened FCB1			
60	61	62	63	64	65	66	67
Unopened FCB1							
68	69	6A	6B	6C	6D	6E	6F
Unopened FCB1				Unopened FCB2			
70	71	72	73	74	75	76	77
78	79	7A	7B	7C	7D	7E	7F
Unopened FCB2							
80	81	82	83	84	85	86	87
Parm len.	Command parameters starting with blank						
F8	F9	FA	FB	FC	FD	FE	FF
Command parameteres							

Burada hemen hatırlatmak isterizki herhangi bir programın başında yapılan şu işlemler

```
        PUSH    DS      ; PSP adresi
        SUB     AX,AX    ; Ax = 0
        PUSH    AX      ; IP olarak Ax
```

aslında program bittiğinde RET komutu ile geri alınmakta (önce ofset yani AX değeri ki bu değer 0 dır. Daha sonra DS değeri ki bu değer aslında PSP 'nin bulunduğu segmentin adresidir.) böylece DS:AX çifti PSP nin ilk 2 byte'ında bulunan INT 20h 'yi göstermektedir. Bu sayede program Dos ortamına dönebilmektedir.

Switch.Exe programının PSP'si aşağıdaki gibidir.

```
D 0 FF
1A2C:0000  CD 20 00 A0 00 9A F0 FE-1D F0 F4 02 65 16 2F 03
1A2C:0010  65 16 BC 02 65 16 2D 16-01 01 01 00 02 FF FF FF
1A2C:0020  FF FF FF FF FF FF FF FF-FF FF FF FF 25 1A 4C 01
1A2C:0030  9F 19 14 00 18 00 2C 1A-FF FF FF FF 00 00 00 00
1A2C:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
1A2C:0050  CD 21 CB 00 00 00 00 00-00 00 00 00 00 20 20 20
1A2C:0060  20 20 20 20 20 20 20 20-00 00 00 00 00 20 20 20
1A2C:0070  20 20 20 20 20 20 20 20-00 00 00 00 10 56 B8 01
1A2C:0080  00 0D 53 57 49 54 43 48-31 2E 45 58 45 0D 80 4E
1A2C:0090  E8 04 C7 46 F6 00 00 F6-44 2B 20 74 09 F6 44 2B
1A2C:00A0  10 74 03 2B C0 C3 C7 46-E6 01 00 F6 44 2B 20 74
1A2C:00B0  0C B8 02 00 50 9A 20 A9-FE FF E9 29 02 8D 46 EE
1A2C:00C0  16 50 2B C0 50 50 FF 36-22 07 FF 36 24 07 9A 98
1A2C:00D0  4B FE FF B8 FF FF 50 9A-4C 60 33 17 8D 45 08 1E
1A2C:00E0  50 8D 45 0A 1E 50 83 7D-08 9D 75 05 B8 01 00 EB
1A2C:00F0  03 B8 02 00 50 0E E8 E1-FD B8 FF FF 50 9A 56 60
```


5.2 COM Program Yapısı

Bunlar 64K (1 segment) büyüklüğündeki küçük ve basit programlardır. Code, Data ve Stack segment'in hepsi bu 64K'lık alan içerisinde bulunur. Buna rağmen Dos tüm boş bulunan hafızayı Com program için tahsis etmektedir. Ayrıca Com programlar kendilerine ayrılmış data alanını segment saklayıcılarının değerlerini değiştirmek suretiyle artırabilirler.

Com programlara ait en önemli özellik 100h adresinden başlamak zorunda olmasıdır. Bu başta kalan 100h 'lik kısım o segment içindeki PSP (Program Segment Prefix) için ayrılmıştır.

Com programlarda Stack (yığın) hakkında da birkaç cümle söylemekte fayda vardır. Com programlarda tüm segmentler aynı 64K lık alan içerisinde olduklarına göre yığına atılan değerler nereye konulacaktır. ?

Yığına saklanacak bilgilerin SS (stack segment) içinde konulacağı yeri SP (stack pointer) değeri belirler. Bu değeri Com programlarda FFFFh olarak almak gerekir. Bu şekilde yığına konulan değerler 64K lık alanın en altından azalan adreslere doğru yerleştirilmektedirler. (adres 2 byte olarak azalmakadır çünkü yığına konulan değerler word yani 2 byte'dır.)

Önemli hususlardan biride Com programlarda kullanılan procedure'lerin NEAR olmasıdır. Bu yine 64K sınırından kaynaklanmaktadır.

(Com programlar derlenip link edildiğinde 'no stack segment' şeklinde bir mesaj belirir. Bu Stack segment'in ayrı olarak tariflenmemesinden dolayı meydana gelir. Yazılan bir programın Com uzantısına sahip olabilmesi için EXE2BIN ile Exe'den Com haline getirilmesi gereklidir.)

5.3 EXE Program yapısı

Exe programlar Com programlara nazaran daha karmaşık bir yapıya sahiptir ler. Bir segmente sıkışmak yerine pekçok segment kullanırlar. Exe file'lerde Dos tarafından (özellikle 4Bh EXEC fonksiyonu tarafından) kullanılan bir başlık (header) kısmı vardır. Bu başlık programın hafıza ihtiyacı ve konumlandırılması ile ilgili birtakım bilgiler içerir. Exe programlar Com olanlar gibi 100h adresinden başlamak zorunda değildirler. Dolayısıyla 'relocatable' olarak tariflenirler. Exe programlar Com programlara nazaran daha esnek ve daha uzundurlar. Data, Code ve Stack olmak üzere bağımsız segmentlerden oluşurlar.

Exe programlarda procedure'ler NEAR olabileceği gibi FAR da olabilirler. Zira exe programlarda kullanılan procedure'ler 64K lık alan içerisinde olmak zorunda değildirler.

5.3.1 Exe file başlığı

Daha önce Exe programlar başlığı altında file başlığının 4Bh EXEC fonksiyonu için bir takım bilgiler içerdiğinde bahis etmiştik. Buradaki bilgiler 2 byte olup alçak anlamlı byte'ı önce olmak üzere yazılmışlardır.

1-) Offset 00h -> 5A4Dh ise file'lin Exe olup olmadığını belirler.

2-) Offset 02h -> File'lin uzunluğunu belirler (512 modülo)

3-) Offset 04h -> File'li uzunluğu (buna header bilgisi de dahildir.) Herbiri 512 byte olan kaç tane page olduğunu belirler.

4-) Offset 06h -> Relocation tablosunun kaç parçadan

oluşturduğunu belirler. (relocation tablosu adresi 18h de belirlidir.)

5-) Offset 08h -> File header'ın paragraf cinsinden uzunluğu (16 byte = 1 paragraf)

6-) Offset 0Ah -> Programın çalışması için gerekli asgari paragraf değeri (Bu sağlanmaz ise program çalışamaz)

7-) Offset 0Ch -> Programın kullanmak istediği azami miktarda paragraf değeri

8-) Offset 0Eh -> Programın başından itibaren SS (stack segment) değeri (offset)

9-) Offset 10h -> Program çalışmaya başladığı zaman SP (stack pointer) de olması gereken değer.

10-) Offset 12h -> Toplam kelime kontrol alanı

11-) Offset 14h -> Program çalıştırıldığı zaman IP (instruction pointer) de bulunması gereken değer.

12-) Offset 16h -> Yüklenen modül içinde CS (code segment) in paragraf olarak ötelenme değeri

13-) Offset 18h -> Exe file'da relocation table adresi

14-) Offset 1Ah -> Overlay ise 0 dır.

3.4 Bina/Box Arsenali

Exe dosyanın başlığı ise aşağıdaki gibidir;

HEXDUMP -- Release o3.1

Dumping File: C:SWITCH.EXE

```

0000: 4d 5a 0b 00 0d 00 08 00 20 00 00 00 ff ff 00 00
0010: 00 08 11 98 00 00 d4 00 1e 00 00 00 01 00 05 00
0020: d4 00 c7 01 d4 00 1e 03 d4 00 3b 03 d4 00 6a 04
0030: d4 00 ad 06 d4 00 7e 08 d4 00 a7 08 d4 00 00 00
0040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0100: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

```


5.4 Bios/Dos kesmeleri

Bu kesmeler ve görevleri şu şekildedir.

5.4.1 Bios kesmeleri

<i>Int</i>	<i>Function</i>	<i>Purpose</i>
00h		Divide-by-Zero Interrupt
01h		Single Step Interrupt
02h		Non-Maskable Interrupt
03h		Breakpoint Interrupt
04h		Arithmetic Overflow Interrupt
05h		Print Screen
06h		Reserved
07h		Reserved
08h		System Timer
09h		Keyboard
0Ah		Reserved
0Bh		Communications
0Ch		Communications
0Dh		Hard Disk Controller
0Eh		Floppy Disk Management
0Fh		Printer Management
10h		Video
	00h	Set Video Mode
	01h	Set Cursor Type
	02h	Set Cursor Position
	03h	Read Cursor Position and Configuration
	04h	Read Light Pen Position
	05h	Select Active Display Page
	06h	Scroll Window Up
	07h	Scroll Window Down
	08h	Read Character and Attribute
	09h	Write Character and Attribute
	0Ah	Write Character at Cursor
	0Bh	Set Color Palette
	0Ch	Write Graphics Pixel
	0Dh	Read Graphics Pixel
	0Eh	Write Text in Teletype Mode
	0Fh	Get Current Display Mode
	10h	Set Palette Registers
	11h	Character Generator
	13h	Write String
11h		Equipment Determination
12h		Memory Size Determination

<i>Int</i>	<i>Function</i>	<i>Purpose</i>
13h		Diskette
	00h	Reset Floppy Disk System.....
	01h	Get Floppy Disk System Status.....
	02h	Read Floppy Disk.....
	03h	Write Disk Sectors.....
	04h	Verify Disk Sectors
	05h	Format Disk Track.....
	06h	Reserved
	07h	Reserved
	08h	Return Disk Drive Parameters
	09h	Initialize Fixed Disk Table
	0Ah	Read Long Sector.....
	0Bh	Write Long Sector.....
	0Ch	Seek Cylinder.....
	0Dh	Alternate Disk Reset
	0Eh -14h	Reserved
	15h	Return DASD Type
	16h	Read Disk Change Line Status
	17h	Set DASD Type for Disk Format.....
	18h	Set Media Type for Format.....
14h		Asynchronous Communications
	00h	Initialize Communications Port.....
	01h	Write Character to Communications Port.....
	02h	Read Character from Communications Port
	03h	Request Communications Port Status.....
	04h	Extended Initialization (PS/2)
	05h	Extended Communications Port Control (PS/2) ..
15h		System Services
	00h	Turn On Cassette Motor.....
	01h	Turn Off Cassette Motor.....
	02h	Read Data Blocks from Cassette Drive
	03h	Write Data Blocks to Cassette Drive.....
	0Fh	Format Unit Periodic Interrupt.....
	21h	Power-On Self Test Error Log
	4Fh	Keyboard Intercept.....
	80h	Device Open
	81h	Device Close
	82h	Program Termination
	83h	Event Wait
	84h	Joystick Support.....
	85h	System Request Key Pressed

<i>Int</i>	<i>Function</i>	<i>Purpose</i>
	86h	Wait.
	87h	Move Block.
	88h	Extended Memory Size Determination.
	89h	Switch Processor to Protected Mode.
	90h	Device Busy.
	91h	Interrupt Complete.
	C0h	Return System Configuration Parameters.
	C1h	Return Extended BIOS Data Area Segment Address.
	C2h	Pointing Device BIOS Interface.
	C3h	Enable/Disable Watchdog Timeout.
	C4h	Programmable Option Select.
16h		Keyboard.
	00h	Read Keyboard Character.
	01h	Read Keyboard Status.
	02h	Return Keyboard Flags.
	05h	Write to Keyboard Buffer.
	10h	Get Keystroke.
	11h	Check Keyboard.
	12h	Get Keyboard Status Flags.
17h		Printer.
	00h	Write Character to Printer.
	01h	Initialize Printer Port.
	02h	Request Printer Port Status.
18h		Execute ROM BASIC.
19h		System Warm Boot.
1Ah		System Timer/Real-Time Clock Services.
	00h	Get Clock Counter.
	01h	Set Clock Counter.
	02h	Read Real-Time Clock.
	03h	Set Real-Time Clock.
	04h	Read Date from Real-Time Clock.
	05h	Set Date of Real-Time Clock.
	06h	Set System Alarm.
	07h	Disable Real-Time Clock Alarm.
1Bh		Ctrl-Break Handler Address.
1Ch		Timer Tick Interrupt.
1Dh		Video-Initialization Parameter Table.
1Eh		Disk-Initialization Parameter Table.
1Fh		Graphics Display Character Bit-Map Table.
70h		Real-Time Clock Interrupt.

5.4.2 Dos kesmeleri

Table DOS.1 continued

Int	Func (AH)	Subfunc (AL)	Purpose
1B			Reserved
1F			Reserved

Table DOS.1. DOS Function Chart

Int	Func (AH)	Subfunc (AL)	Purpose
20			Terminate program
21	00		Terminate program
	01		Keyboard input with Echo
	02		Display output
	03		Auxiliary input
	04		Auxiliary output
	05		Printer output
	06		Direct console I/O
	07		Direct STDIN input
	08		STDIN input
	09		Display string
	0A		Buffered STDIN input
	0B		Check STDIN status
	0C		Clear buffer and input
	0D		Reset disk
	0E		Select disk
	0F		Open file (FCB)
	10		Close file (FCB)
	11		Search for first entry (FCB)
	12		Search for next entry (FCB)
	13		Delete file (FCB)
	14		Read sequential file (FCB)
	15		Write sequential file (FCB)
	16		Create file (FCB)
	17		Rename file (FCB)
	18		Reserved
	19		Get default drive
	1A		Set DTA address
	1B		Get allocation table information
	1C		Get allocation table information for specific drive
	1D		Reserved

Table DOS.1 continued

<i>Int</i>	<i>Func (AH)</i>	<i>Subfunc (AL)</i>	<i>Purpose</i>
	1E		Reserved
	1F*		Get default disk parameter block
	20		Reserved
	21		Random file read (FCB)
	22		Random file write (FCB)
	23		Get file size (FCB)
	24		Set random record field (FCB)
	25		Set interrupt vector
	26		Create PSP
	27		Random block read (FCB)
	28		Random block write (FCB)
	29		Parse file name
	2A		Get system date
	2B		Set system date
	2C		Get system time
	2D		Set system time
	2E		Set verify flag
	2F		Get DTA address
	30		Get DOS version number
	31		Terminate and stay resident
	32*		Get drive parameter block
	33		Get/set Ctrl-Break flag
	34*		Return address of InDOS flag
	35		Get interrupt vector
	36		Get free disk space
	37*		Get/set switchchar
	38		Get/set country information
	39		Create subdirectory
	3A		Remove subdirectory
	3B		Set directory
	3C		Create/truncate file (handle)
	3D		Open file (handle)
	3E		Close file (handle)
	3F		Read file or device (handle)

<i>Int</i>	<i>Func (AH)</i>	<i>Subfunc (AL)</i>	<i>Purpose</i>
	40		Write to file or device (handle)
	41		Delete file
	42		Move file pointer
	43		Get/set file attributes
	44	00	Get device information
		01	Set device information
		02	Character device read
		03	Character device write
		04	Block driver read
		05	Block driver write
		06	Get input status
		07	Get output status
		08	Block device removable?
		09	Block device local or remote?
		0A	Handle local or remote?
		0B	Set sharing retry count
	45		Duplicate handle
	46		Force duplicate handle
	47		Get current directory
	48		Allocate memory
	49		Release memory
	4A		Modify Memory Allocation
	4B	00	Execute program (EXEC)
		03	Load overlay
	4C		Terminate with return code
	4D		Get return code
	4E		Search for first match
	4F		Search for next match
	50*		Set PSP segment
	51*		Get PSP segment
	52*		Get disk list
	53*		Translate BPB
	54		Get verify flag
	55*		Create PSP

Table DOS.1 continued

<i>Int</i>	<i>Func (AH)</i>	<i>Subfunc (AL)</i>	<i>Purpose</i>
	56		Rename file
	57	00	Get file date and time
		01	Set file date and time
	58*	00	Get allocation strategy
		01	Set allocation strategy
	59		Get extended error information
	5A		Create temporary file
	5B		Create file
	5C		Set file access
	5D*	06	Get critical error flag address
	5E	00	Get machine name
		02	Set printer setup
		03	Get printer setup
	5F	02	Get redirection list entry
		03	Redirect device
		04	Cancel redirection
	60		Reserved
	61		Reserved
	62		Get PSP address
	63	00	Get system lead byte table
		01	Set interim console flag
		02	Get interim console flag
	64		Reserved
	65		Get extended country information
	66	01	Get global code page
		02	Set global code page
	67		Set handle count
	68		Flush buffer
22			Terminate address
23			Ctrl-C interrupt vector
24			Critical error vector
25			Absolute disk read
26			Absolute disk write
27			Terminate and stay resident

5.4.3 Kullanılan Komutlar ve Aciklamalari

INT 28h (System Library)

Sistem kütüphanesi tarafından sunulan yaklaşık 18 tane servis çağırılmaktadır. Sistem kütüphanesi/görevleri için gerekli olan bu kütüphane ayrıca ICH (timer/tick) zamanlama ve çağırılmaktadır. Aşağıdaki tabloda kullanılan komutların amacı açıklanmıştır.

Int	Func (AH)	Subfunc (AL)	Purpose
28h			Keyboard busy loop
29h			Fast putchar
2Fh	01	00	Print installation check
		01	Print submit file
		02	Print remove file
		03	Print remove all files
		04	Print hold queue/get status
		05	Print restart queue
	B7	00	APPEND installation check
		01h	APPEND
33h			Microsoft mouse
5Ch			NETBIOS interface
67h	40		Get EMM status
	41		Get page frame segment
	42		Get number of pages
	43		Get handle/allocate memory
	44		Map memory
	45		Release handle and memory
	46		Get EMM version
	47		Save mapping context
	48		Restore mapping context
	49		Reserved
	4Ah		Reserved
	4Bh		Get number of EMM handle
	4Ch		Get pages owned by handle
	4Dh		Get pages for all handles
	4Eh	00	Get page mapping registers
		01	Set page mapping registers
		02	Get/set page mapping registers
		03	Get size of page mapping array

5.4.3 Kullanılan kesmler ve açıklamaları

INT 08h (System timer)

Sistem saati tarafından saniyede yaklaşık 18.2 defa çağırılmaktadır. Sistem saat/gün değerleri için gerekli olan bu kesme ayrıca 1Ch (timer tick) kesmesini de çağırmaktadır. Bu kesme diğer kesmelere nazaran daha önceliklidir. (IRQ0) Bu kesme aynı zaman da diskin çalıştırılıp/durdurulması içinde gereklidir.

INT 20h (program terminate)

Bu kesme programı bitirmek amacıyla kullanılan kesmedir. PSP'nin ilk iki byte'ında (programı bitirmek amacıyla) da bulunur. Programın bitirilmesi ve tahsis edilen hafızanın boşaltılması işlerinin yanı sıra aşağıdaki işlemleride gerçeklemektedir.

- * PSP'nin 0Ah adresinden terminate handler vektörünü yükler.
- * PSP'nin 0Eh adresinden Ctrl-C vektörünü yükler.
- * PSP'nin 12h adresinden Critical-error handler'i yükler.
- * File buffer'larını temizler.

INT 21h Fonksiyon 00h (Program terminate)

Programın bitirilmesini sağlar. İşlem itibariyle INT 20h ile aynıdır.

INT 21h Fonksiyon 09h (Display string)

Bu fonksiyon ekrana kullanıcının tanımladığı mesajı yazdırmak amacıyla kullanılır. Mesajın sonunu belirlemek üzere Ascii 24h değeri (\$) kullanılmaktadır.

Kesmenin kullanılabilmesi için;

Ah <- 09h

DS:Dx <- yazılacak karakter dizisinin başını gösterir.

Bu kesme işlendikten sonra geriye herhangi bir bilgi dönmez.

INT 21h Fonksiyon 25h (Set interrupt vector)

Güvenli bir şekilde kesme vektörünün değiştirilmesini sağlar.

Kesme vektörleri tablo halinde hafızanın en altındaki 1024 byte'ta tutulmakta ve her vektör 4 byte ile temsil edilmektedir. Bu bilginin 2 byte'ı segment, kalan 2 byte'ıda offset değerini taşır. Bir kesme vektörünün değiştirilmesi için bulunduğu hafıza alanına gidilip burada bulunan bilgiler değiştirilebilir. Ancak kesme vektörünün tam olarak değiştirilebilmesi için hafızada o vektörü gösteren 4 byte'lık bilginin tam olarak (herhangi bir şekilde bölünmeden) değiştirilmesi gerekir. Eğer kesme vektörünün ilk 2 byte'ını değiştirdikten hemen sonra değiştirilmek istenen kesme herhangi bir şekilde üreyecek olursa vektör adresi tam olarak değişmediğinden dolayı program belli olmayan bir yere gider ve muhtemelen sistemin kitlenip kalmasına neden olur. Bunu engellemek üzere bu 4 byte'lık alan tam olarak değiştirilene kadar herhangi bir şekilde kesmelerin maskelenmesi gerekmektedir. Buna göre değiştirme başlamadan hemen önce CLI komutu ile kesmelere verilen izini kaldırıp işlemi yapmalı, ve işlemin hemen sonunda STI komutu ile kesmelere izin verilmelidir. Görüldüğü gibi bu şekilde kesme vektörünün değiştirilmesi hiçde kolay bir iş değildir. Üstelik bir takım eksik yönlerinin de olduğu rahatlıkla söylenebilir. İşte INT 21h Fonksiyon 25h bu tip zorlukları ortadan kaldırmak üzere tasarlanmış bir Dos fonksiyonudur.

Burada yeri gelmişken kesme vektörlerinin değiştirilmesi durumunda çıkabilecek sorunlardan da bahis etmek isteriz. Her ne kadar INT 22h, 23h, 24h programın bitmesi ile eski değerlerini alsalarda program akışı içerisinde değiştirilen diğer kesme vektörleri eski değerlerini geri alamazlar. Buda program çalıştırılıp bittikten sonra değiştirilen vektörler eğer eski halleri yerine konmaz ise vektörlerin olmadık alanları göstermesi gibi bir problemi karşımıza çıkartır. Eğer bu değiştirilmiş kesmelerden bir tanesi üreyecek olursa aslında olmayan bir kod alanına dallanacak ve sistemin kitlenmesine yada aslında yapması gereken işlemden farklı bir işlem yapmasına neden olacaktır.

Ortaya çıkan bu sevimsiz durumu ancak şu adımlardan geçerek ortadan kaldıracabiliriz.

* Değiştirilmesi istenilen kesme vektörünün değeri INT 21 Fonksiyon 35h ile alınıp saklanmalıdır. (Bu kesme ileride anlatılacaktır.)

* Kesme vektörü INT 21h Fonksiyon 25h kullanılarak istenilen program dilimini gösterecek şekilde değiştirilmelidir.

* Program yapması gereken işlemleri yerine geçirir.

* Program bitirilmeden hemen önce ilk adımda INT 21h Fonksiyon 35h yardımıyla saklanan vektör değeri INT 21h Fonksiyon 25h kullanılarak tekrar set edilmelidir.

Bu şekilde program işlemeye başlamadan önce bulduğu herşeyi yerli yerine bırakarak işlemi tamamlamış olur.

Kullanılırken yapılması gereken işlemler şu şekildedir;

Ah <- 25h

DS:DX <- set edilmesi istenilen program diliminin adresi

Al <- set edilmesi istenen kesmenin numarası

Bu kesme işlendikten sonra geriye herhangi bir bilgi dönmez.

INT 21h Fonksiyon 2Ch (Get system time)

Bu kesme sistem saatinin okunmasını sağlamaktadır. Bu kesme yazdığımız program içerisinde hangi işin ne zaman çalıştırıldığını belirlemek amacıyla kullanılmıştır.

Kullanılırken yapılması gereken işlemler şu şekildedir;

Ah <- 2Ch

Bu kesmenin kullanılması ile şu değerler geri dönmektedir;

Ch -> Saat (0-23)

Cl -> Dakika (0-59)

Dh -> Saniye (0-59)

Dl -> Salise (0-99)

INT 21h Fonksiyon 34h (InDos Flag)

Bu Dos tarafından kullanılan bir bayraktır ve Dos'un fonksiyonlarından herhangi bir tanesi yürütülmekte iken set edilir. Dos işletim sistemi tasarlanırken tek kullanılıcılı olarak tasarlandığından reentrant değildir. Bu nedenle Dos'un herhangi bir fonksiyonu kullanılmadan önce Dos'un herhangi bir fonksiyon yerine getirip/getirmediğinin kontrol edilmesi gerekmektedir.

Kesme şu şekilde kullanılmaktadır;

Ah <- 34h

Kesmeden geri dönen değerler ise;

ES:BX -> InDos bayrağının bulunduğu alana işaretçi

INT 21h Fonksiyon 35h (Get interrupt vector)

Kesme vektörünün değerinin okunmasını sağlayan fonksiyondur. Bu fonksiyon kesme vektörünün en doğru şekilde okunmasını sağlar ve genelde INT 21h fonksiyon 25h ile birlikte kullanılır.

Bu kesme şu şekilde kullanılmaktadır;

Ah <- 35h

Al <- Okunması istenen vektörün numarası

Dönüş değerleri şu şekildedir;

ES:BX -> Kesme vektörüne işaret eder.

INT 21h Fonksiyon 3Ch (Create a file)

Yeni bir dosya açılması amacıyla kullanılır. Eğer aynı isimde daha önceden açılmış bir dosya varsa bunun içindeki bilgiler kayıp olacaktır.

Bu kesme şu şekilde kullanılmaktadır;

Ah <- 3Ch

Cx <- Dosyanın niteliği

DS:DX <- Dosyanın ismini gösterir. (ASCIIIZ)

Dönüş değerleri ise şöyledir;

Ax -> Dosyayı belirleyen bir işaret (File handle). Eğer CF set edilmiş ise hata kodu değeri.

INT 21h Fonksiyon 3Dh (Open file)

Bu kesme, belirlenen isimde bir dosya açmak amacıyla kullanılmaktadır. (Yazdığımız programda oluşturduğumuz bilgi dosyasının açılması için kullanılmıştır.)

Bu kesme şu şekilde kullanılır.

Ah <- 3Dh

Al <- Açma şekli

DS:DX <- Açılmak istenen dosyanın ismi (ASCIIIZ)

Dönüş değerleri ise şöyledir.

Ax -> Dosyayı belirleyen bir işaret (File handle). Eğer CF set edilmiş ise hata kodu değeri.

INT 21h Fonksiyon 3Eh (Close File)

Daha önceden açılmış olan dosyanın kapatılması için kullanılan kesmedir. (programda açılan bilgi dosyasını kapatmak amacıyla kullanılmıştır.)

Bu kesme şu şekilde kullanılmaktadır;

Ah <- 3Eh

Bx <- Dosyayı belirleyen işaret (File Handle)

Dönüş değerleri ise şu şekildedir;

Ax -> CF set edilmiş ise hata kodu aksi halde bir değer ifade etmez.

INT 21h Fonksiyon 3Fh (Read File)

Bu fonksiyon INT 21h Fonksiyon 3Dh ile açılmış olan dosyadan bilgi okumak amacıyla kullanılır. Dosyadan okunan bilgi hafızada buffer olarak kullanılan bir alanda saklanmaktadır. (Programda işlerin dosya başlıklarındaki bilgileri okumak amacıyla kullanılmıştır.)

Bu kesmenin kullanılması şu şekildedir;

Ah <- 3Fh

Bx <- Dosya işaretçisi

DS:DX <- Buffer olarak kullanılan hafıza alanının adresi

Cx <- Okunması istenen byte sayısı

Bu kesmenin dönüş değerleri ise şu şekildedir;

Ax -> Okunan byte sayısı, CF set edilmiş ise hata kodu

INT 21h Fonksiyon 40h (Write File)

Belirlenen miktarda bilgiyi açık olan dosyaya, dosya işaretçisinin (file pointer) gösterdiği yerden itibaren yazar. (programda oluşturulan bilgi dosyasına gerekli değerleri yazmak amacıyla kullanılmıştır.)

Bu kesme şu şekilde kullanılır;

Ah <- 40h

Bx <- Dosya işaretçisi (file handle)

DS:Dx <- Bilginin yazılacağı adres

Cx <- Yazılacak bilginin byte cinsinden uzunluğu

Bu kesme sonucunda oluşan dönüş değerleri ise;

Ax -> Yazılan byte sayısı, CF set edilmiş ise hata kodu

INT 21h Fonksiyon 48h (Allocate memory)

Bir hafıza bloğunun tahsis edilmesi amacıyla kullanılır. Dönüş değeri o hafıza bloğunun başına bir işaretçidir. Her zaman istenilen miktarda hafıza tahsis edilemez. BU durumda tahsis edilebilecek en büyük hafıza bloğunun adresi geri döner. (Programda işler için hafıza tahsisi bu fonksiyon yardımı ile yapılmıştır.)

Bu kesme şu şekilde kullanılır;

Ah <- 48h

Bx <- Paragraf cinsinden gerek duyulan hafıza miktarı.

Dönüş değerleri ise şu şekildedir;

Ax:0 -> Tahsis edilen hafıza alanını işaretler.
Ax -> CF set edilmiş ise hata kodu
Bx -> Mevcut olan en büyük alanın kaç paragraf olduğuna dair bilgi

INT 21h Fonksiyon 49h (Free memory)

Belirlenen hafıza bloğunu boş olan hafıza alanına eklemek üzere kullanılan Dos fonksiyonudur. (Programda iş sona erdikten sonra ona ayrılan alanın tekrar kullanılmak üzere boş olarak işaretlenmesi amacıyla kullanılmıştır.)

Bu kesme şu şekilde kullanılmaktadır;

Ah <- 49h
ES <- Boşaltılması istenen hafızanın segment adresi

Dönüş değerleri ise;

Ax -> CF set edilmiş hata kodu

INT 21h Fonksiyon 4Ah (Set block)

Daha önceden hafıza tahsis edilmiş olan hafıza alanlarının büyültüp/küçültmek amacıyla kullanılır. (Programda kontrol işlemini yerine getiren kısmın hafızada kendisine yetecek kadar alana yerleştirilmesi amacıyla kullanılmıştır.)

Bu kesme şu şekilde kullanılmaktadır;

Ah <- 4Ah
Bx <- İstenilen büyüklük (paragraf cinsinden)
Es <- Büyüklüğü değiştirilmesi istene hafıza bloğunun segment adresi.

Dönüş değerleri ise;

Ax -> CF set edilmiş ise hata kodu

BX -> Müsait olan azami hafıza miktarı.

INT 21h Fonksiyon 4Bh (Load/Execute program)

Bu kesme sayesinde, bir program bir değerinin kontrolü altında kalmak şartıyla çalıştırılır. Aslında bu fonksiyon iki tane de alt fonksiyona sahiptir. Bunlardan bir tanesi programı sadece hafızaya yüklemekle yetinirken, diğeri hem yükler hemde programı çalıştırır. (Programda biz bunlardan ilki olan yükleme işlemini gerçekleştiren alt fonksiyonu kullandık. Bu sayede programın ne zaman çalıştırılacağına karar verme hakkı ana programda kalmıştır.)

Bu kesme şu şekilde kullanılmaktadır;

Ah <- 4Bh

Al <- 00h Yükleyip çalıştırmak için

<- 03h Sadece yüklemek için

DS:Dx <- Programın ismini ve nerede bulunduğunu belirten dizi (ASCIIZ)

ES:Bx <- Parametre bloğuna işaretçi

Dönüş değerleri ise şöyledir;

Ax -> CF set edilmiş ise hata kodu

INT 21h Fonksiyon 4Ch (Terminate a process-Exit)

Program bu kesme kullanıldığında bir dönüş değeri ile ana programa dönmektedir. INT 20h ve INT 21 Fonksiyon 00h de yapılan işlemlere ek olarak program termination handler address'e gider.

Bu kesme şu şekilde çalışmaktadır;

Ah <- 4Ch

Al <- Dönüş değeri

Bu kesme ayrıca bir değer ile dönmez. Zaman bu adreste
bulunan değer 50h'ye 50h adresli gözüne kopyalanır.

Program INT 21h Fonksiyon 50h (Set PSP Segment)

Çalıştırılmakta yada çalıştırılacak olan programın
PSP'sinin adresini set etmek amacıyla kullanılmaktadır.
Bunun kullanılması sayesinde context switching işlemi
yapılabilmektedir. Bu esnada çalışmakta olan programın kontrolü
burada belirlenen adrese aktarılır.

bu kesme şu şekilde kullanılmaktadır;

Ah <- 50h

Bx <- Yeni PSP'nin segment adresi

Bu kesmenin dönüş değeri yoktur.

INT 21h Fonksiyon 55h (Create PSP)

Belirlenen segment adresinde bir PSP yaratmak amacıyla
kullanılır. Bu kesme aslında INT 21h Fonksiyon 26h'e
benzemekle birlikte fonksiyon 26h'in yaptığı gibi ana
programın PSP'ini kopyalamak yerine ondan farklı olarak
yeni bir PSP yaratmaktadır. (PSP, INT 21h fonksiyon 4Bh'in
kullanılması içinde gereklidir.)

Bu kesme şu şekilde kullanılır; Ah <- 55h

DX <- PSP'nin yaratılacağı segment adresi

Bu kesmenin dönüş değeri yoktur.

INT 22h (Terminate address)

Bu tam anlamıyla bir kesmeden ziyade bir adres
değeridir. Sadece çalışmakta olan program sona erdiği zaman
kontrolün verileceği kodun başlangıç adres değeridir.

Bir program hafızaya yüklendiği zaman bu adresde bulunan deger PSP'nin 0Ah adresli gözüne kopyalanır. Program sona erdiğinde de buradan geri alınır.

INT 23h (Ctrl-C)

Buda INT 22h gibi kesmeden ziyade bir adres değeridir. Ctrl-C yada Ctrl-Break oluştuğu zaman programın kontrolü burada belirlenen adrese aktarılmaktadır.

5.5 Kullanılan programların çıkışları

5.5.1 Switch.Asm

```
-----  
; This program is a task switcher .  
; By using the system timer interrupt 08h  
; it runs given tasks during the specified  
; quantum. (this quantum can be changed in the  
; program)  
; Program can handle several programs with  
; extention .COM and .EXE  
;  
; Name Of the program : SWITCH.ASM  
; Macro used           : MACRO.LIB  
;                       : EXTRA.LIB  
; Debuging Macro       : DEBUG.LIB  
; Created              : 07/12/90  
; Written By           : Ahmet Tevfik İNAN  
;  
; (c) All rights of this program is reserved by A.Tevfik İNAN  
; (c) Kopya hakkı ve diğer bütün hakları A.Tevfik İNAN'a aittir.  
-----  
  
Title Task switcher  
Include Debug.Lib  
Include Extra.Lib  
Include Macro.Lib  
  
Max_Task Equ 7 ;Max Task number to be run at once  
Quantum Equ 15 ;each time period given to any job  
; (will be multiple of 55ms)  
  
Stack Segment Para Stack 'Stack'  
Db 2048 Dup (?)  
; 2k of stack area defined  
  
Stack Ends  
Data Segment Para 'Data'  
REGS Struc ;Structure for registers
```



```

aax      Dw  ?
bbx      Dw  ?
ccx      Dw  ?
ddx      Dw  ?
ssi      Dw  ?
ddi      Dw  ?
ssp      Dw  ?
bbp      Dw  ?
iip      Dw  ?
sss      Dw  ?
ccs      Dw  ?
dds      Dw  ?
ees      Dw  ?
flags    Dw  ?
REGS      ENDS

T1        REGS    <>      ;Memory area to save registers of Task #1
T2        REGS    <>      ;Memory area to save registers of Task #2

;-----
; Task declarations
;-----

Tick_C    Dw  0          ;Initialize tick count
Task_Id    Db  0          ;Can be 1 or 2 to show the active task #
                        ;0 mean uninitialized

Task       Db  0Fh,0Fh,0Fh ;Shows task sequence number(index)
                        ;can be 0..9 or 0Fh (free)

Task_index Db  0          ;index for task_pointer and Task_Status
Task_Status Db  Max_Task dup (4)
                        ;Task status is as follows
                        ; 0 --> Terminated
                        ; 1 --> Waiting
                        ; 2 --> Not started
                        ; 3 --> Active
                        ; 4 --> No task to perform

Alloc_Task Db  0          ;Shows thw number of allocated tasks

;-----

```



```

; The old address of Int 08h, 22h
; and the address of InDos flag
;-----
Old_int08 Label Dword
Off_08h Dw ?
Seg_08h Dw ?
Old_int22 Label Dword
Off_22H Dw ?
Seg_22h Dw ?
Dos_OK Label Dword ;area to save the DosOK flag address
Off_Dos_F Dw ?
Seg_Dos_F Dw ?
;-----
; Message descriptions
;-----
Msg1 Db 'Parsing Arguments...',10,13,'$'
Msg2 Db 'Parsing terminated.',10,13,'$'
Msg3 Db 'Initializing Vector Table.',10,13,'$'
Msg4 Db 'INT 21H Initialization completed.',10,13,'$'
Msg5 Db 'INT 08H Initialization completed.',10,13,'$'
Msg6 Db 'INT 22H Initialization completed.',10,13,'$'
Msg7 Db 'INDOS flag address found.',10,13,'$'
Msg8 Db 'Initialization completed.',10,13,'$'
Msg9 Db 'Resident program installed .',10,13,'$'
Msg10 Db 'INT 23H Initialization completed.',10,13,'$'
ParseErr1 Db 'Argument(s) not found.',10,13,'$'
Usage Db 'Switch [path:\]filename [[path:\]filename.]',10,13,'$'
AllocErr1 Db 'File open error..',10,13,'$'
AllocErr2 Db 'Setblock error..',10,13,'$'
AllocErr3 Db 'Read error..',10,13,'$'
AllocErr4 Db 'Close file error..',10,13,'$'
AllocErr5 Db 'Allocation error..',10,13,'$'
AllocErr6 Db 'Can t load program ..',10,13,'$'
AllocErr7 Db 'Can t free the allocated memory ..',10,13,'$'
Ack1 Db 10,13,'INT 08H Restored back.', '$'

```



```

Ack2      Db 10,13,'INT 22H Restored back.','$$'
Ack3      Db 10,13,'INT 21H Restored back.','$$'
Ack4      Db 10,13,'Program terminated normally.','$$'
Result_Err1 Db 10,13,'Result file open error ..','$$'
Result_Err2 Db 10,13,'Result file seek operation error ..','$$'
Result_Err3 Db 10,13,'Result file write operation error ..','$$'
licence1   db 10,13,'TASK SWITCHER (c) ','$$'
licence2   db 10,13,'All Right of this program is reserved','$$'
licence3   db 10,13,'A. Tevfik iNAN',10,13,10,13,'$$'

```

```

;-----

```

```

; Parametre block description for

```

```

; set block operation or laod

```

```

; end execute procedure

```

```

;-----

```

```

PARM      Struc

```

```

Segm      Dw ?          ;Start segment of the task allocated

```

```

Reloc     Dw ?          ;relocation factor which effects the

```

```

PARM      ENDS          ;displacement in the allocated task

```

```

Task_1    PARM <>       ;Tasks parametter descriptions

```

```

Task_2    PARM <>

```

```

Env       Dw ?          ;the address of comspec etc.

```

```

;-----

```

```

; Data area to store Stack segment

```

```

; and stack pointer to be able to

```

```

; return back to the main program

```

```

; when we start to execute a task

```

```

; the original ones will be destroyed

```

```

;-----

```

```

Old_Ss    Dw ?

```

```

Old_Sp    Dw ?

```

```

;-----

```

```

; The arguments declarations

```

```

;-----

```

```

File_Names Db Max_Task dup ( 30 dup (0) )

```

```

;data area for the argumnets

```



```

Len            Db 30            ;the length for each path and file name
File_Pointer   Dw  ?            ;Points the offset of the specifeid file
File_Handle    Dw  ?            ;Keeps the handle of the file
File_Buff      Db 128 Dup (00)  ;buffer area to store data from file

Const          Dw 512           ;
;-----
; the area to save the exit address
;-----

Cikis label Dword ;Points to the first byte of PSP (INT 20h)
cik_off dw 0      ;to terminate the main program and return
Psp      Dw ?     ;to DOS
;-----
; The declarations concerning the result
; file generation
;-----

Kayit      Struc
F_Task     Db  ?      ;Task number
F_Name     Db 30 dup (0);Name of the task file name
F_Start    Db 4 dup (?) ;Task start time
F_Stop     Db 4 dup (?) ;Task stop time
F_Stat     Db  ?      ;Task Status
Kayit      Ends

Rec Kayit < > ;Record for task
Result_file Db 'Result.ref',0
;-----
;Name of the result file to store
Result_handle Dw ? ;Handle for result file
Result_Rec_Len Dw 40 ;Lenght of the record which will be
;-----
;written to the result file
Result_File_Pointer Dw 0
;-----
; Allocation variables
;-----

Top_mem     Dw  ?      ;Top of the available memory (RAM)
Alloc_Area1 Dw  ?      ;Those two variable will be used if

```



```

;we want to create already defined
;areas for the tasks.
Alloc_Area2      Dw  ?
Program_Area_Para_Len  Dw  ?
;Lenght of the program in paragraph
Alloc_Size      Dw  ?
;It also used in fixed allocation
;area usage to determine the allocation
;size
User_exit        Db  0
Data      Ends
Code      Segment Para 'Code'

```

```

;-----
; Install is the first procedure running to
; to parse the argument string using PSP and
; to make necessary vector table changes
; The parsing part of the procedure uses the
; values in PSP at starting from ES:80h to
; ES:FFh . It acceptst the path and the file
; name which will be executed.
;-----

```

```

Install Proc      Far
    Assume  Cs:Code,Ds:Data,Ss:Stack
    Push    Ds
    Sub     Ax,Ax
    Push    Ax
    Mov     Ax,Data
    Mov     Ds,Ax
    Mov     Psp,Es    ;holds the PSP address
    Cls      ;Macro which clears the screen
    Printch  Licence1 ;Print screen the licence lines
    Printch  Licence2
    Printch  Licence3
    Mov     Es,Psp
    Mov     Ax,Es:[2Ch]

```



```

Mov     Env,Psp      ;Es:2Ch holds the segment address of
                    ;environment where comspec ans sets are
                    ;save the environment area address
Mov     Env,Ax,Ax
;-----
; Adjust program size to minimum
;-----
Shr     Ax,1         ;When we run a program it allocates hole
Sub     Ax,10h       ;memory. To be able to allocates other
Mov     Alloc_Size   ;tasks to the memory we need to fix the
                    ;size of the program and obtain some
                    ;free memory for the tasks which will be
                    ;loaded later on to run.
Mov     Bx,Offset Endoffset
Mov     Cx,0007h     ;To be able to use setblock func. Bx must
Mov     Bx,0h        ;contain the new size of the program in
Sub     Cx,Cx        ;paragraf. (1 paragraph = 16 bytes)
Mov     Cx,Bx        ;now bx contains the length of CS in bytes
Mov     C1,4         ;Cx contains arguments length
Shr     Bx,C1        ;Bx contains length of CS in paragraph
Inc     Bx            ;Usually we take 1 paragfaph more
Mov     Cx,CS        ;to calculate the length of the program
                    ;part between CS and PSP(the very begining
                    ;of the program)
Mov     Ax,Es
Sub     Cx,Ax        ;CS has a higher memory address then the
                    ;PSP .So sub Cs-Es give us the paragraph
                    ;size of the program between CS and PSP
Add     Bx,Cx        ;add this to the CS paragraph size
Inc     Bx
Mov     Program_Area_Para_Len,Bx
Set_Block_Size:
Mov     Ah,4Ah       ;Set block size
                    ;Size will be in BX (paragraph)
Int     21h          ;Call Dos

```



```

Mov     Es,Psp
Mov     Ax,Es:[02h]
;-----
;Get the top of the memory segment address
Mov     Top_Mem,Ax
Mov     Bx,Es
Add     Bx,Program_Area_Para_Len
Sub     Ax,Bx
Shr     Ax,1
Sub     Ax,10h
Mov     Alloc_Size,Ax
;-----
; Parse arguments of the program
;-----
Mov     Di,0h      ;Index for file_names array
Mov     Si,0082h   ;Start add. of arg list
Mov     Bx,0h      ;Used as an argument counter
Sub     Cx,Cx
Mov     Cl, Byte Ptr Es:[Si-2]
;Cx contains arguments length
Cmp     Cl,00h     ;If arg. length is 0 end
Je      Err_Of_Parse
Mov     Task_Status[0],2
Start_To_Parse:
Mov     Al, Byte Ptr Es:[Si]
;To parse control the blank (20h) char.
Cmp     Al,20h
Je      Next_Argument
Cmp     Al,0dh     ;Is it the end of the argument line
Jne     Continue_Parse
;terminated by CR (0Dh) char.
Mov     File_Names[Di],0h
;Put a null terminator for ASCIIZ
Jmp     End_Of_Parse
Err_Of_Parse:
PRINTCH ParseErr1 ;Print screen the error message and

```



```

    PRINTCH Usage      ;the usage of the program
    Ret               ;Return to Dos

Continue_Parse:
    Mov     File_Names[Di],Al
    Mov     Ax,2522h    ;Copy the value to the file names
    Inc     Di          ;string and increment the indexes
    Inc     Si
    Loop    Start_To_Parse

Next_Argument:
    Mov     File_Names[di],0h
    ;put a null terminator for ASCIIZ
    Inc     Bx
    Mov     Di,Bx
    Mov     Task_Status[Di],2
    Cmp     Bx,Max_Task
    ;Accept Max_Task arguments
    Je      End_Of_Parse
    Mov     Ax,Bx
    Mul     Len          ;Find the effective address of the next
    Mov     Di,Ax        ;argument
    Inc     Si
    Loop    Start_To_Parse

End_Of_Parse:
    PRINTCH Msg2        ;Print screen message
    ;-----
    ; Get the original int 22h and change vector
    ; 22h ==> Terminate program
    ; Control is transferred to the address at this
    ; interrupt location when the program terminates
    ;-----
    Mov     Ax,3522h    ;Get interrupt vector
    Int     21h
    Mov     Off_22h,Bx
    ;Store offset and segment values
    Mov     Seg_22h,Es

```



```

;-----
; Set the new interrupt handler (int 22h)
;-----
    Push    Ds          ;Save DS
    Mov     Ax,2522h    ;Set interrupt vector
    Push    Cs
    Pop     Ds
    Mov     Dx,Offset New_i22
    Mov     Ax,3508h    ;The new interrupt handler will be New_i22
    Int     21h         ;Call Dos
    Pop     Ds          ;Get DS
    PRINTCH Msg6        ;Print screen a message
;-----
; Save the address of the DosOK flag for future usage
; If Dos is running something else the DosOK flag
; will show us that Dos is busy. To be able to use
; Dos we need to wait until it becomes available
; We can control Dos status by checking the value
; at this address location
;-----
    Mov     Ah,34h      ;Function 34h return the pointer to the
    Int     21h         ;DosOK flag
    Mov     Seg_Dos_F,Es
    PRINTCH Msg5        ;Save the segment and the offset values
    Mov     Off_Dos_F,Bx
    PRINTCH Msg7        ;Print screen a message
;-----
; Set the address of Int 23h (Ctrl-C or Ctrl-Break)
;-----
    Mov     Ax,2523h    ;Set The Ctrl-Break address
    Push    Ds          ;Save DS
    Push    Cs
    Pop     Ds
    Mov     Dx,Offset New_i23

```



```

; H08; The new Int 23h handler will be New_i23
Int 21h ;Call dos of the program and
Pop Ds ;Get DS
PRINTCH Msg10 ;Print screen a message
;-----
; Get the original int 08h and change vector
; 08h ==> User timer tick
;-----
Mov Ax,3508h ;Get interrupt vector
Int 21h ;Dos service call
Mov Off_08h,Bx ;Save Offset of interrupt
Mov Seg_08h,Es ;Save Segment of interrupt
;-----
; Set the system timer interrupt 08h
;-----
Push Ds
Mov Ax,2508h ;Set interrupt vector
Push Cs
Pop Ds
Mov Dx,Offset Cs:H08_Int
;The procedure which will handle Int 08h
Int 21h ;Dos service call
Pop Ds
PRINTCH Msg5 ;Print screen a message
Mov Old_Ss,Ss ;Save the SS and SP to the memory
Mov Old_Sp,Sp ;When we will run the task they
;will be destroyed
CREATE_RESULT_FILE
;Macro which will create result file
Do_Infinite:
debug 'a',0
Jmp Do_Infinite
;Wait for int 08h call.
Install Endp
;-----

```



```

; H08_Int is specially written to handle the original
; Int 08h . It controls the flow of the program and
; dedicates the cpu to the available tasks
;-----

```

```

H08_Int Proc Far
    Sti     Task_Status ;To enable the interrupt flag
    Push    Ds          ;Save Ds and Ax to the stack
    Push    Ax          ;of the program
    Mov     Ax,Data
    Mov     Ds,Ax        ;Load the Ds to the result file
    Pushf    Task_Flags ;Save flags
    Call    Old_int08    ;Call the old interrupt routine and
    ;execute it .(For system timer)
    ;At the return it restores the falgs back
    Cmp     Tick_C,0     ;Compare the Tick_C
    Je      In_Dos_Ctrl  ;If Tick_C is zero then control DosOK
    Dec     Tick_C
    Pop     Ax           ;Get the pushed values back
    Pop     Ds
    Iret     Task_Id     ;And turn back to the task which
    ;was running

In_Dos_Ctrl:
    CTRL_DOS          ;Call the CTRL_DOS amcro to
    ;Control DosOK flag
    RESTORE_08H       ;Call RESTORE_08H which will
    ;restore the old Int 08h
    SAVE_TASK_REGISTERS
    ;Save registers (Macro)
    GET_STOP_TIME     ;Get stop time of the task for result file
    ;If not change the task

Init_Tick:
    Mov     Tick_C,Quantum
    ;Reset the timer tick count
    Cmp     Task_Id,0
    Je      Init_Task

```



```

        Jmp      Ctrl_Status

Init_Task:
        Mov     Al,Task_Index
        Cbw
        Mov     Di,Ax
        Cmp     Task_Status[Di],4h
        Jne     Init_First
        WRITE_RECORD ;Write record to the result file
        Jmp     Thats_The_End

Init_First:
        Mov     Al,Task_Index
        Mov     Task[1],Al
        ;Assign the available task at the
        ;queue to task1
        Cbw
        Mov     Di,Ax
        Mov     Task_Status[Di],3h
        ;Activate task
        Mov     Task_Id,1 ;Task number 1 will be executed first
        Inc     Task_Index
        Call    Far Ptr Allocate_Memory4Task
        Jmp     Load_Task

Ctrl_Status:
        Mov     Al,Task_Id
        ;Find the status value of the task
        Cbw
        Mov     Di,Ax
        Mov     Al,Task[Di]
        Cbw
        Mov     Si,Ax
        Cmp     Task_Status[Si],0
        ;control if task is terminated or not
        je     Mark_Free

```



```

        Jmp      Chg_Status
;-----;Segment value of the memory to be
; Make the allocated memory free
;-----;Call Dos
Mark_Free:      Chg_Task      ;We need to set the main program PSP
Chg_Status:      ;When we turn from a task

        Mov     Ax,Data
        Mov     Ds,Ax
        Mov     Ah,50h      ;Set PSP segment (Dos reserved interrupt)
        Mov     Bx,Psp      ;Segment of the new PSP
Chg_Task:      Int     21h      ;Call Dos
GET_STOP_TIME   ;Call macro to record the stop time of tas
        Mov     Al,Task_Id
        Cbw
        Mov     Di,Ax
        Mov     Al,Task[Di]
        Cbw
        Mov     Si,Ax
        Mov     Task_Status[Si],0h
        ;Mark task as terminated.

GET_STATUS
        Mov     Task[Di],0fh ;Mark task as free
        Dec     Alloc_Task   ;Decrement allocated task number
        WRITE_RECORD          ;Write record to the result file
        Mov     Ah,49h      ;Deallocate the memory
        Cmp     Task_Id,1h
        Je      Free_Task1_Memory
        Jmp     Free_Task2_Memory
Free_Task1_Memory:
        Mov     Es,Task_1.Segm
Next_Alt:      ;Segment value of the memory to be
                ;deallocated
        Int     21h      ;Call Dos
        Jmp     Chg_Task
Free_Task2_Memory:

```



```

Mov     Es,Task_2.Segm ;Segment value of the memory to be
        Assign_No;
        Switch_No;deallocated
Assign_ Int    21h      ;Call Dos
        Jmp     Chg_Task_Id

Chg_Status:
        Mov     Task_Status[Si],1
        Mov     Al,Task_Id;Task is waiting for another quantum
        GET_STATUS [Di];Record the status of the task
        WRITE_RECORD ;Write record to the result file

Chg_Task:
        Mov     Task_Status[Si],1
        Cmp     User_exit,1
        Jne     continue_chg_task
        Jmp     thats_the_end
        Continue_chg_task:
        Mov     Al,Task_Id
        Sub     Si,Ax      ;Switch to the other task
        Cbw
        Mov     Ax,Si
        Mov     Task_Id,Al
        Mov     Al,Task[Si]
        Cbw
        Mov     Si,Ax
        Cmp     Si,0fh     ;Control if the other task is free
        Jnc     ;or not
        jne     Cont_Task
        Jmp     Next_Arg

Cont_Task:
        Mov     Task_Status[Si],3 ;task is active
        Jmp     Res_Reg_Ctrl_Dos

Next_Arg:
        Mov     Al,Task_Index
        Cbw
        Mov     Di,Ax
        Cmp     Task_Status[Di],2h

```



```

                                ;Is there any other task available
je      Assign_Next
jmp     Switch_Back
Assign_Next:
mov     al,Task_Id
cbw
mov     di,ax
mov     al,Task_Index
mov     task[di],al ;Select next task from the queue
mov     di,ax
mov     task_status[di],3 ;Activate task
inc     task_index
call    far ptr Allocate_Memory4Task
Load_Task:
GET_FILE_NAME      ;Record the file name of the program
GET_TASK_NUM       ;Record the task number
GET_START_TIME     ;Record the start time of the task
;-----
; Control file name to find out if the
; program is .Com or .Exe
;-----
mov     si,File_Pointer
dec     si
Find_File_Type:    ;Check the file type to determine
inc     si          ;whether the file is an .Exe file
cmp     file_names[si], '.'
                ;or not
jne     Find_File_Type
cmp     file_names[si+1], 'e'
jne     Upcase
jmp     Exe_Program
Upcase:
cmp     file_names[si+1], 'E'
                ;Control for upcase letters.
jne     Com_Program

```



```

        Jmp      Exe_Program
Com_program:
        ;-----
        ; This is a .Com program. It starts at
        ; hex 100. So load directly the
        ; program to the address org 100h
        ;-----
        Mov      Cs:T_Ip,0h ;The Ip must be 0
        Mov      Ax,Cs:T_Cs
        Mov      Ss,Ax      ;Cs is also Ss in a .Com program
        Mov      Sp,0ffffh ;The Sp start from the bottom of a
                           ;.Com program

        Cmp      Task_Id,1
        Je       Com_Task1_Segm
Com_Task2_Segm:
        Sub      Task_2.Segm,10h
                           ;Task_?.Segm gives the segment address
                           ;of the load module. To set the PSP
                           ;we need to sub 100h from the segment
                           ;value (but in terms of paragraph
                           ;100h = 10h paragraph)

        Jmp      Go_To_Execute
Com_Task1_Segm:
        Sub      Task_1.Segm,10h
Go_To_Execute:
        Sub      Ax,10h      ;Ds is also Cs in a .Com program
                           ;So sub 10h paragraph from that value

        Mov      Ds,Ax      ;Find the value of Ds
        Jmp      Start_Task
Exe_program:
        ;-----
        ; The program is an .Exe . We had to open and read the file
        ; header to find out the necessary information to load the
        ; task correctly to the proper location
        ;-----

```



```

Mov      Ax,Word Ptr File_Buff[14h]
                                ;Get the value of the IP from the
Mov      Cs:T_Ip,Ax ;header
Mov      Ax,Word Ptr File_Buff[16h]
                                ;Displacement in paragraph of code
                                ;segment
Add      Cs:T_Cs,Ax ;Add this value to the start address
                                ;to find the start address of code S.
Mov      Sp,Word Ptr File_Buff[10h]
                                ;Offset to be in SP when the module
                                ;given control(points the end of stack)
Mov      Ax,Word Ptr File_Buff[0Eh]
                                ;Displacement in paragraph of stack
                                ;segment within the load module
Cmp      Task_Id,1h
Je       Task1_Stack_Location
Jump     Task2_Stack_Location

Task1_Stack_Location:
Add      Ax,Task_1.Segm
                                ;find the location of stack segment
Mov      Ss,Ax
                                ;To be able to call the program we
                                ;need to specify the address of the PSP
                                ;Where the program will find INT 20h to
                                ;return to the program which calls it.
                                ;So we need to modify ES and DS before
                                ;Calling the program.
Sub      Task_1.Segm,10h
                                ;10h paragraf=100h
Mov      Bx,Task_1.Segm
                                ;Set the Psp of the Child process
Mov      Ah,50h ;Dos reserved func. (Bx holds the
                                ;new PSP value)
Int      21h ;Call Dos
Mov      Es,Task_1.Segm

```



```

;First ES must be changed.Otherwise
;Ds will be changed and it will be
;impossible to find the Segm value
Mov     Ds,Task_1.Segm
;move same value to Ds
Jmp     Start_Task
Task2_Stack_Location:
Add     Ax,Task_2.Segm
;find the location of stack segment
Mov     Ss,Ax ;The same instructions as Task1_Stack_loc.
Sub     Task_2.Segm,10h
Mov     Bx,Task_2.Segm
;Set the child process PSP
Mov     Ah,50h ;Dos reserved func.
Int     21h
Mov     Es,Task_2.Segm
;Before starting the task Es and DS must
;point to the begining of the program
;So Es,Ds points to PSP
;We need to change Es befor Ds
Mov     Ds,Task_2.Segm
Jmp     Start_Task
Switch_Back:
Mov     Si,3h
Mov     Al,Task_Id
Cbw
Sub     Si,Ax
Mov     Ax,Si
Mov     Task_Id,Al ;Change the task id
Mov     Al,Task[Si]
Cbw
Mov     Si,Ax
Cmp     Si,0fh ;Control if the other task is free
;or not
Je      Ctrl_Alloc_Task

```



```

        Jmp      Assign_Previous
Ctrl_Alloc_Task:
        Cmp      Alloc_Task,0
        ja       Assign_Previous
        Jmp      Thats_The_End
Assign_Previous:
        Mov      Task_Status[Si],3 ;Activate task back
Res_Reg_Ctrl_Dos:
        SET_PSP_SEGMENT ;Call macro to set the child process PSP
        GET_FILE_NAME ;Record the file name of the task
        GET_TASK_NUM ;Record the task number
        GET_START_TIME ;Record the startting time of the task
        SET_08h ;Set the interrupt handler specially
                ;written to take place of the original
                ;int 08h (system timer) interrupt
        RESTORE_SAVED_REGISTERS
                ;put the registers already saved back
        Jmp      Cs:Task_To_Run
                ;go to the pointed location of the program
                ;and continue to run the task
Start_Task:
        SET_08h ;set the interrupt handler
        Jmp      Cs:Task_To_Run
                ;jump to the begining of the task and
                ;start execute the code
;-----
;That shows the end of all tasks
;restore back the changed interrupts
;vectors and terminate the main program
;-----
Thats_The_End:
        Mov      Ax,Data
        Mov      Ds,Ax
        Mov      Ah,50h ;Set the PSP of the main program
        Mov      Bx,Psp ;The new PSP value

```



```

Int      21h          ;Call Dos
;-----
; Close result file before leaving
;-----
Mov      Ah,3eh       ;Close file
Mov      Bx,Result_Handle
                        ;Use the handle to close

Int      21h          ;Call Dos
PRINTCH Ack2          ;Print screen a message
;-----
; Set the original int 08h and change vector
; 08h ==> Terminate program
;-----
Push     Ds
Mov      Dx,Off_08h
                        ;Restore back the old interrupt 08h

Mov      Ds,Seg_08h
Mov      Ax,2508h     ;Set interrupt vector
Int      21H          ;Call Dos
Pop      Ds
PRINTCH Ack1          ;Print screen a message
Pop      Ax           ;Get back the two words pushed at the
Pop      Ax           ;begining of the program
PRINTCH Ack4          ;Print screen a message
Jmp      Cikis        ;Jump to the PSP of the program to be able
                        ;to terminate the task switcher program

```

H08_Int Endp

```

;-----
; This procedure allocates necessary amount of
; memory by reading the header information
; to calculate the amount of memory to the task
; which will be allocated
;-----

```

Allocate_Memory4Task Proc Far

```

Mov      Al,Task_Id

```



```

Cbw
Mov     Si,Ax
Mov     Al,Task[Si]
Cbw
Mul     Len      ;Multiply with the length of the file name
Mov     File_Pointer,Ax
Mov     Ah,20h    ;Find the pointer
;-----
; Control file name to find out if the
; program is .Com or .Exe
;-----
Mov     Si,File_Pointer
Dec     Si
Find_Type:
;-----
; Look to the file type and control
Inc     Si        ;if it is an .Exe or .Com
Cmp     File_Names[Si], '.'
Jne     Find_Type
Cmp     File_Names[Si+1], 'e'
Jne     Com_Prog
Jump    Exe_Prog
Cmp     File_Names[Si+1], 'E'
Jne     Com_Prog
Jump    Exe_Prog
Com_Prog:
;-----
; This a .Com program and takes
; 64k of maximum memory
;-----
Mov     Cs:T_Ip,0h ;The offset will be 0
Mov     Bx,0fffh   ;64k in paragraph
Mov     Ah,48h
Int     21h
Jnc     File_Allocation
Jump    Error5
Exe_Prog:

```



```

;-----
; The program is an .Exe . We need to read the file
; header to obtain some info about the file to find out
; proper cs and ip offsets
;-----

```

Open_File:

```

Mov     Ah,3Dh      ;File open function
Push    Si
Mov     Si,File_Pointer
Lea     Dx,File_Names[Si]
                        ;Pointer to ASCIIZ string
Pop     Si
Mov     Al,0h       ;Just read
Int     21h
Jnc     Read_File
Jump    Error1      ;If carry there is an error

```

Read_File:

```

Mov     File_Handle,Ax
                        ;Store file handle
Mov     Ah,3Fh      ;Read from file
Mov     Bx,File_Handle
Mov     Dx,Offset File_Buff
                        ;To the file buffer
Mov     Cx,28       ;Read file header (40 bytes)
Int     21h
Jnc     Close_File
Jump    Error3

```

Close_File:

```

Mov     Ah,3Eh      ;Close file
Mov     Bx,File_Handle
Int     21h
Jnc     Find_File_Name
Jump    Error4

```

Find_File_Name:

```

Mov     Bx,Alloc_Size

```



```

;Bx contains the size in paragraf
Mov     Ah,48h
Int     21h      ;Ax is a pointer to allocated memory
Jnc     File_Allocation
Jmp     Error5

File_Allocation:
    Cmp     Task_Id,1h
    Je      Task1_Parm_Segm
    Jmp     Task2_Parm_Segm

Task1_Parm_Segm:
    Mov     Task_1.Segm,Ax
                    ;Start address of the allocated block
    Mov     Dx,Task_1.Segm
                    ;Create PSP startting that address
    Jmp     Create_PSP

Task2_Parm_Segm:
    Mov     Task_2.Segm,Ax
                    ;Start address of the allocated block
    Mov     Dx,Task_2.Segm
                    ;Create PSP startting that address

Create_PSP:
    mov     ah,55h      ;Create psp of a child process reserved do
    int     21h      ;Call Dos
    Cmp     Task_Id,1h
    Je      Task1_Parm_Reloc
    Jmp     Task2_Parm_Reloc

Task1_Parm_Reloc:
    Add     Task_1.Segm,10h
                    ;10h paragraph=100h bytes PSP length
    Mov     Ax,Task_1.Segm
    Mov     Task_1.Reloc,Ax
    Jmp     Find_Task_Cs

Task2_Parm_Reloc:
    Add     Task_2.Segm,10h
                    ;10h paragraph=100h bytes PSP length

```




```

Error2:  Mov     Ax,Task_2.Segm
        Mov     Task_2.Reloc,Ax
Find_Task_Cs:
        Mov     Cs:T_Cs,Ax;Move the start address of the block . later
;on we will calculate the real address
;of the code segment using file header
Error3:  Push    Si
        Mov     Si,File_Pointer
Error4:  LeatCh  Dx,File_Names[Si]
Exit:    ;Name of the program which will be loaded
        Mov     Ax,4B03h;Load program
        Push    Ds
        Pop     Es
        Pop     Si
        Cmp     Task_Id,2
        Jne     Segm_Off_Task1
        Jmp     Segm_Off_Task2
Segm_Off_Task1:
        Mov     Bx,Offset Task_1.Segm
        Jmp     Load_Prg
Segm_Off_Task2:
        Mov     Bx,Offset Task_2.Segm
Load_Prg:
        Int     21h
        Jnc     End_Prog
        Jmp     Error6
End_Prog:
        Inc     Alloc_Task;Increment the value of allocated tasks
        Ret
Task_To_Run Label Dword ;That's the Cs:Ip value for the task
;We will make an intersegment jump
T_IP      Dw     ?
T_CS      Dw     ?
Error1:  PRINTCH AllocErr1 ;print error messages for allocation
        Jmp     Exit

```



```

Error2: PRINTCH AllocErr2
        Jmp      Exit
Error3: PRINTCH AllocErr3
        Jmp      Exit
Error4: PRINTCH AllocErr4
        Jmp      Exit
Error5: PRINTCH AllocErr5
        Jmp      Exit
Error6: PRINTCH AllocErr6
Exit:
        Jmp      Thats_The_End
        Ret                      ;Return to the calling procedure
Allocate_Memory4Task Endp
;-----
; This procedure is used to terminate task.
; It restores the stack segment and pointer
; values back. (they were destroyed)
;-----
New_i22 Proc      Far          ;The new interrupt handler for Int 22h
        Cli                      ;Clear interrupt
        Mov      Ax,data
        Mov      Ds,Ax
        RESTORE_08h          ;Call macro to restore the original Int 08h
        Mov      Ss,Old_Ss ;Restore the SS and SP of the main
        Mov      Sp,Old_Sp ;allocation program
        Sti                      ;Store Interrupt
        Jmp      Mark_Free ;task is terminated so go to deallocate
                          ;the memory
New_i22 Endp
;-----
; This procedure handles the Int 23h Ctrl-Break
;-----
New_i23 Proc      Far
        Mov      Ax,Data
        Mov      Ds,Ax

```



```

5.2.2 Restore_08h
mov user_exit,1
; Mark_free
New_i23 Endp
Endoffset: ;Is a label to specify the length of the CS
Code Ends
End Install

; Clear screen
CLS MACRO
PUSHB
MOV CX,0000H ;Upper left row/col
MOV CX,104FH ;Lower right row/col
MOV BH,07H ;Normal Attribute (Black & White)
MOV AX,0600H ;AH=06 (scroll), AL=00 (left)
INC J0B
POPB
ENDM

; Put Cursor Position
; Entry: 1000H - Row (0-100), 1100H - Col (0-100)
CURSOR MOVSI LOCATE
PUSHB
MOV SI,1000H ;Row (0-100)
MOV DI,1100H ;Col (0-100)
MOVSI LOCATE
PUSHB
MOV SI,1000H ;Row (0-100)
MOV DI,1100H ;Col (0-100)

```


5.5.2 Macro.Lib

```
;-----  
; Name of the program : MACRO.LIB  
; Created      AX      : 07/12/90  
; Written By   BX      : Ahmet Tevfik İNAN  
;  
; (c) All rights of this program is reserved by A.Tevfik İNAN  
; (c) Kopya hakkı ve diğer bütün hakları A.Tevfik İNAN'a aittir.  
;-----  
;-----  
; Clear Screen  
;-----  
CLS      MACRO      DX  
          PUSHAX  
          MOV     CX,0000H      ;Upper left row/col  
          MOV     DX,184FH      ;Lower right row/col  
          MOV     BH,07H        ;Normal attribute (Black & White)  
          MOV     AX,0600H      ;AH 06 (scroll), AL 00 (full screen)  
          INT     10h  
          POPAX  
          ENDM  
;-----  
; Set Cursor Position  
; Entry : Locate => Row (High) Col (Low)  
;-----  
CURSOR   MACRO      LOCATE  
          PUSHAX  
          MOV     AH,15         ;Get current video mode  
          INT     10h  
          MOV     DX,LOCATE     ;Cursor position  
          MOV     AH,2          ;Set cursor position  
          INT     10h  
          POPAX  
          ENDM  
;-----
```



```

; Push Registers
;-----
PUSHA    MACRO
    PUSH    AX
    PUSH    BX
    PUSH    CX
    PUSH    DX
ENDM

;-----
; Pop Registers
;-----
POPA     MACRO
    POP     DX
    POP     CX
    POP     BX
    POP     AX
ENDM

;-----
; Writing string
; Entry : Pointer to the text
;-----
PRINTCH  MACRO    TEXT
    PUSHA
    LEA     DX,TEXT
    MOV     AH,9      ;Write string
    INT     21H
    POPA
ENDM

;-----
; Save register to the structure
; Entry : Task
;-----
SAVE_REGISTERS MACRO TASK
    ASSUME  CS:CODE,DS:DATA,SS:STACK
    MOV     TASK.BBP,BP

```



```

PUSH    BP
PUSH    AX
MOV     BP,SP
MOV     AX,[BP+4]
MOV     TASK.AAX,AX
MOV     AX,[BP+6]
MOV     TASK.DDS,AX
MOV     AX,[BP+12]
MOV     TASK.FLAGS,AX
MOV     AX,[BP+8]
MOV     TASK.IIP,AX
MOV     TASK.BBX,BX
MOV     TASK.CCX,CX
MOV     TASK.DDX,DX
MOV     TASK.SSI,SI
MOV     TASK.DDI,DI
MOV     AX,[BP+10]
MOV     TASK.CCS,AX
MOV     TASK.EES,ES
MOV     TASK.SSS,SS
ADD     BP,14
MOV     TASK.SSP,BP
POP     AX
POP     BP
ENDM

```

```

;-----
; Restore already saved registers
; Entry : Task
;-----

```

RESTORE_REGISTERS MACRO TASK

```

ASSUME  CS:CODE,DS:DATA,SS:STACK
PUSH    TASK.FLAGS
POPF
MOV     AX,TASK.CCS
MOV     CS:T_CS,AX

```



```

MOV     AX,TASK.IIP
MOV     CS:T_IP,AX
MOV     SP,TASK.SSP
MOV     AX,TASK.AAX
MOV     BX,TASK.BBX
MOV     CX,TASK.CCX
MOV     DX,TASK.DDX
MOV     BP,TASK.BBP
MOV     SI,TASK.SSI
MOV     DI,TASK.DDI
MOV     ES,TASK.EES
MOV     SS,TASK.SSS
MOV     DS,TASK.DDS
ENDM

```

```

;-----
; Control which tasks registers to save
; Uses Save_registers macro
;-----

```

SAVE_TASK_REGISTERS MACRO

```

LOCAL   EOMSAVE,SAVE_TASK1,SAVE_TASK2
ASSUME  CS:CODE,DS:DATA,SS:STACK
CMP     TASK_ID,02H
JE      SAVE_TASK2

```

SAVE_TASK1:

```

SAVE_REGISTERS T1
JMP     EOMSAVE

```

SAVE_TASK2:

```

SAVE_REGISTERS T2

```

EOMSAVE:

```

ENDM

```

```

;-----
; Control which tasks registers to restore
; Uses Restore_registers macro
;-----

```

RESTORE_SAVED_REGISTERS MACRO


```

LOCAL    RESTORE_TASK2, RERTORE_TASK1, EOMRESTORE
ASSUME   CS:CODE, DS:DATA, SS:STACK
CMP      TASK_Id, 02H
JE       RESTORE_TASK2
;-----
RESTORE_TASK1:
    RESTORE_REGISTERS T1
    JMP    EOMRESTORE
RESTORE_TASK2:
    RESTORE_REGISTERS T2
EOMRESTORE:
    ENDM
;-----
; Controls if DosOK flag
;-----
CTRL_DOS Macro
    LOCAL CONT
    PUSH    SI
    PUSH    ES
    LES     SI, DOS_OK
    CMP     BYTE PTR ES:[SI], 0H
    JE      CONT
    POP     ES    ;get back the pushed registers .
    POP     SI
    POP     AX    ;get back ax and ds. They were pushed
    POP     DS    ;at the Int 08h handler routine
    IRET
CONT:
    POP     ES    ;get back the pushed registers
    POP     SI
    ENDM
;-----
; Sets the Int 08h handler
;-----
SET_08H MACRO
    ASSUME CS:CODE, DS:DATA, SS:STACK
    PUSH    ES

```



```

PUSH    AX
XOR     AX,AX
MOV     ES,AX
MOV     AX,OFFSET CS:H08_INT
CLI     AX,OFF_71H
MOV     ES:[20H],AX
MOV     AX,CS
MOV     ES:[22H],AX
STI     ES:[04H],AX
POP     AX
POP     ES
ENDM

```

```

;-----
; Restores back the original Int 08h
;-----

```

RESTORE_08H MACRO

```

ASSUME  CS:CODE,DS:DATA,SS:STACK
PUSH    ES
PUSH    AX
XOR     AX,AX
MOV     ES,AX
MOV     AX,OFF_08H
CLI
MOV     ES:[20H],AX
MOV     AX,SEG_08H
MOV     ES:[22H],AX
STI
POP     AX
POP     ES
ENDM

```

```

;-----
; Restores back the original Int 21h
;-----

```

RESTORE_21H MACRO

```

ASSUME  CS:CODE,DS:DATA,SS:STACK

```



```

PUSH     ES
PUSH     AX
XOR      AX,AX
MOV      ES,AX
MOV      AX,OFF_21H
CLI
MOV      ES:[84H],AX
MOV      AX,SEG_21H
MOV      ES:[86H],AX
STI
POP      AX
POP      ES
ENDM

```

```

;-----
; Sets the Int 21h handler
;-----

```

```

SET_21H MACRO

```

```

    ASSUME  CS:CODE,DS:DATA,SS:STACK
    PUSH   ES
    PUSH   AX
    XOR    AX,AX
    MOV    ES,AX
    MOV    AX,OFFSET CS:NEW_I21
    CLI
    MOV    ES:[84H],AX
    MOV    AX,CS
    MOV    ES:[86H],AX
    STI
    POP    AX
    POP    ES
ENDM

```


5.5.3 Extra.Lib

```
;-----  
; Name of the program : EXTRA.LIB  
; Created              : 07/12/90  
; Written By          : Ahmet Tevfik İNAN  
;  
; (c) All rights of this program is reserved by A.Tevfik İNAN  
; (c) Kopya hakkı ve diğer bütün hakları A.Tevfik İNAN'a aittir.  
;-----  
;  
; Get start time of the task  
;  
GET_START_TIME MACRO  
    ASSUME CS:CODE,DS:DATA,SS:STACK  
    MOV     AH,2CH  
    INT     21H  
    LEA     SI,REC.F_START  
    MOV     [SI],CH  
    MOV     [SI+1],CL  
    MOV     [SI+2],DH  
    MOV     [SI+3],DL  
    ENDM  
;  
; Get stop time of the task  
;  
GET_STOP_TIME MACRO  
    ASSUME CS:CODE,DS:DATA,SS:STACK  
    MOV     AH,2CH  
    INT     21H  
    LEA     SI,REC.F_STOP  
    MOV     [SI],CH  
    MOV     [SI+1],CL  
    MOV     [SI+2],DH  
    MOV     [SI+3],DL  
    ENDM
```



```

;-----
; Get number of the task
;-----

GET_TASK_NUM      MACRO
    ASSUME  CS:CODE,DS:DATA,SS:STACK
    MOV     AL,TASK_ID
    MOV     REC.F_TASK,AL
    ENDM

;-----
; Get the name of the file
;-----

GET_FILE_NAME     MACRO
    LOCAL   REP,EO_PROCESS,A20
    ASSUME  CS:CODE,DS:DATA,SS:STACK
    LEA     DI,REC.F_NAME
    MOV     CX,30
    MOV     AL,0
A20:    MOV     [DI],AL
    INC     DI
    LOOP    A20
    MOV     AL,TASK_ID
    CBW
    MOV     SI,AX
    MOV     AL,TASK[SI]
    CBW
    MUL     LEN
    MOV     SI,AX
    LEA     DI,REC.F_NAME
REP:    MOV     AL,BYTE PTR FILE_NAMES[SI]
    CMP     AL,00H
    JE      EO_PROCESS
    MOV     [DI],AL
    INC     SI
    INC     DI

```



```

        JMP      REP
EO_PROCESS:
        ENDM
;-----
; Write the record to the result file
;-----

WRITE_RECORD    MACRO
        LOCAL    CONT3,EO_WRITE_RESULT
        ASSUME   CS:CODE,DS:DATA,SS:STACK
        MOV      BX,RESULT_HANDLE
        MOV      AH,40H          ;Write to the file
        MOV      DX,OFFSET REC
                                ;Offset address of the record which will
                                ;Be written to the file
        MOV      CX,RESULT_REC_LEN
                                ;Numbers of byte to write to the file

        INT      21H
        JNC      CONT3
        PRINTCH  RESULT_ERR3
        JMP      EO_WRITE_RESULT
CONT3:
        INC      RESULT_FILE_POINTER
                                ;The pointer which shows thw start position
                                ;of the next record of the result file
EO_WRITE_RESULT:
        ENDM
CREATE_RESULT_FILE    MACRO
        ASSUME   CS:CODE,DS:DATA,SS:STACK
        MOV      AH,3CH          ;Create file
        LEA      DX,RESULT_FILE  ;File to create
        MOV      CX,0H           ;File attribute
        INT      21H
        MOV      RESULT_HANDLE,AX ;Save the file handle
        ENDM
;-----

```



```

; Get the status of the task
;-----
GET_STATUS MACRO
    ASSUME CS:CODE,DS:DATA,SS:STACK
    MOV     AL,TASK_ID
    CBW
    MOV     SI,AX
    MOV     AL,TASK[SI]
    CBW
    MOV     SI,AX
    MOV     AL,TASK_STATUS[SI]
    MOV     REC.F_STAT,AL
ENDM
;-----
; Set the PSP(Program Segment Prefix) of the task
;-----
SET_PSP_SEGMENT MACRO
    LOCAL TASK_PSP1,TASK_PSP2,END_PSP
    ASSUME CS:CODE,DS:DATA,SS:STACK
    CMP     TASK_ID,1
    JE      TASK_PSP1
    JMP     TASK_PSP2
TASK_PSP1:
    MOV     BX,TASK_1.SEGM
    JMP     END_PSP
TASK_PSP2:
    MOV     BX,TASK_2.SEGM
END_PSP:
    MOV     AH,50H
    INT     21H
ENDM

```


5.5.4 Debug.Lib

```
;-----  
; Name of the program : DEBUG.LIB  
; Created              : 07/12/90  
; Written By          : Ahmet Tefvik İNAN  
;  
; (c) All rights of this program is reserved by A.Tefvik İNAN  
; (c) Kopya hakkı ve diğer bütün hakları A.Tefvik İNAN'a aittir.  
;-----
```

```
PRINT_MSG    MACRO    STR  
               ASSUME  CS:CODE,DS:DATA,SS:STACK  
               PRINTCH STR  
               MOV     AH,0      ;wait key  
               INT     16H  
               ENDM
```

```
WRITE_REG    MACRO    REG,OFFSET  
               PUSH    DS  
               PUSH    BX  
               PUSH    AX  
               MOV     AX,0B800H  
               MOV     DS,AX  
               POP     AX  
               PUSH    AX  
               PUSH    REG  
               MOV     BX,REG  
               PREP_HEX BH  
               MOV     BYTE PTR DS:[OFFSET],AH  
               MOV     BYTE PTR DS:[OFFSET+2],AL  
               POP     REG  
               MOV     BX,REG  
               PREP_HEX BL  
               MOV     BYTE PTR DS:[OFFSET+4],AH  
               MOV     BYTE PTR DS:[OFFSET+6],AL  
               MOV     AH,0 ; wait key  
               INT     16H
```



```

5.5.1.2 POP      AX
(*) POP      BX
POP      DS
ENDM

ADJUST_TO_HEX MACRO REG
    LOCAL     DONE
    CMP      REG,10
    JB       DONE
    ADD      REG,7
DONE:
    ADD      REG,30H
ENDM

PREP_HEX MACRO REG
    PUSH     CX
    MOV      AH,REG
    MOV      CL,4
    SHR      AH,CL
    ADJUST_TO_HEX AH
    MOV      AL,REG
    AND      AL,0FH
    ADJUST_TO_HEX AL
    POP      CX
ENDM

DEBUG MACRO CHAR,OFFSET
    PUSH     DS
    PUSH     AX
    MOV      AX,0B800H
    MOV      DS,AX
    MOV      BYTE PTR DS:[OFFSET],CHAR
    POP      AX
    POP      DS
ENDM

```


5.5.5 Create.Pas

```
(*
Name of the program : Create.Pas
Created           : 7/12/1990
Written by       : A. Tefvik İNAN
(c) All rights of this program is reserved by A. Tefvik İNAN
(c) Kopya hakkı ve diğer bütün hakları A. Tefvik İNAN'a aittir.
*)
Program create(input,output);
Uses
    dos;
Var
    sf:file of byte;
    df,f:text;
    ratio,weighted_time,tot_weighted_time:real;
    i,j,k,line_count,page_count,tot_quantum,end_mark,mark:integer;
    Task_names:array [1..10] of string[30];
    Task_hh,task_mm,task_ss,task_dd,entry: array [1..10] of integer;
    Task_pointer,temp_pointer: integer;
    file_name,temp_file: string [30];
    file_num,file_status:byte;
    temp:byte;
    starth,startm,starts,startd:integer;
    stoph,stopm,stops,stopd:integer;
    stoph1,stopm1,stops1,stopd1:integer;
    wh_start,wm_start,ws_start,wd_start: array [1..10] of integer;
    wh_stop,wm_stop,ws_stop,wd_stop: array [1..10] of integer;
    durh,durm,durs,durd:integer;
    toth,totm,tots,totd:integer;
    match:boolean;
    year,month,day,dayofweek:word;
    hour,minute,second,hund:word;
    time1,time2,time3,time4,turn_around_time:longint;
    t_hh,t_mm,t_ss,t_dd,time5:integer;
Begin
```



```

(***** initialization of the arrays *****)
for i:=1 to 10 do begin task_hh[i]:=0;
                        task_mm[i]:=0;
                        task_ss[i]:=0;
                        task_dd[i]:=0;
                        entry[i]:=0;
                        wh_start[i]:=0;wm_start[i]:=0;
                        ws_start[i]:=0;wd_start[i]:=0;
                        wh_stop[i]:=0;wm_stop[i]:=0;
                        ws_stop[i]:=0;wd_stop[i]:=0;
                        end;
tot_quantum:=0;
toth:=0;totm:=0;tots:=0;totd:=0;
durh:=0;durm:=0;durs:=0;durd:=0;
starth:=0;startm:=0;starts:=0;startd:=0;
stopph:=0;stopm:=0;stops:=0;stopd:=0;
line_count:=56;page_count:=1;
(***** open input and output files *****)
assign(f,'err');
rewrite(f);
Assign(sf,'result.ref');
Assign(df,'report.dat');
Task_pointer:=0;
Rewrite(df);
Reset(sf);

(** read source file and make the necessary computations **)
writeln(df);writeln(df);writeln(df);
writeln(df);writeln(df);writeln(df);
getdate(year,month,day,dayofweek); gettime(hour,minute,second,hund
writeln(df,'REPORT FILE OF TASK SWITCHER');
Write(df,'Created at ');
case dayofweek of
0: write(df,'Sunday ');
1: write(df,'Monday ');
2: write(df,'Tuesday ');

```



```

3: write(df,'Wednesday ');
4: write(df,'Thursday ');
5: write(df,'Friday ');
6: write(df,'Saturday ');
end;
write(df,day,'/',month,'/',year,' ');
if (hour div 10) = 0 then write(df,'0');
write(df,hour,':');
if (minute div 10) = 0 then write(df,'0');
write(df,minute,':');
if (second div 10) = 0 then write(df,'0');
writeln(df,second);
writeln(df);writeln(df);
writeln(df);writeln(df);
writeln(df,'All Rights of this program is reserved');
writeln(df,'No one can use this utility without the');
writeln(df,'permission of A. Tevfik INAN ');
writeln(df,'Istanbul 22/11/1990');
While not eof(sf) do
Begin
    line_count:=line_count+1;
    if (line_count > 55) then
    begin
        line_count:=8;
        writeln(df,chr(12));
        writeln(df,'
                Page: ',page_count);
        Writeln(df,'          T A S K   S W I T C H E R
        R E P O R T   F I L E ');
        Writeln(df,'-----
        -----');
        writeln(df,'Task Switcher (c) All Rights Reserved
        by A. Tevfik INAN');
        Writeln(df,'');
        Writeln(df,' # Task Name                Start Time

```



```

Stop Time    Duration    Status ');
Writeln(df,'--- -----
----- ');
page_count:=page_count+1;
end;
writeln(df);
Read(sf,file_num); Write(df,file_num:2);Write(df,' ');
for i:=1 to 30 do Begin
    Read(sf,temp);
    if ( temp= 0) then temp:=32;
    file_name[i]:=chr(temp);
End;
file_name[0]:=chr(30);
write(df,file_name);
(***** get the start time and make the necessary changes *****)
Read(sf,temp);starth:=temp;
if ((temp div 10) = 0) Then write(df,'0');
write(df,starth,':');
Read(sf,temp);startm:=temp;
if ((temp div 10) = 0) Then write(df,'0');
write(df,startm,':');
Read(sf,temp);starts:=temp;
if ((temp div 10) = 0) Then write(df,'0');
write(df,starts,':');
Read(sf,temp);startd:=temp;
if ((temp div 10) = 0) Then write(df,'0');
write(df,startd,':');
(***** get the stop time and make the necessary changes *****)
Read(sf,temp);stoph:=temp;
if ((temp div 10) = 0) Then write(df,'0');
write(df,stoph,':');
Read(sf,temp);stopm:=temp;
if ((temp div 10) = 0) Then write(df,'0');
write(df,stopm,':');
Read(sf,temp);stops:=temp;

```



```

    if ((temp div 10) = 0) Then write(df,'0');
    write(df,stops,'. ');
    Read(sf,temp);stopd:=temp;
    if ((temp div 10) = 0) Then write(df,'0');
    write(df,stopd,' ');

    (***** calculate the duration of the task at that quantum *****)
    stoph1:=stoph;stopm1:=stopm;stops1:=stops;stopd1:=stopd;
    durd:=stopd-startd;
    if(durd < 0 ) Then Begin stopd:=stopd+100;
                        stops:=stops-1;
                        durd:=stopd-startd;
                        End;
    durs:=stops-starts;
    if(durs < 0 ) Then Begin stops:=stops+60;
                        stopm:=stopm-1;
                        durs:=stops-starts;
                        End;
    durm:=stopm-startm;
    if(durm < 0 ) Then Begin stopm:=stopm+60;
                        stoph:=stoph-1;
                        durm:=stopm-startm;
                        End;
    durh:=stoph-starth;
    if ((durh div 10) = 0) Then write(df,'0');
    Write(df,durh,': ');
    if ((durm div 10) = 0) Then write(df,'0');
    Write(df,durm,': ');
    if ((durs div 10) = 0) Then write(df,'0');
    Write(df,durs,'. ');
    if ((durd div 10) = 0) Then write(df,'0');
    write(df,durd,' ');

    (***** get the status at that specific task *****)
    Read(sf,temp);file_status:=temp;
    end_mark:=0;
    case (file_status) of

```



```

0: begin
    write(df, 'Ended');
    end_mark:=1
end;

1: write(df, 'Waiting');
2: write(df, 'Not Started');
3: write(df, 'Active');
4: write(df, 'No Task');
End;

(***** look to the task names already used if it is present
continue with adding the time otherwise first add the
name of the file then add the time values *****)
i:=1;
match:=false;
while (i<=task_pointer) do
begin
    j:=i;
    temp_file:=task_names[i];
    if (file_name = temp_file) then begin mark:=i;
                                     i:=task_pointer+1;
                                     match:=true;
    end;
    i:=i+1;
end;
if ( not match ) then begin
    task_pointer:= task_pointer+1;
    j:=task_pointer;
    mark:=j;
    task_names[j]:=file_name;
    wh_start[j]:=starth;
    wm_start[j]:=startm;
    ws_start[j]:=startis;
    wd_start[j]:=startd;
end;

(***** calculate the total time spend for all tasks *****)

```



```

totd:=totd+durd; if (totd > 99 ) then begin totd:=totd-100;
                                                    tots:=tots+1;
                                                    end;
tots:=tots+durs; if (tots > 59 ) then begin totd:=totd-60;
                                                    totm:=totm+1;
                                                    end;
totm:=totm+durm; if (totm > 99 ) then begin totm:=totm-60;
                                                    toth:=toth+1;
                                                    end;
toth:=toth+durh;
if ( end_mark = 1 ) then begin
                                                    wh_stop[mark]:=stoph1;
                                                    wm_stop[mark]:=stopm1;
                                                    ws_stop[mark]:=stops1;
                                                    wd_stop[mark]:=stopd1;
                                                    end;
(***** try to calculate the time spend for each task *****)
entry[j]:=entry[j]+1;
task_dd[j]:=task_dd[j]+durd;
task_ss[j]:=task_ss[j]+durs;
task_mm[j]:=task_mm[j]+durm;
task_hh[j]:=task_hh[j]+durh;
if (task_dd[j] > 99 ) then begin task_dd[j]:=task_dd[j]-100;
                                task_ss[j]:=task_ss[j]+1;
                                end;
if (task_ss[j] > 59 ) then begin task_ss[j]:=task_ss[j]-60;
                                task_mm[j]:=task_mm[j]+1;
                                end;
if (task_mm[j] > 59 ) then begin task_mm[j]:=task_mm[j]-60;
                                task_hh[j]:=task_hh[j]+1;
                                end;
End; (* end of while eof(df) *)
(***** write the total spen time to the report file *****)
writeln(df);
writeln(df,'=====');

```



```

if ((task_hh[i] div 10 ) = 0) then write(df,'0');
write(df,task_hh[i],':');
if ((task_mm[i] div 10 ) = 0) then write(df,'0');
write(df,task_mm[i],':');
if ((task_ss[i] div 10 ) = 0) then write(df,'0');
write(df,task_ss[i],'.');
if ((task_dd[i] div 10 ) = 0) then write(df,'0');
write(df,task_dd[i],' |');
time1:=task_hh[i]*60*60*100+task_mm[i]*60*100+
      task_ss[i]*100+task_dd[i];
time2:=toth*60*60*100+totm*60*100+tots*100+totd;
ratio:=time1/time2;
if (( 100 * ratio / 10 ) < 1) then
  write(df,' % ',(100*ratio):2:2,' | ');
else
  write(df,' % ',(100*ratio):2:2,' | ');
tot_quantum:=tot_quantum+entry[i];
if (entry[i] div 10 ) = 0 then
  writeln(df,' ',entry[i],' |')
else writeln(df,' ',entry[i],' |');
writeln(df,'+-----+-----+');
      +-----+-----+');

End;
line_count:=55;
tot_weighted_time:=0;
for i:=1 to task_pointer do
begin
  time1:=0;time2:=0;
  line_count:=line_count+1;
  if (line_count > 55 ) then
  begin
    line_count:=4;
    writeln(df,chr(12));
    writeln(df,'
          Page: ',page_count);

```



```

writeln(df,'Task Switcher (c) All Rights Reserved by A.
Tevfik INAN');
writeln(df);
writeln(df,'Special time statistics are as follows ...')
page_count:=page_count+1;
writeln(df,'+-----+-----+-----+-----+-----+
+-----+-----+');
writeln(df,'| Task name | Turn around
| Elapsed | Weighted TT |');
writeln(df,'+-----+-----+-----+-----+-----+
+-----+-----+');
end;
write(df,'| ',task_names[i],'| ');
time1:=wh_start[i]*60*60*100+wm_start[i]*60*100
+ws_start[i]*100+wd_start[i];
time2:=wh_stop[i]*60*60*100+wm_stop[i]*60*100
+ws_stop[i]*100+wd_stop[i];
turn_around_time:=time2-time1;
time4:=turn_around_time;
t_hh:=time4 ;
t_hh:=t_hh div (60*60*100);
if t_hh = 0 then write(df,'00:')
else begin
if ((t_hh div 10 ) = 0) then write(df,'0');
write(df,t_hh,':');
end;
time4:= time4 - t_hh*60*60*100;
t_mm:= time4;
t_mm:=t_mm div (60*100);
if t_hh = 0 then write(df,'00:')
else begin
if ((t_hh div 10 ) = 0) then write(df,'0');
write(df,t_hh,':');
end;
time4:= time4 - t_mm*60*100;

```

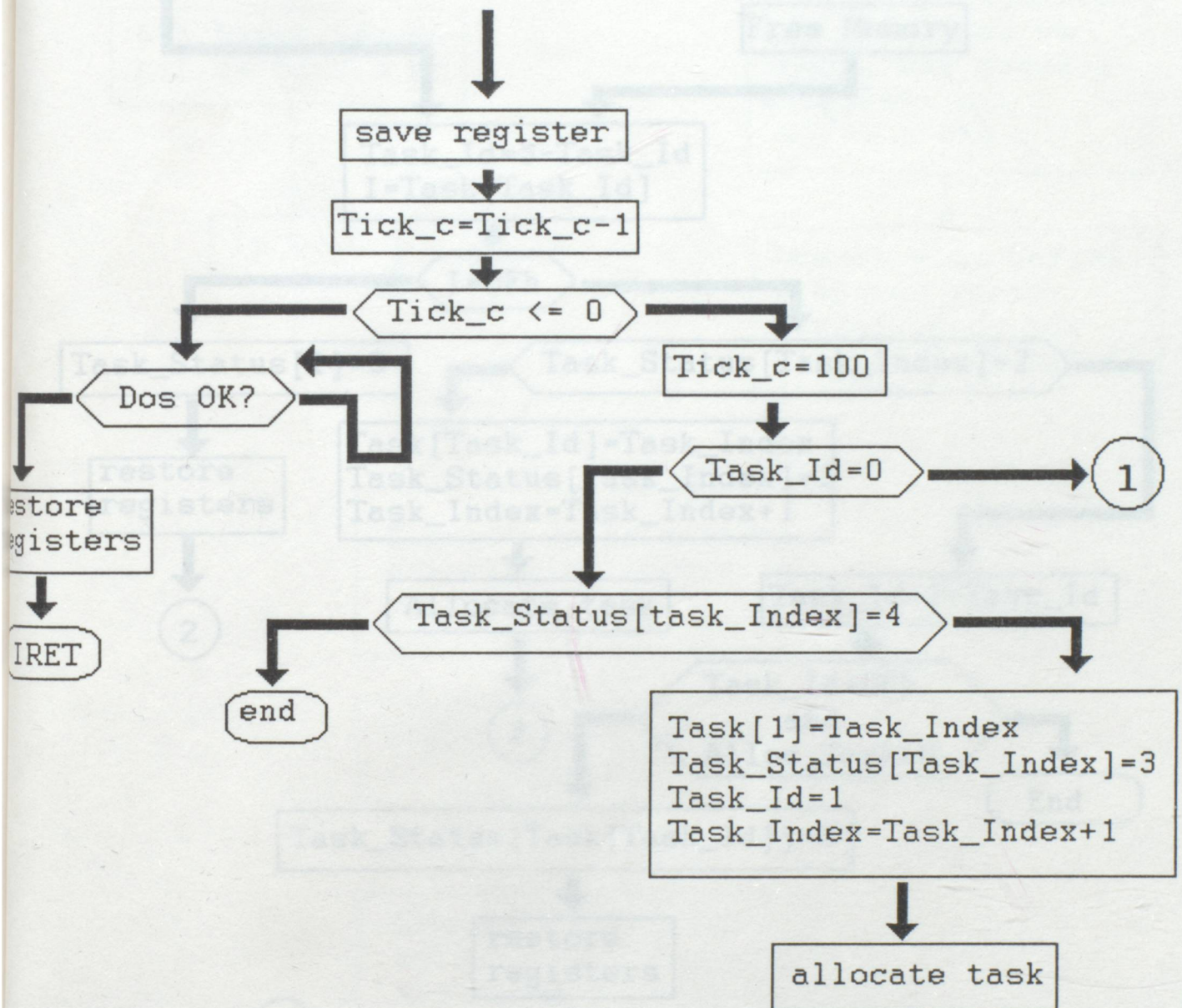


```

t_ss:= time4 ;
t_ss:=t_ss div 100;
if t_ss = 0 then write(df,'00:')
else begin
    if ((t_ss div 10) = 0) then write(df,'0');
    write(df,t_ss,'. ');
end;
time4:= time4 - t_ss*100;
t_dd:=time4;
if t_dd = 0 then write(df,'00:')
else begin
    if ((t_dd div 10) = 0) then write(df,'0');
    write(df,t_dd,' | ');
end;
t_dd:=t_dd - task_dd[i];
if t_dd < 0 then begin t_dd:=t_dd + 100;
t_ss:=t_ss-1;
end;
t_ss:=t_ss - task_ss[i];
if t_ss < 0 then begin t_ss:=t_ss + 60;
t_mm:=t_mm-1;
end;
t_mm:=t_mm - task_mm[i];
if t_mm < 0 then begin t_mm:=t_mm + 60;
t_hh:=t_hh-1;
end;
t_hh:=t_hh - task_hh[i];
if (t_hh div 10) = 0 then write(df,'0');
write(df,t_hh,': ');
if (t_mm div 10) = 0 then write(df,'0');
write(df,t_mm,': ');
if (t_ss div 10) = 0 then write(df,'0');
write(df,t_ss,'. ');
if (t_dd div 10) = 0 then write(df,'0');
write(df,t_dd,' | ');

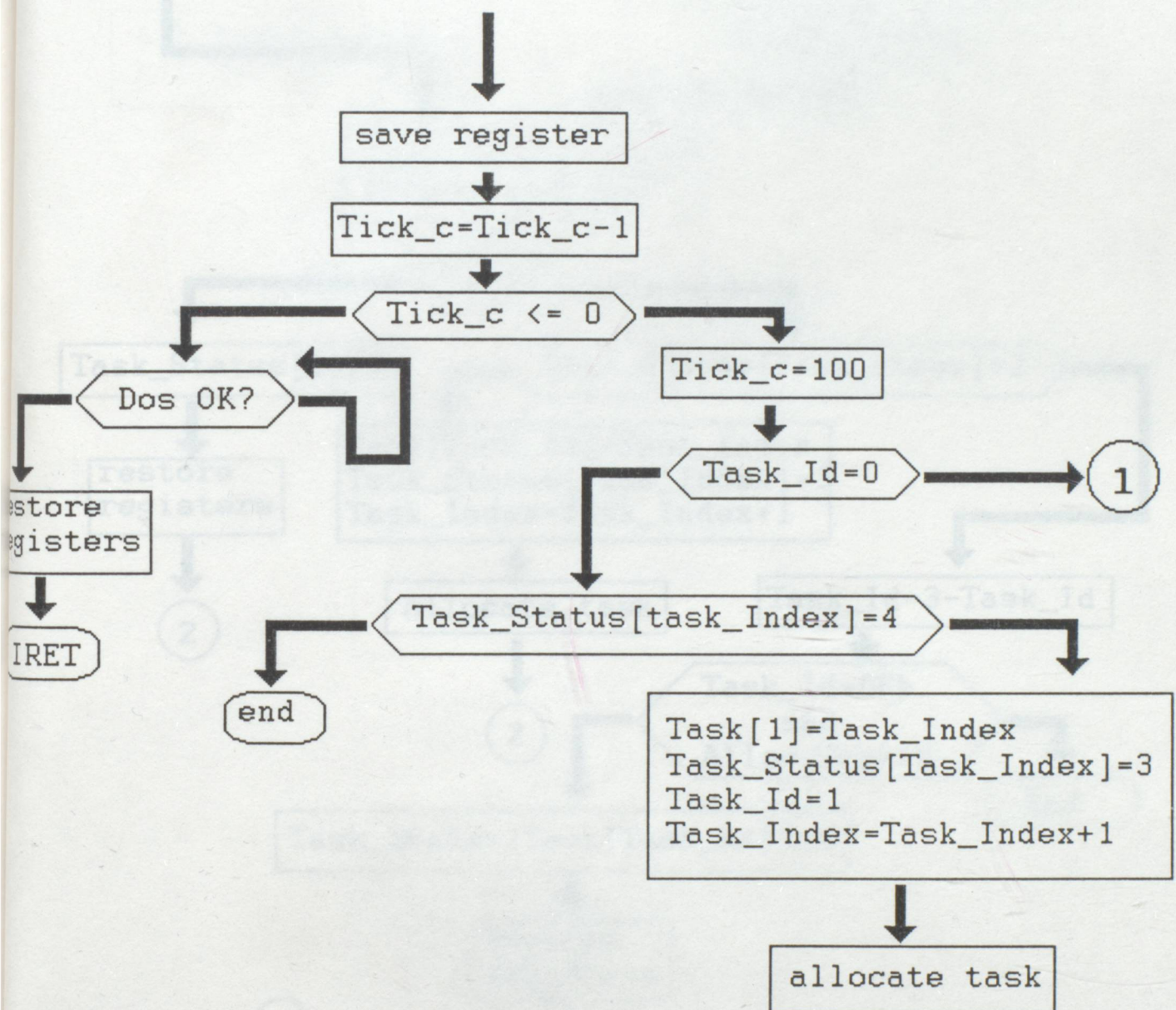
```


5.5.6 Programin Akisi

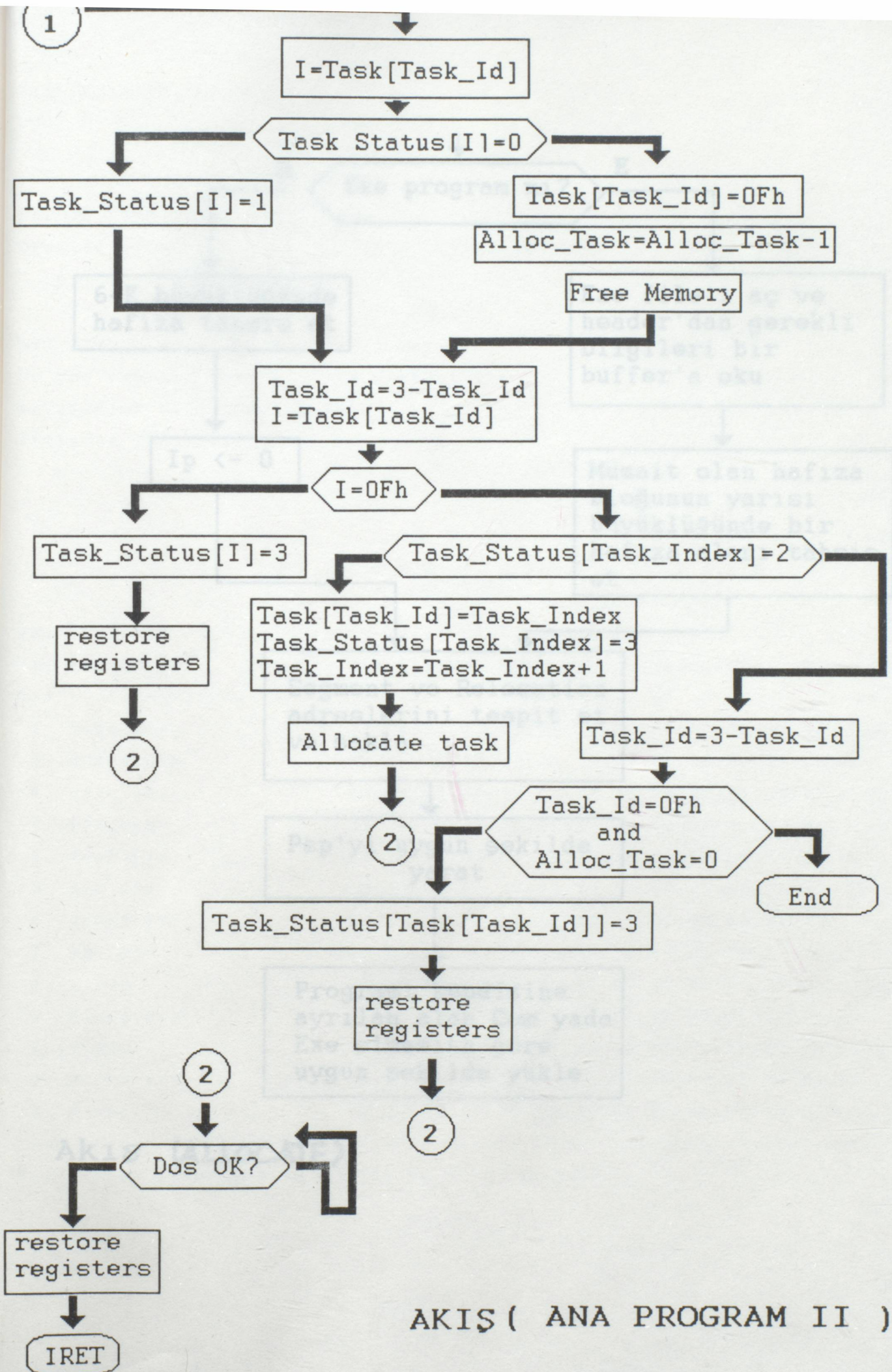


AKIS (ANA PROGRAM I)

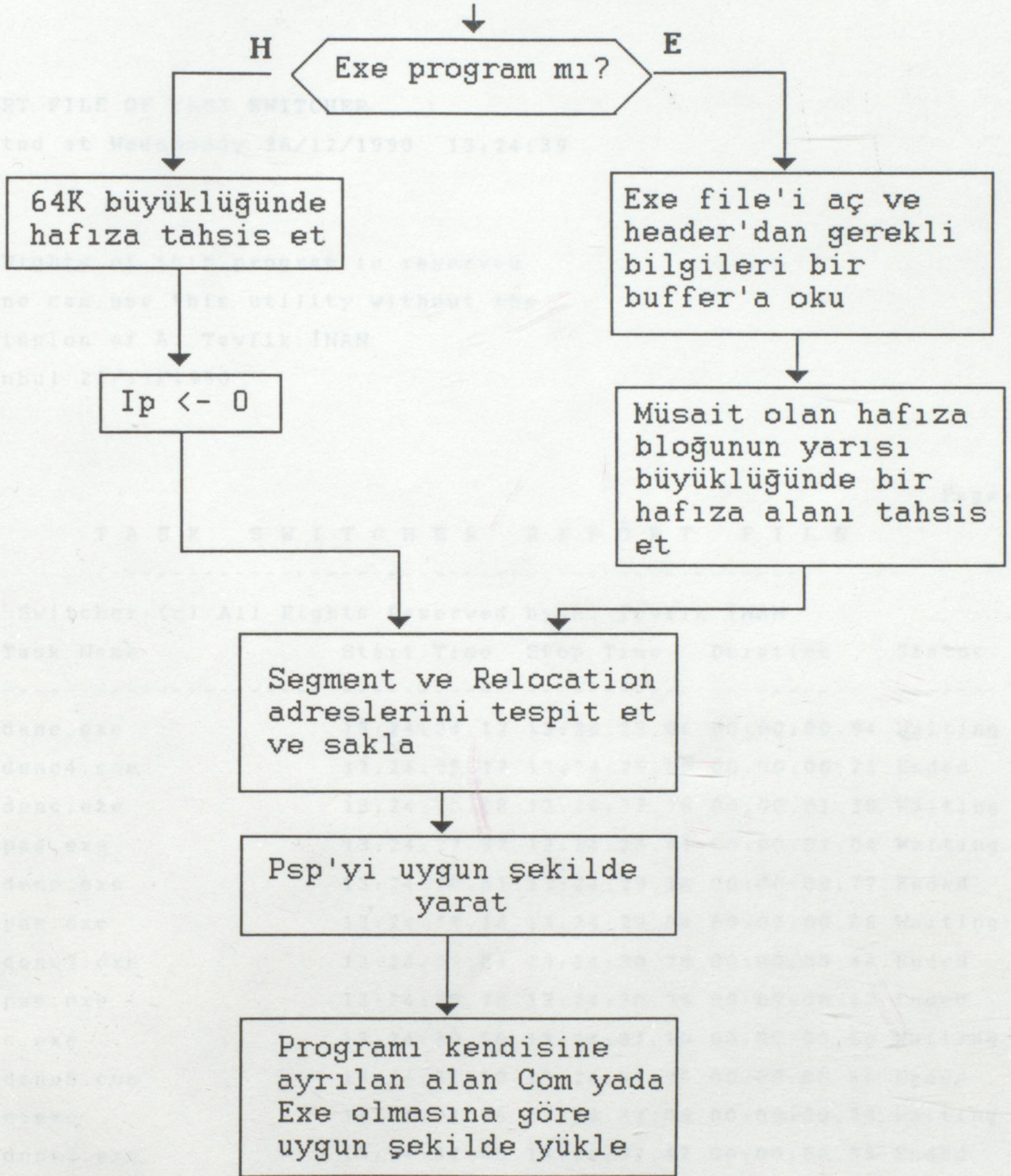
5.5.6 Programin Akisi



AKIS (ANA PROGRAM I)



AKIS (ANA PROGRAM II)



Akış (ALLOCATE)

5.6 Üretilen raporlar

Task Switcher (c) All Rights Reserved by A. Tevfik İNAN
Individual Task statistics are as follows

REPORT FILE OF TASK SWITCHER

Created at Wednesday 26/12/1990 13:24:39

All Rights of this program is reserved

No one can use this utility without the

permission of A. Tevfik İNAN

İstanbul 22/11/1990

Page: 1

T A S K S W I T C H E R R E P O R T F I L E

Task Switcher (c) All Rights Reserved by A. Tevfik İNAN

#	Task Name	Start Time	Stop Time	Duration	Status
1	dene.exe	13:24:24.12	13:24:25.06	00:00:00.94	Waiting
2	dene4.com	13:24:25.17	13:24:25.88	00:00:00.71	Ended
1	dene.exe	13:24:25.88	13:24:27.26	00:00:01.38	Waiting
2	pas.exe	13:24:27.37	13:24:28.41	00:00:01.04	Waiting
1	dene.exe	13:24:28.41	13:24:29.18	00:00:00.77	Ended
2	pas.exe	13:24:29.18	13:24:29.84	00:00:00.66	Waiting
1	dene2.exe	13:24:29.84	13:24:30.28	00:00:00.44	Ended
2	pas.exe	13:24:30.28	13:24:30.39	00:00:00.11	Ended
1	c.exe	13:24:30.50	13:24:31.10	00:00:00.60	Waiting
2	dene5.com	13:24:31.10	13:24:31.76	00:00:00.66	Ended
1	c.exe	13:24:31.76	13:24:32.09	00:00:00.33	Waiting
2	dene3.exe	13:24:32.09	13:24:32.47	00:00:00.38	Ended
1	c.exe	13:24:32.47	13:24:32.91	00:00:00.44	Ended

Total time spend for all tasks is 00:00:08.46

Task Switcher (c) All Rights Reserved by A. Tevfik İNAN

Individual Task statistics are as follows ...

Task name	Run Time	% time spend	Quan
dene.exe	00:00:03.09	% 36.52	3
dene4.com	00:00:00.71	% 8.39	1
pas.exe	00:00:01.81	% 21.39	3
dene2.exe	00:00:00.44	% 5.20	1
c.exe	00:00:01.37	% 16.19	3
dene5.com	00:00:00.66	% 7.80	1
dene3.exe	00:00:00.38	% 4.49	1

Average Weighted Turn Around Time for this group of tasks is 1.990

Task Switcher (c) All Rights Reserved by A. Tevfik İNAN

Special time statistics are as follows ...

Task name	Turn around	Elapsed	Weighted TT
dene.exe	00:00:05.06	00:00:01.97	1.64
dene4.com	00:00:00:71	00:00:00.00	1.00
pas.exe	00:00:03.02	00:00:01.21	1.67
dene2.exe	00:00:00:44	00:00:00.00	1.00
c.exe	00:00:02.41	00:00:01.04	1.76
dene5.com	00:00:00:66	00:00:00.00	1.00
dene3.exe	00:00:00:38	00:00:00.00	1.00

Average Weighted Turn Around Time for this group of tasks is 1.30

KAYNAKLAR

Kitaplar:

- 1-) Advanced Microprocessor Architectures
Luigi Ciminiera & Adriano Valenzana
ISBN 0-201-145502
- 2-) An Introduction to Operating Systems
Harvey M. Deiter
Addison Wesley
- 3-) CTask 2.00
Thomas Wagner
- 4-) Assembly Language Tools & Techniques for IBM Microcomputers
Julio Sanchez
Prentice Hall ISBN 0-13-050493-9
- 5-) Dos Programmers Reference
Terry R. Dettman
QUE Corp. ISBN 0-88022-327-8
- 6-) Extending Dos
Ray Duncan
Addison Wesley ISBN 0-13-449067-3
- 7-) IBM PC Assembler Language and Programming
Peter Abel
Prentice Hall ISBN 0-13-449067-3
- 8-) Ms-Dos Technical Reference
- 9-) Operating Systems
Stuart E. Madnick & John J. Donovan
McGraw Hill ISBN 0-07-Y85467-X
- 10-) Operating Systems Design and Implementation
Andrew S. Tanenbaum
Prentice Hall ISBN 0-13-637331-3
- 11-) Operating System Design Vol.1 The XINU Approach
Douglas Comer & Timothy V. Fossum
Prentice Hall ISBN 0-13-638313-0
- 12-) The Logical Design of Operating Systems

Lubomir Bic & Alan C. Shaw

Prentice Hall ISBN 0-13-540196-8

Makaleler:

- 1-) Fair Share
Jonathan Robie (Byte July 1988)
- 2-) First Come First Served
Micheal B. Parker (Byte July 1988)
- 3-) Free At Last
Mark J. Minasi (Byte July 1990)
- 4-) It's Natural
Bud E. Smith (Byte July 1988)
- 5-) Marrying UNIX and the 80386
Carl Hensler & Ken Sarno (Byte April 1988)
- 6-) Mastering OS/2 Threads
Douglas A. Hamilton (Byte September 1990)
- 7-) Multitasking for the Masses
Rick Grehan (Byte February 1990)
- 8-) OS/2 Marries Deskview
Mark J. Minasi (Byte June 1990)
- 9-) OS/2 Multitasking Revisited
Mark j. Minasi (Byte December 1989)
- 10-) OS/2 Programmer's Reference
- 11-) OS/2 Virtual Memory Management
Vic Heller (Byte April 1988)
- 12-) Overview of Memory Management
Randall L. Hyde (Byte 1988)
- 13-) Protected Mode
L. Brett Glass (Byte December 1989)
- 14-) Weighting the Options
Brett Glass (Byte July 1988)

SONSÖZ

Hazırladığım bu tez çalışmamda çoklu işlemeyi (Multitasking) tanıtmaya ve yazdığım programla ortaya basitte olsa bir uygulama koymaya çalıştım. Çalışmamda kullandığım yöntemler hep makinenin yazılım gücünü ortaya çıkaracak şekilde oldu. Ancak günümüzde Intel'in ürettiği 80286 ve sonrasını teşkil eden 80386 ve 80486 işlemcileri donanım olarak çoklu işlemeye imkan sağlayan bir tasarımın ürünüdürler. Özellikle 80286 ile başlayan protected mode'un kullanılmasıyla çoklu işleme yapılırken donanımın desteğini de kullanmak mümkündür. 80386 işlemcisinin, V8086 modunda çalıştırılmasıyla tek işlemci donanım olarak birden fazla 8086 işlemcisi olarak kullanılabilmekte, 80386'nın sağladığı sayfalı hafıza yönetimi desteği ile mevcut hafızadan daha büyük programlar herhangi bir kontrol zorluğu ile karşılaşılmaksızın çalıştırılabilmektedir. Ancak tekrar belirtmek isterim ki bunlar işlemcinin bilinen fakat benim hakkında detaylı bilgi ve kaynak bulamadığım protected mode' da çalışması halinde mümkündür. İleri bu konu üzerinde çalışma yapmak isteyen arkadaşların bu imkanları da göz önünde bulundurarak çok daha etkin, makinenin yazılım etkinliği yanında donanım desteklerinden de faydalanacak yazılımlar üreteceklerine hiç şüphe yoktur.

Bu konu üzerinde yapılan çalışmalar ve kaynaklar incelendiğinde genellikle çoklu işleme yapan yazılımların C programla dilinde geliştirilmiş olduğu görülmektedir. C programlama dili assembly diline yakınlığı, herhangi bir üst düzey diline benzer yapı ve komutları ile bu konudaki çalışmalarda oldukça etkin bir kullanıma sahiptir. Dolayısıyla bu tip çalışmalarda C programlama dilinin kullanılması pek çok açıdan faydalı olmaktadır.



