Norwegian University of Science and Technology

Operating Systems

Example solutions for Theoretical Exercise 6

Michael Engel

6.1 Rate Monotonic Schedulability

The following tasks are given:

	τ_1	τ_2	τ_3
C_i	1	3	2
T_i	3	8	9

The priorities of the tasks are assigned statically before the actual execution of the task set. RMS assigns higher priority to tasks with smaller periods. Tasks are preempted by the higher priority tasks.

It is an optimal scheduling algorithm amongst fixed-priority algorithms; if a task set cannot be scheduled with RM, it cannot be scheduled by any fixed-priority algorithm.

6.1 Rate Monotonic Schedulability

The following tasks are given:

	τ_1	τ_2	τ_3
C_i	1	3	2
T_i	3	8	9

a. Test if the given task set is schedulable under RM, using the sufficient test

The sufficient schedulability test is given by:

$$U = \sum_{i=1}^{n} \frac{C_i}{D_i} \leq n(2^{1/n} - 1)$$

The term *U* is the processor utilization factor (the fraction of the processor time spent on executing task set). *n* is the number of tasks.

For our case: $1/3 + 3/8 + 2/9 = 0.93 \not\leq 3(2^{1/3} - 1) = 0.78$ fails!

The above condition *is not necessary*, so we don't know if the task set is schedulable with RM or not. Accordingly, we can try to figure this out using the test in subtask b.

6.1 Rate Monotonic Schedulability

The following tasks are given:

	τ_1	τ_2	τ_3
C_i	1	3	2
T_i	3	8	9

b. Test if the given task set is schedulable under RM, using the necessary test.

We have to guarantee that all the tasks can be scheduled, in any possible instance. In particular, if a task can be scheduled in its critical instances, then the schedulability guarantee condition holds (a critical instance of a task occurs whenever the task is released simultaneously with all higher priority tasks).

In the following, we perform the iterative schedulability algorithm on slide 6-54 (for the details of the algorithm, please have a look at page 99, Buttazzo's book).

The tasks are first ordered by their priorities: τ1, τ2 and τ3. (here the tasks are already ordered.)

6.1 Rate Monotonic Schedulability

The following tasks are given:

	τ_1	τ_2	τ_3
C_i	1	3	2
T_i	3	8	9

b. Test if the given task set is schedulable under RM, using the necessary test

In our case:

τ_3:

$R_3^0 = C_3 = 2 \quad I_3^0 = \lceil \frac{2}{3} \rceil 1 + \lceil \frac{2}{8} \rceil 3 = 1 + 3 = 4 \quad 4 + 2 \neq 2$

$R_3^1 = 4 + 2 = 6 \quad I_3^1 = \lceil \frac{6}{3} \rceil 1 + \lceil \frac{6}{8} \rceil 3 = 2 + 3 = 5 \quad 5 + 2 \neq 6$

$R_3^2 = 5 + 2 = 7 \quad I_3^2 = \lceil \frac{7}{3} \rceil 1 + \lceil \frac{7}{8} \rceil 3 = 3 + 3 = 6 \quad 6 + 2 \neq 7$

$R_3^3 = 6 + 2 = 8 \quad I_3^3 = \lceil \frac{8}{3} \rceil 1 + \lceil \frac{8}{8} \rceil 3 = 3 + 3 = 6 \quad 6 + 2 = 8 \ldots \text{OK}$

(since $R_3 = 8 \leq T_3 = 9$)

τ_2:

$R_2^0 = C_2 = 3 \quad I_2^0 = \lceil \frac{3}{3} \rceil 1 = 1 \quad 1 + 3 \neq 3$

$R_2^1 = 1 + 3 = 4 \quad I_2^1 = \lceil \frac{4}{3} \rceil 1 = 2 \quad 2 + 3 \neq 4$

$R_2^2 = 2 + 3 = 5 \quad I_2^2 = \lceil \frac{5}{3} \rceil 1 = 2 \quad 2 + 3 = 5 \ldots \text{OK}$ (since $R_2 = 5 \leq T_2 = 8$)

τ_1:

$R_1^0 = C_1 = 1 \quad I_1^0 = 0 \quad 0 + 1 = 1 \ldots \text{OK}$ (since $R_1 = 1 \leq T_1 = 3$)

The necessary and sufficient test succeeds. This means that the task set is schedulable with RM.

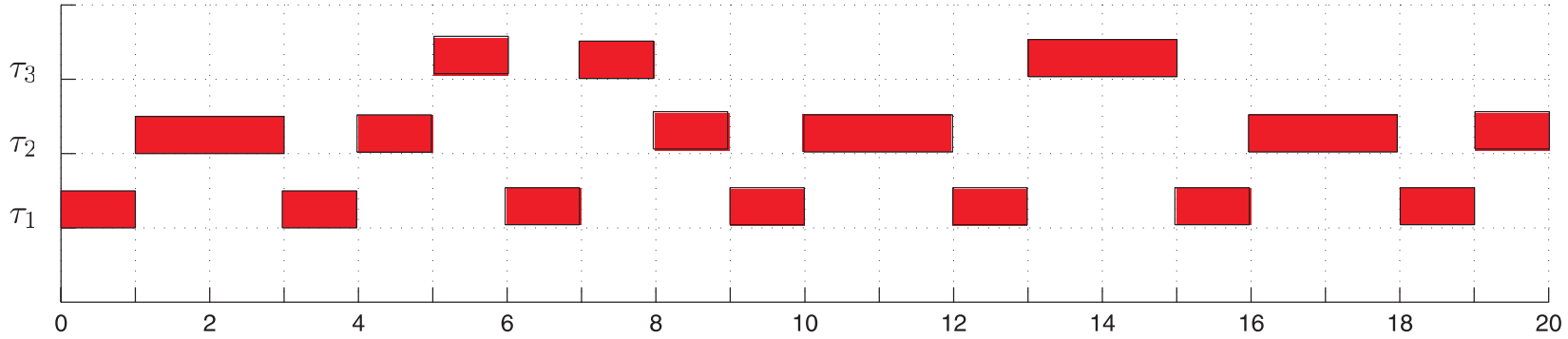
6.1 Rate Monotonic Schedulability

The following tasks are given:

	τ_1	τ_2	τ_3
C_i	1	3	2
T_i	3	8	9

c. Assume that the first job of each task arrives at time 0. Construct the schedule for the interval [0, 20] and illustrate it graphically. In case they exist, identify deadline misses.

There are no deadline misses (as shown in the previous subtask)



6.2 Periodic Scheduling

A processor is supposed to execute the following set of tasks described by their execution times *C*, relative deadlines *D* and periods *T*:

	τ_1	τ_2	τ_3
C_i	2	2	4
D_i	5	4	8
T_i	6	8	12

a. Execute the sufficient schedulability test for EDF and calculate the result. What statement regarding schedulability can be made based on your result?

Using $\sum_{i=1}^{n} \frac{C_i}{D_i} \leq 1$ we compute $2/5 + 2/4 + 4/8 = 1.4 \not\leq 1$ ***failed***

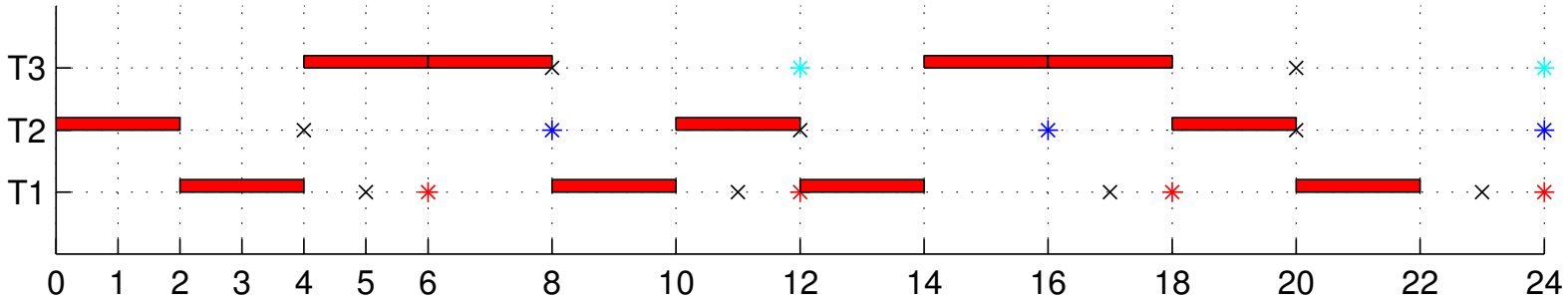
Since $D_i < T_i$ this test is sufficient, but not necessary. Hence, the task set ***might be schedulable*** (as we see below).

6.2 Periodic Scheduling

A processor is supposed to execute the following set of tasks described by their execution times *C*, relative deadlines *D* and periods *T*:

	τ_1	τ_2	τ_3
C_i	2	2	4
D_i	5	4	8
T_i	6	8	12

b. If there is a feasible schedule for the given task set, construct it graphically. Let the phase Φi = 0 ∀ i.



A ***necessary and sufficient test*** for EDF is to use the demand bound analysis, which is not covered in the course. Interested reader may look at the following reference S. Baruah, A. Mok, L. Rosier. Preemptive Scheduling Hard-Real-Time Sporadic Tasks on One Processor. Proceedings of the Real-Time Systems Symposium, 182-190, 1990.

6.3 SSTF Disk Scheduling

Explain why SSTF scheduling tends to favor middle cylinders over the innermost and outermost cylinders.

The center of the disk is the location having the smallest average distance to all other tracks.

Thus after servicing the first request, the algorithm would be more likely to be closer to the center track than to any other particular track, and hence would more often go there first.

Once at a particular track, SSTF tends to keep the head near this track; thus, this scheduling strategy would compound the initial tendency to go to the center.

6.4 FAT Scheduling

Requests are not usually uniformly distributed. For example, a cylinder containing the file system FAT or inodes can be expected to be accessed more frequently than a cylinder that only contains files. Suppose that you know that 50 percent of the requests are for a small, fixed number of cylinders.

a. Would any of the scheduling algorithms discussed in this chapter be particularly good for this case? Explain your answer.

SSTF would take greatest advantage of the situation. FCFS could cause unnecessary head movement if references to the “high-demand” cylinders were interspersed with references to cylinders far away.

6.4 FAT Scheduling

Requests are not usually uniformly distributed. For example, a cylinder containing the file system FAT or inodes can be expected to be accessed more frequently than a cylinder that only contains files. Suppose that you know that 50 percent of the requests are for a small, fixed number of cylinders.

b. Propose a disk scheduling algorithm that gives even better performance by taking advantage of this “hot spot” on the disk.

One suggestion: SSTF with some modification to prevent starvation.

Second suggestion: (A variation of SCAN) The head is allowed to make one “zigzag” over the high-volume tracks. An upper limit on the number of consecutive requests (on the same track) that can be serviced before moving on would prevent starvation.

6.4 FAT Scheduling

Requests are not usually uniformly distributed. For example, a cylinder containing the file system FAT or inodes can be expected to be accessed more frequently than a cylinder that only contains files. Suppose that you know that 50 percent of the requests are for a small, fixed number of cylinders.

c. Filesystems typically find data blocks via an indirection table, such as a FAT in DOS or inodes in UNIX. Describe one or more ways to take advantage of this indirection to improve the disk performance.

One idea: the indirection table is relatively small, so it could be cached in RAM and only written back when the system is shut down or rebooted.

However, the integrity of the system is in danger here, since at least a part of the file metadata is not synchronized to disk. In case of a sudden power failure or kernel panic, all information since the last synchronization could be irrecoverably lost.

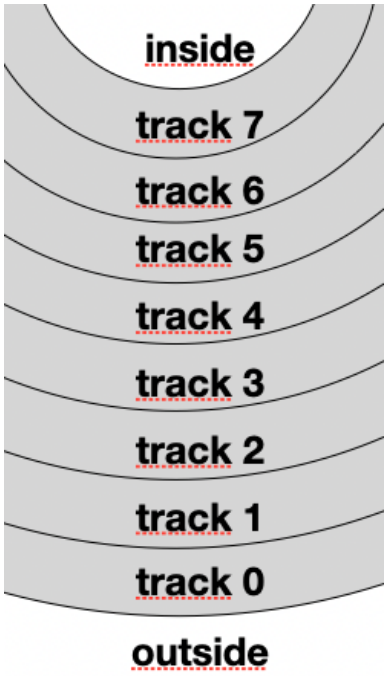
6.5 Disk scheduling

a. Assume a magnetic disk with 8 tracks. After each second read request (starting from *L1*), the I/O scheduler receives additional read requests which are grouped (requested) together (*L2* and finally *L3*).

Initially, the read/write head of the disk is at track 0.

Give the I/O scheduling order that would be performed according to the **SSTF (shortest seek time first) algorithm**:

L1 = {2,4,3,1}, *L2* = {5,6}, *L3* = {0,7}



Access order:

1. L1 received and ordered SSTF: 1, 2, 3, 4
2. Read track 1 and track 2
3. L2 received, added to remaining, SSTF: 3, 4, 5, 6
4. Read tracks 3 and 4
5. L3 received, added to remaining, SSTF: 5, 6, 7, 0

=> Order: {1, 2, 3, 4, 5, 6, 7, 0}

6.5 Disk scheduling

b. Assume a magnetic disk with 8 tracks. After each ***third*** read request (starting from *L1*), the I/O scheduler receives additional read requests which are grouped (requested) together (*L2* and finally *L3*).

Initially, the read/write head of the disk is at track 0.

Give the I/O scheduling order that would be performed according to the

elevator algorithm: ***that "4" was a typo, should have read "3"***

L1 = {1,4,7,2}, *L2* = {**4**,6,0}, *L3* = {5,2}

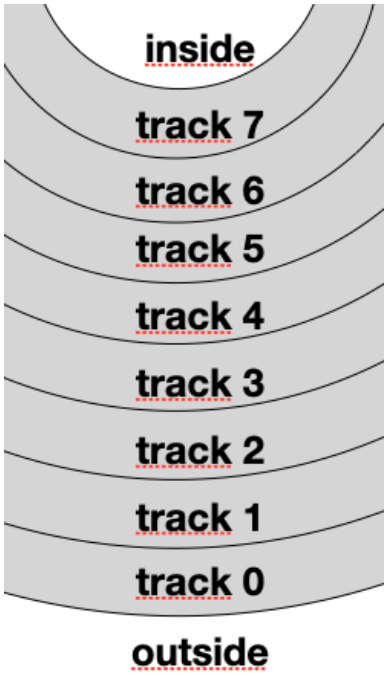
Access order:

1. L1 received and ordered elevator: 1, 2, 4, 7
2. Read track 1, track 2 and track 4
3. L2 received, added to remaining, elevator: 6, 7, 0
4. *Read tracks 4* ***and 4? ➛ ambiguous, 1 or 2 reads?***

➛ 4. Read tracks 4, 6 and 7!

5. L3 received, added to remaining, elevator: 5, 2, 0

=> Order: {1, 2, 4, 6, 7, 5, 2, 0}



6.5 Disk scheduling

b. Assume a magnetic disk with 8 tracks. After each ***third*** read request (starting from *L1*), the I/O scheduler receives additional read requests which are grouped (requested) together (*L2* and finally *L3*).

Initially, the read/write head of the disk is at track 0.

Give the I/O scheduling order that would be performed according to the

elevator algorithm:

L1 = {1,4,7,2}, *L2* = {***3***,6,0}, *L3* = {5,2}

with typo corrected…

Access order:

1. L1 received and ordered elevator: 1, 2, 4, 7

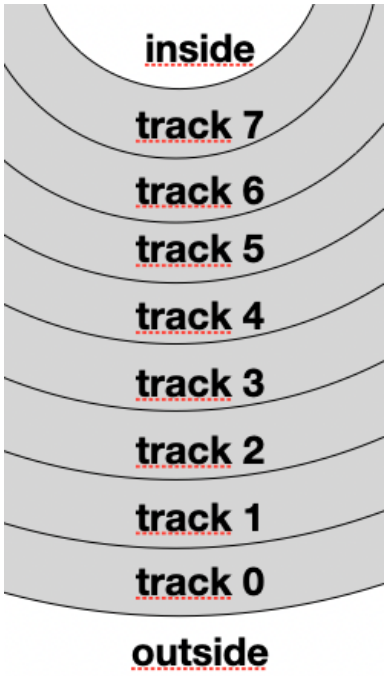
2. Read track 1 and track 2

3. L2 received, added to remaining, elevator: 3, 4, 6, 7, 0

4. Read tracks 3 and 4

5. L3 received, added to remaining, elevator: 5, 6, 7, 2, 0

=> Order: {1, 2, 3, 4, 5, 6, 7, 2, 0}



6.6 FAT file system

This question is related to a task you would need to perform if you were a computer security expert doing a forensic analysis of a disk.

Given a hexadecimal dump of blocks on the disk and a description of the block contents, you need to figure out the meaning of that data.

Bytes	Content
0–10	File name (8 bytes) with extension (3 bytes)
11	Attribute - a bitvector. Bit 0: read only. Bit 1: hidden. Bit 2: system file. Bit 3: volume label. Bit 4: subdirectory. Bit 5: archive. Bits 6-7: unused.
12–21	Reserved
22–23	Time (5/6/5 bits, for hour/minutes/doubleseconds)
24–25	Date (7/4/5 bits, for year-since-1980/month/day)
26–27	Starting cluster (0 for an empty file)
28-31	File size in bytes

6.6 FAT file system

Bytes	Content
0–10	File name (8 bytes) with extension (3 bytes)
11	Attribute - a bitvector. Bit 0: read only. Bit 1: hidden. Bit 2: system file. Bit 3: volume label. Bit 4: subdirectory. Bit 5: archive. Bits 6-7: unused.
12–21	Reserved
22–23	Time (5/6/5 bits, for hour/minutes/doubleseconds)
24–25	Date (7/4/5 bits, for year-since-1980/month/day)
26–27	Starting cluster (0 for an empty file)
28-31	File size in bytes

For each directory entry, find out:

a. The name of the entry

That was simple enough! Note that the "." in the name is not stored on disk, but the first 8 bytes are filled with 0x20 if that part is < 8 characters!

```
address data bytes ..................................  ASCII representation
                    file name       extension
0009728 49 4f 20 20 20 20 20 20 53 59 53 27 00 00 00 00  IO      .SYS
0009744 00 00 00 00 00 00 08 5d 62 1b 1d 00 16 9f 00 00
0009760 4d 53 44 4f 53 20 20 20 53 59 53 27 00 00 00 00  MSDOS   .SYS
0009776 00 00 00 00 00 00 08 5d 62 1b 6d 00 38 95 00 00
0009792 43 4f 4d 4d 41 4e 44 20 43 4f 4d 20 00 00 00 00  COMMAND .COM
0009808 00 00 00 00 00 00 07 5d 62 1b b8 00 39 dd 00 00
0009824 44 42 4c 53 50 41 43 45 42 49 4e 27 00 00 00 00  DBLSPACE.BIN
0009840 00 00 00 00 00 00 08 5d 62 1b 27 01 f6 fc 00 00
0009856 4d 53 44 4f 53 20 20 20 20 20 20 28 00 00 00 00  MSDOS
0009872 00 00 00 00 00 00 1a 88 99 1c 00 00 00 00 00 00
0009888 46 44 49 53 4b 20 20 20 45 58 45 20 00 00 00 00  FDISK   .EXE
0009904 00 00 00 00 00 00 36 59 62 1b 02 00 17 73 00 00
```

These dots are a lie :) ***Not stored on disk***

6.6 FAT file system

Bytes	Content
0–10	File name (8 bytes) with extension (3 bytes)
11	Attribute - a bitvector. Bit 0: read only. Bit 1: hidden. Bit 2: system file. Bit 3: volume label. Bit 4: subdirectory. Bit 5: archive. Bits 6-7: unused.
12–21	Reserved
22–23	Time (5/6/5 bits, for hour/minutes/doubleseconds)
24–25	Date (7/4/5 bits, for year-since-1980/month/day)
26–27	Starting cluster (0 for an empty file)
28-31	File size in bytes

For each directory entry, find out:

b. Type of the entry + file attributes

0x20 = 0010 0000 = archive

0x27 = 0010 0111 = archive, read only, system file, hidden

0x28 = 0010 1000 = archive, volume label *(so "MSDOS" is not a file)*

```
address data bytes ..................................  ASCII representation
                                  attribute byte

0009728 49 4f 20 20 20 20 20 20 53 59 53 27 00 00 00 00  IO      .SYS
0009744 00 00 00 00 00 00 08 5d 62 1b 1d 00 16 9f 00 00
0009760 4d 53 44 4f 53 20 20 20 53 59 53 27 00 00 00 00  MSDOS   .SYS
0009776 00 00 00 00 00 00 08 5d 62 1b 6d 00 38 95 00 00
0009792 43 4f 4d 4d 41 4e 44 20 43 4f 4d 20 00 00 00 00  COMMAND .COM
0009808 00 00 00 00 00 00 07 5d 62 1b b8 00 39 dd 00 00
0009824 44 42 4c 53 50 41 43 45 42 49 4e 27 00 00 00 00  DBLSPACE.BIN
0009840 00 00 00 00 00 00 08 5d 62 1b 27 01 f6 fc 00 00
0009856 4d 53 44 4f 53 20 20 20 20 20 20 28 00 00 00 00  MSDOS
0009872 00 00 00 00 00 00 1a 88 99 1c 00 00 00 00 00 00
0009888 46 44 49 53 4b 20 20 20 45 58 45 20 00 00 00 00  FDISK   .EXE
0009904 00 00 00 00 00 00 36 59 62 1b 02 00 17 73 00 00
```

The disk itself is called "MSDOS"

6.6 FAT file system

Bytes	Content
0–10	File name (8 bytes) with extension (3 bytes)
11	Attribute - a bitvector. Bit 0: read only. Bit 1: hidden. Bit 2: system file. Bit 3: volume label. Bit 4: subdirectory. Bit 5: archive. Bits 6-7: unused.
12–21	Reserved
22–23	Time (5/6/5 bits, for hour/minutes/doubleseconds)
24–25	Date (7/4/5 bits, for year-since-1980/month/day)
26–27	Starting cluster (0 for an empty file)
28-31	File size in bytes

For each directory entry, find out:

c. The starting cluster number

Find bytes 26-27 (decimal)

Little endian byte order!

Cluster size on FAT12 = 512 – 4096 bytes (usually 512 on floppy disks)

```
address data bytes ..................................  ASCII representation
                                        starting cluster
0009728 49 4f 20 20 20 20 20 20 53 59 53 27 00 00 00 00  IO      .SYS
0009744 00 00 00 00 00 00 08 5d 62 1b 1d 00 16 9f 00 00
0009760 4d 53 44 4f 53 20 20 20 53 59 53 27 00 00 00 00  MSDOS   .SYS
0009776 00 00 00 00 00 00 08 5d 62 1b 6d 00 38 95 00 00
0009792 43 4f 4d 4d 41 4e 44 20 43 4f 4d 20 00 00 00 00  COMMAND .COM
0009808 00 00 00 00 00 00 07 5d 62 1b b8 00 39 dd 00 00
0009824 44 42 4c 53 50 41 43 45 42 49 4e 27 00 00 00 00  DBLSPACE.BIN
0009840 00 00 00 00 00 00 08 5d 62 1b 27 01 f6 fc 00 00
0009856 4d 53 44 4f 53 20 20 20 20 20 20 28 00 00 00 00  MSDOS
0009872 00 00 00 00 00 00 1a 88 99 1c 00 00 00 00 00 00
0009888 46 44 49 53 4b 20 20 20 45 58 45 20 00 00 00 00  FDISK   .EXE
0009904 00 00 00 00 00 00 36 59 62 1b 02 00 17 73 00 00
```

0x001d = dec. 29

0x006d = dec. 109

0x00b8 = dec. 184

0x0127 = dec. 295

0x0000 – not a file

0x0002 = dec. 2

6.6 FAT file system

Bytes	Content
0–10	File name (8 bytes) with extension (3 bytes)
11	Attribute - a bitvector. Bit 0: read only. Bit 1: hidden. Bit 2: system file. Bit 3: volume label. Bit 4: subdirectory. Bit 5: archive. Bits 6-7: unused.
12–21	Reserved
22–23	Time (5/6/5 bits, for hour/minutes/doubleseconds)
24–25	Date (7/4/5 bits, for year-since-1980/month/day)
26–27	Starting cluster (0 for an empty file)
28-31	File size in bytes

For each directory entry, find out:

d. The file size in bytes

Find bytes 28-31 (decimal)

Little endian byte order!

File size is given in bytes

```
address data bytes ..................................  ASCII representation
                                          file size
0009728 49 4f 20 20 20 20 20 20 53 59 53 27 00 00 00 00  IO      .SYS
0009744 00 00 00 00 00 00 08 5d 62 1b 1d 00 16 9f 00 00
0009760 4d 53 44 4f 53 20 20 20 53 59 53 27 00 00 00 00  MSDOS   .SYS
0009776 00 00 00 00 00 00 08 5d 62 1b 6d 00 38 95 00 00
0009792 43 4f 4d 4d 41 4e 44 20 43 4f 4d 20 00 00 00 00  COMMAND .COM
0009808 00 00 00 00 00 00 07 5d 62 1b b8 00 39 dd 00 00
0009824 44 42 4c 53 50 41 43 45 42 49 4e 27 00 00 00 00  DBLSPACE.BIN
0009840 00 00 00 00 00 00 08 5d 62 1b 27 01 f6 fc 00 00
0009856 4d 53 44 4f 53 20 20 20 20 20 20 28 00 00 00 00  MSDOS
0009872 00 00 00 00 00 00 1a 88 99 1c 00 00 00 00 00 00
0009888 46 44 49 53 4b 20 20 20 45 58 45 20 00 00 00 00  FDISK   .EXE
0009904 00 00 00 00 00 00 36 59 62 1b 02 00 17 73 00 00
```

0x9f16 = dec. 40726

0x9538 = dec. 38200

0xdd39 = dec. 56633

0xfcf6 = dec. 64758

0x0000 – not a file

0x7317 = dec. 29463