MIKADOS

Principles of Operation

Dansk Data Elektronik ApS
21 September 1979

Author: Rolf Molich

Table of contents

1. Introduction

MIKADOS is a modular multiprogrammed real-time disc operating system for the ID-7000 and SPC/1 microcomputer systems manufactured by Dansk Data Elektronik ApS.

This manual contains a description of the internal MIKADOS system structure. The information in this manual enables the user to write application programs utilizing internal system information (such as new file system subroutines), and to make minor changes in the operating system, e.g. to modify an existing device driver or to write a new one.

The reader of this manual is expected to have a good knowledge of the MIKADOS system specifications as contained in the manual "MIKADOS User´s Guide" from Dansk Data Elektronik ApS.

This manual corresponds to MIKADOS version 3. september 1979.

Dansk Data Elektronik ApS reserves the right to change the specifications in this manual without warning. Dansk Data Elektronik ApS is not responsible for the effects of typographical errors and other inaccuracies that may exist in this manual. Dansk Data Elektronik Aps cannot be held liable for the effects of the implementation and use of the structures described herein.

2. General remarks

The MIKADOS system consists of a basic system monitor (process scheduler, routines for message switching etc.), a number of processes (device drivers, main storage administration and operator communication), a number of utility routines (file system, arithmetic routines etc.) used internally by MIKADOS but in many cases accessible to the application program writer, and a data area.

The MIKADOS system modules may be combined into MIKADOS systems with various characteristics using a procedure called system generation, described in the manual "MIKADOS System Generation". Among the parameters that may be varied are number of user processes, number and type of device drivers, and hardware characteristics for target computer.

2.1 System modules

A standard MIKADOS system consists of the following modules:

MIKM MIKADOS main program;
defines the contents of the interrupt addresses (addresses 0 through 3F); controls the loading of all other MIKADOS modules; defines console and printer i/o ports.

MONIT basic system monitor;
SENDs and RECEIves messages, handles general semaphore operations (SIGNL and WAIT), performs process scheduling (the process scheduling algorithm is described in section 2.8 of the MIKADOS User´s Guide); includes the dummy process which always executes a ´JMP $´ instruction.

CLOCK monitors the system clock;
measures time intervals corresponding to messages sent to the SLEEP semaphore (see "MIKADOS User's Guide", section 3.1.1); updates the real time clock.

PRIND printer driver for the Data 100 model 3400 printer or similar Centronics parallel interface compatible printer.

PRINI printer driver for serial interface printer.

PRINQ printer driver for Qume model 3/45 printer.

DISCD disc driver (all disc models).

PLATB disc description table (see section 5.2).

CONS console driver (all console terminal models).

CONSA console interrupt routine (ID-7004 interface).

CONSB console interrupt routine (ID-7012 interface or SPC/1 console port).

CONSM macro file which defines console driver process data area layout and terminal dependent control character values (cursor up, enter, break etc.).

UTIL a collection of standard resident subroutines;
INVH, INVD, INVB, TESTH, MUL, DIV, ASCII, BIDEC, FAINI, FAINH, LBSKD, VENTS, VENTH, VSEND, ASEND, VENTT, MOVE, MOVEB, MOVED, and EOLIN described in "MIKADOS Utility Programs and Subroutines",
OUT1, OUT2, IN1, IN2, BSELC, and BKTST described in section 2.2.2 and 2.2.3 of this manual.

FILS1 file system OPEN routine.

FILS2 file system CLOSE routine.

FILS3 file system READV and WRITV routines (sequential read/write).

FILS4 file system READ and WRITE routines (direct read/write).

FILS5 file system CREXT routine (create file extent).

FILS6 a collection of standard resident file system subroutines, described in section 2.2.1

RESSA ressource administration; subroutines RESRV and RELSE.

DATAA data administration; subroutines ALLOC and DELOC.

HLADM main storage administration process;
handles all ´start process´ and ´terminate process´ requests.

HLAD1 main storage administration utility subroutines;
EXIT (see section 7.2.2) and HLSND (see section 2.2.2).

OPKOM operator communication process;
processes console terminal commands; performs automatic start-up of program ´START ´ at system start-up time if this option was selected during system generation.

INIT MIKADOS initialization routine;
called directly from MIKM after system start-up;
initializes process control blocks according to a local table, initializes POOL and all semaphores, initializes the symbolic ressource data area and the area administered by the data administration.

MDATA common data area;
all process control blocks, message semaphores, general semaphores, message buffers, console data areas, and data administration area; all other data areas are defined locally as RAM data in the corresponding modules and linked into one contiguous data area by the MIKADOS linker.

SYMB macro file which contains all symbolic constants (error codes and control block offsets) used by MIKADOS.

COND macro file which defines all parameters used to control the conditional assembly during a MIKADOS system generation (see ´MIKADOS System Generation´).

2.2 Static system entry points

The MIKADOS system contains a number of static entry points described neither in the "MIKADOS User´s Guide" nor in the "MIKADOS Utility Programs and Subroutines" manual. These entry points are described in the following sections.

2.2.1 File system routines

BASIF releases the ressource '$BASIS--xx' (see section 5.3.1 and 5.3.2) corresponding to the disc described by the file control block pointed at by (HL);
exit: (A) result code returned by RELSE.

BASIR reserves the ressource '$BASIS-xx' (see section 5.3.1 and 5.3.2) corresponding to the disc described by the file control block pointed at by (HL). (C) must contain the subparameter required by RESRV;
exit: (A) result code returned by RESRV.

BCAIM stores the contents of the B, C, and A registers in addresses (M) through (M+2) (track/sector address).

ENSNV locates a file name in a given catalog sector;
entry: (HL) -> first character in file name
(DE) -> start of catalog sector buffer
(A) 0 stop search if deleted or empty file pointer found
1 stop search if empty file pointer found; ignore deleted file pointer
exit: (A) 0 search stopped by deleted or empty file pointer
-1 file pointer not found
1 file pointer found
(HL) -> file pointer in buffer.

FRIIF initializes the buffer in the file control block pointed at by (HL) to the contents of the sector at the start of the available area on a disc (see section 5.3.3).

FYLDB moves the contents of (M) through (M+2) to the C, B, and D registers (track/sector address).

GEMIB stores the contents of the (DE) register in address (HL)+(BC).

HASH computes the hash value corresponding to a file name (see section 5.3.2);
entry: (HL) -> file name followed by file type (9 characters)
(DE) -> disc description
exit: (DE) hash value; the hash value is a number between 0 and (number of catalog groups-1).

KTSØG searches the catalog for a file name;
entry: (HL) -> file control block
(A) 0 stop search if deleted or empty file pointer found
1 stop search if empty file pointer found; ignore deleted file pointer
the subroutine uses bytes ANDET through ANDET+4 in the file control block for intermediate results (see section 5.5)
exit: (A) 0 search stopped by deleted or empty file pointer
1 file name located
-1 catalog full; file name not found
other values: error codes from UDIND
(HL) -> file pointer
(BCD) track/sector address of catalog sector where file pointer is located.

MOVE3 moves 3 characters from the location pointed at by (DE) to the location pointed at by (HL); (DE) and (HL) are both increased by 2 (used for moving track/sector addresses).

PFIND searches the disc description table (see section 5.2) for a disc identification;
entry: (DE) disc identification; (E) always 'P'
(HL) -> file control block
exit: (A) 0 if disc identification legal
otherwise error code UPLAD
if the disc identification is legal, the PLBET field in the file control block will point at the disc description corresponding to the disc identification.

PLADR adds the number of sectors in (HL) to the track/sector address in (BCA); the resulting track/sector address is returned in (BCD);
on entry (DE) must point to a file control block whose PLBET field contains a pointer to the disc description for the disc in question.

PLADS checks if there is room for a new file on a disc;
entry: (HL) -> file control block;
the BASIS field in the file control block must contain the new file size;
the file control block buffer must contain the label sector for the disc in question
(DE) -> disc description
exit: (A) 0 if ok, otherwise error code EJPLD
(BCD) track/sector address of new start of available area on disc
(E) 1 if disc full after allocation
0 if disc not full after allocation.

PLLÆS reads the label sector into a file control block buffer;
entry: (HL) -> file control block
the PLBET field must point to the disc identification of the disc whose label is to be read
exit: (A) result code from UDIND.

RLEAS releases the ressource corresponding to a file;
entry: (HL) -> file name
(HL)+PLBET must point to the disc identification of the disc where the file is expected
exit: (A) result code from RELSE.

RSRVR reserves the ressource corresponding to a file;
entry: (HL) -> file name
(HL)+PLBET must point to the disc identification of the disc where the file is located
(C) reservation type code to RESRV
exit: (A) result code from RESRV.

SKRIV writes the first sector in the file control block buffer onto the disc;
entry: (BCD) track/sector address where sector is written
(HL) -> file control block
exit: (A) result code from UDIND.

TAGBC moves two bytes from the address pointed at by (HL)+(BC) to the address pointed at by (DE)

ÆNDR changes the address of the next available track/sector in the label sector for a disc and writes the label sector back onto the disc;
entry: (BCD) track/sector address of first available sector on disc after allocation
(HL) -> file control block;
the file control block buffer must contain the label sector for the disc in question
exit: (A) result code from UDIND
(BCD) track/sector address of first available sector on disc after allocation
(HLE) track/sector address of first available sector on disc before allocation = start of new file.

2.2.2 Main storage administration routines

BKTST tests if a buffer lies partly or completely within the bank switch area;
entry: (HL) -> buffer start
exit: (CY) 1 if buffer start >= bank switch area start - 100
0 otherwise;
the starting address for the bank switch area is defined during system generation (BKLIM in module COND).

BSELC determines the bank corresponding to the process whose main or auxiliary semaphore is pointed at by (HL) on entry; the bank is selected and connected to the process by issuing a ´ OUT BANKR´ instruction, by storing the bank select code in BANKI (see section

4.3), and by placing the bank select code in the PKBNK field of the process control block for the calling (active) process; the bank select code is returned in (A); (BC) is not changed by BSELC.
This subroutine is used by all device driver processes in bank switched systems to ensure that the buffer area of the calling user process is permanently available during the processing of an i/o request.

HLSND sends a message to the main semaphore of the main storage administration process;
entry: (A) request code
(BC) address of answer semaphore

2.2.3 Miscellaneous routines

IN1 inputs (A) from i/o port (E); note 1.

IN2 inputs (A) from i/o port (E)+1; note 1.

OUT1 outputs (A) to i/o port (E); note 1.

OUT2 outputs (A) to i/o port (E)+1; note 1.

Note 1: before calling subroutines IN1, IN2, OUT1, OUT2 the interrupt system must be disabled using a DI instruction; upon return from these subroutines a EI instruction must be executed.

2.2.4 MIKADOS data area

ACTIV	start of ready queue (see section 4.1)
ASEMA	general semaphore used to reserve arithmetic processing unit (APU) in COMAL systems
BANKI	current bank mask (see section 4.3)
CxMES	main semaphore for console driver x (x = 1,2,...)
DIMES	main semaphore for disc driver
DISEM	general semaphore used for communication between disc interrupt handler and disc driver process
HLAKT	see section 7.2
HLMES	main semaphore for main storage administration process
HVENT	see section 7.2
MIKSL	end of MIKADOS data area
OPMES	main semaphore for operator communication process
P1MES	main semaphore for printer driver
RUNN	pointer to process control block for active process (see section 4.3); also start of MIKADOS data area

2.3 Accessing system control blocks

When addressing data fields in a system control block (process control block, file control block etc.) the user should always use symbolic offsets to ensure that his programs will run under future versions of MIKADOS. The symbolic name for each field offset is given in the control block descriptions in the following chapters. The symbolic names are defined in an assembler macro file called ´SYMB´, which is delivered as a part of each MIKADOS system.

Example: increasing the priority of the current process should be done with the following sequence

```
LHLD    RUNN    ;see section 4.3
LXI     D,PRIO  ;do not write LXI D,4
DAD     D
DCR     M       ;increase priority by
                ; decreasing value
```

3. Synchronization and communication between processes

MIKADOS processes may exchange information and achieve synchronization by communicating via messages or general semaphores as outlined in chapter 2 of the "MIKADOS User´s Guide".

The MIKADOS system monitor and in particular the MIKADOS synchronization and communication routines are based on the MIK system devised and programmed by Bodil Schrøder, Institute of Datalogy, University of Copenhagen, in May 1975. A thorough discussion of the theoretical background and ideas behind this system is given in the report "MIK - et korutineorienteret styresystem til en mikrodatamat" ("MIK - a coroutine oriented control system for a microcomputer"), which is available from the Institute of Datalogy.

3.1 Semaphore and message format

The format of a general or message semaphore is:

byte 0	semaphore value (see below)
byte 1	number of elements waiting for semaphore
byte 2 - 3	pointer to first element waiting for semaphore (undefined if no elements are waiting)
byte 4 - 5	pointer to last element waiting for semaphore (undefined if no elements are waiting)

The value of a general semaphore may be 0, 1, 2, ..., and the number of elements is always the number of processes waiting for the general semaphore. If processes are waiting for the semaphore the value of the semaphore will be 0. If no processes are waiting, the value will be the total number of SIGNLs minus the total number of WAITs issued to the general semaphore.

The value of a message semaphore may be -1, 0 or 1. The elements waiting for the semaphore may be messages (value = 1) or processes (value = -1). If no one is waiting for the semaphore the value will be 0.

Process control blocks waiting for a semaphore are chained using the pointer in byte 0 - 1 of the process control block. Messages waiting for a semaphore are chained using byte -2 - -1 in the message. Internal MIKADOS message pointers always point to byte -2 of a message.

The message format is described in detail in section 2.2 of the "MIKADOS User's Guide".

3.2 Symbolic ressources

The data area describing the reserved symbolic ressources is not accessible from user programs.

The layout of the data area is as follows:

```
STADD:: DS      2           ;pointer to byte 10 in first
                            ; ressource element
LAST::  DS      2           ;pointer to byte 10 in last
                            ; ressource element
        DS      1           ;for internal use
RRTA::  DS      1           ;number of active elements,
                            ; i.e. elements corresponding
                            ; to a reserved ressource
RRT::   DS      RANTL*RRLEN ;ressource elements
```

The layout of a ressource element is

byte 0 - 9 ressource name
byte 10 - 11 pointer to byte 10 of next ressource element
byte 12 flags
bit 0: 1 if reservation exclusive
bit 1 - 7: number of reservations

The active elements always appear first in the chain of ressource elements.

The general semaphore RESEM must be reserved before any access to the ressource elements is made.

3.3 Monitor utility routines

This section describes the MIKADOS Monitor entry points. Note that some of these entry points are not static.

The queue descriptor used by some of the utility routines must have the following format:

byte 0 : number of elements in queue
byte 1 - 2: pointer to first element in queue
byte 3 - 4: pointer to last element in queue

Note the similarity between a queue descriptor and a semaphore.

COMM MIKADOS scheduler - see section 4.0;
before jumping to the scheduler the registers belonging to the active process must be saved on the stack in the order push psw, push b, push d, push h. Further, the active process must be entered into a system queue.

FIRST removes the first element from a queue; the address of the removed element is returned in (DE);
entry: (BC) -> queue descriptor
exit: (DE) -> removed element
(A) 1 if queue empty (no element removed)
0 if queue not empty.

INTAC enters a process into the ready queue;
entry: (DE) -> process control block
(A) 1 enter process last in queue for process priority level and set new time slice in KVANT field
0 enter process first in queue for process priority level; leave priority field (KVANT) unchanged
exit: (DE) unchanged.

INTO enters a process control block into a queue;
entry: (BC) -> queue descriptor
(DE) -> process control block
exit: (DE) unchanged.

RECEI see "MIKADOS User´s Guide".

SEND see "MIKADOS User´s Guide".

SND2 see section 6.3.

SIGNL see "MIKADOS User´s Guide".

SIG2 see section 6.3.

WAIT see "MIKADOS User´s Guide".

4. Process control

A MIKADOS process is described by a process control block. Any process in the system will always be in either of two states: 'ready' to execute program instructions or 'not ready', i.e. inhibited for one of the following reasons:

1) waiting for program to be read into memory (during process start-up)
2) waiting for a message
3) waiting for a general semaphore
4) inactive (see section 7.2; implemented as waiting for the general semaphore DEAD, which is never signalled)

The ready processes are given access to the CPU, i.e. made active, according to their priority. At any time the ready process with the highest priority (lowest numerical value of the PRIO field in the process control block) will be active. If several ready processes exist at the highest priority level, the CPU is multiplexed between these processes.

The scheduler operates as follows: Initially, the highest priority level at which ready processes exist is determined. The first process at this level in the ready queue (see section 4.1) is removed from the ready queue and made active. If this process is still active after the time slice has expired it is deactivated, and entered at the end of the priority level queue. The next process in the priority level queue is then made active etc.

4.1 The ready queue

The ready queue contains one entry for each legal priority level in the MIKADOS system. Each entry occupies 5 bytes, which are used as follows:

byte 0: number of ready processes at this priority level

byte 1-2: address of first ready process control block at this priority level

byte 3-4: address of last ready process control block at this priority level (used to enter new processes at the end of the queue)

The contents of byte 1-4 are undefined if byte 0 is zero.

The first byte in the ready queue has the static entry point label ACTIV. The entry for priority level N occupies bytes N*5 through N*5+4 in the queue.

4.2 The process control block

The data structure which contains all internal information about the execution of a program by a process is called a process control block (PCB). The layout of a PCB is:

byte 0 - 1 : pointer to next PCB in the same queue (ready queue or semaphore queue)

byte 2 - 3 (STPIL): pointer to top of stack if process is not active; meaningless if process is active

byte 4 (PRIO) : process priority in 4 least significant

bits; 4 most significant bits reserved for future extensions

byte 5 - 6 (BSKED): pointer to last message RECEIved by this process; used only by FABUF

byte 7 (KVANT): length of remaining time slice in system time units

byte 8 - 9 (SBUND): pointer to start of stack area (absolute bottom of stack)

byte 10 - 11 (STTOP): pointer to end of stack area (STCK1; absolute top of stack)

byte 12 - 13 (ARBJD): pointer to local data area for process; used mainly if several processes execute the same reentrant piece of code (device driver); 0 if no data area defined

byte 14 - 15 (KOSEM): pointer to the main semaphore for this process; 0 if no such semaphore defined

byte 16 - 17 (HJSEM): pointer to the auxiliary semaphore for this process; 0 if no such semaphore defined

byte 18 - 19 (PKDIV): local process information; used mainly if several processes execute the same reentrant piece of code (used e.g. by certain device drivers to store i/o port addresses)

byte 20 (PKBID): process control block identification (letter A - Z, a - z); upper case letter indicates system process (short stack), lower case letter indicates user process (long stack)

byte 21 - 22 (PKBNK): bank code used to select this process; byte 21 is output to address 0FC, byte 22 meant for output to address 0FD (currently not implemented); this field is set by certain device

drivers to ensure that the bank containing the user process i/o buffer is always selected when the driver is active (see section 2.2.2, BSELC)

byte 23 - 24 (PKKÆD): pointer to next PCB in the same main storage administration chain (see section 7.2)

byte 25 - 26 (PKBEG): pointer to start of program area

byte 27 - 28 (PKEND): pointer to end of program area (points at first byte which does not belong to the program region)

byte 29 - 31 (PKPRG): track/sector address of program file + 1 (points at first sector containing code)

byte 32 - 33 (PKPLA): pointer to disc descriptor for disc containing program file

byte 34 (STCK1): absolute top of stack (contents of this byte always FF, used to detect stack overflow)

The process stack is located immediately after the process control block area. The length of the stack is SSTKL bytes for system processes, and BSTKL bytes for user processes. The bottom of the stack (first element pushed on stack) is located at address (SBUND-2, SBUND-1). The bytes at address (SBUND) and (STTOP) both contain FF. If a process is not active the current top of stack, (STPIL), contains the (HL) register for the suspended process while

(STPIL) + 2 contains (DE)
(STPIL) + 4 contains (BC)
(STPIL) + 6 contains (A) and flags
(STPIL) + 8 contains address of next instruction to be executed by process

4.3 Other important process related data

RUNN always contains a pointer to the active PCB. The standard sequence to access a field in the process control block of the active process is

```
LHLD    RUNN
LXI     D,symbolic field name
DAD     D       ;HL now points to the first
                ; byte in the field
```

BANKI always contains the bank select pattern used to select the bank in which the active process is running.

4.4 Standard priority assignments

The standard system is configured with 10 priority levels in the ready queue. The number of priority levels may be changed (maximum 16) without problems (change AKTIV in MONIT, and MAXPR in SYMB) but the resulting MIKADOS systems will not be standard systems.

Priority level	used by
0	clock process
1	all console driver processes
2	printer driver process; also used by user processes during EXIT (see section 7.2.2)
3	main storage administration process
4	operator communication and certain user processes (XREF during initiation to ensure that XREF always runs before any other

scheduled process)

5 user processes (EDIT after initiation)

6 user processes (standard level after initiation by OPKOM or MONITOR)

7 cpu bound processes (XREF after initiation) and disc driver (disc driver only required to be at this low level if the SYKES subdriver is included because this driver uses a wait-for-flag transmission method (active wait))

8 dummy process

9 suspension level - any process may be suspended by setting its priority to 9 (currently not used)

5. The file system

This chapter contains a detailed description of the MIKADOS file system.

Section 5.2 describes the disc description table, which contains a description of the disc units accessible from a MIKADOS system by means of the disc driver.

Section 5.3 describes the disc layout, i.e. the label sector, the file catalog structure and the general file structure.

Section 5.4 describes the detailed file format.

Section 5.5 describes the file control block, which is a data area used by all file system routines to store information about absolute file addresses and file characteristics. The file control block also contains a buffer used to block and unblock records from disc sectors.

5.1 General remarks

An absolute track/sector address on a disc always occupies 3 bytes. The first two bytes indicate the track number (least significant byte first) while the last byte contains the sector number.

All MIKADOS read/write operations from/to a disc are performed using the UDIND routine described in the "MIKADOS User´s Guide".

The disc driver consists of 4 parts:

1) the main driver which receives disc i/o requests from other processes, checks their validity using the disc description table, and converts them into one or more sub-driver calls

2) one or more subdrivers which are capable of performing one contiguous read/write operation on a specific disc type (including head positioning and error retry)

3) an interrupt routine which distributes disc interrupts to the appropriate subdrivers

4) the disc description table which contains physical and logical information about the type and structure of a disc

5.2 The disc description table

The disc description table contains a logical and physical description of the disc units accessible from a MIKADOS system by means of the disc driver.

The disc description table starts at address PLTAB (accessible as a nullfile). The table format is:

```
PLTAB:>DB       number of elements in table
       DB       0        ;reserved for future use
;
       DS       PLSTR    ;description of disc 1
       DS       PLSTR    ;description of disc 2
       ...
       DS       PLSTR    ;description of last disc
```

The number of bytes from PLTAB to the start of the first disc description has been equated to the symbol PLTB1; the current value of PLTB1 is 2.

The description of a disc, an area of length PLSTR bytes, contains the following information:

byte 0 - 1		disc identification: 'Px' where 'x' is an ASCII digit (1-9)
byte 2 - 3	(PSPOR)	number of logical tracks on disc
byte 4 - 5	(KGRUP)	number of catalog groups - 1; number of catalog groups must be a power of 2
byte 6	(SPRGP)	number of sectors in a catalog group
byte 7	(ASPSP)	number of sectors per logical track
byte 8 - 9	(SUBDR)	address of subdriver entry point
byte 10	(FPLID)	physical disc identification (subdriver dependent)
byte 11	(DSCID)	disc type identification 'P' - Pertec cartridge disc 'F' - Sykes floppy disc 'M' - BASF mini disc 'B' - BASF disc
byte 12		currently not used
byte 13	(OPTIO)	transfer characteristics for subdriver bit 7: 1 if read-after-write test must be possible (number of sectors transferred should not exceed length of auxiliary buffer) bit 6: 1 if a subdriver request may extend beyond the end of a track bit 5: 1 if a subdriver request may not involve crossing the middle of a track bit 4-0: currently not used

5.3 Disc layout

A disc contains a label sector (always track 0, sector 0), a catalog, and a file area.

5.3.1 The label sector

Track 0, sector 0 of a logical disc always contains the label sector, which among other information contains the disc label and the address of the first available track/sector on the disc.

The symbolic ressource '$BASIS--xx', where 'xx' is the logical disc identification (e.g. 'P2'), must be reserved before any access to the label sector is made. This ressource must be reserved exclusively if the label sector is to be written or updated. The MIKADOS system routines always wait for this ressource if it is not immediately available.

The label sector contains the following information:

byte 0 - 9	(PLIDN)	MIKADOS initialization program identification ('PLADELAGER')
byte 10 - 12	(LEDIG)	track/sector address of first unused sector on disc
byte 13 - 17	(PLART)	disc type; 5 ASCII characters, last character indicates master disc ('*') or back-up disc ('0')
byte 18 - 27	(PLDTO)	date when last back-up of disc was taken
byte 28 - 37	(PLBTG)	disc name; 10 ASCII characters
byte 38 - 47	(PNUDA)	date of last system startup (only if disc mounted in P1)

5.3.2 The catalog

The disc catalog starts immediately after the label sector, i.e. in track 0, sector 1. The size of the disc catalog is given in the disc description table (number of catalog groups * number of sectors in a catalog group).

The symbolic ressource ´$BASIS--xx´, where ´xx´ is the logical disc identification (e.g. ´P3´), must be reserved before any access to the catalog is made. This ressource must be reserved exclusively if the catalog is to be updated. The MIKADOS system routines always wait for this ressource if it is not immediately available.

The catalog is indexed using a hashing scheme. The hash value is computed by applying a hashing algorithm to the file name (including file type). For details about the hashing algorithm consult a listing of the HASH subroutine in the FILS6 module. The computed value is used to index a catalog group where the search for the file name starts.

A catalog group consists of one or more sectors as defined in the disc description table. A catalog sector contains 19 file pointers of 13 bytes each. The last 256-19*13 = 9 bytes are unused. The format of a file pointer is:

byte 0 - 7	file name; if byte 0 is 0 or 1 then the file pointer is available (0 - empty, 1 - deleted) and bytes 1 - 12 have no significance
byte 8	file type
byte 9	the ASCII character A (extent code for first extent)
byte 10 - 12	track/sector address of first sector in file (nullfiles: value in byte 10 - 11, byte 12 not used)

The search for a specific file continues until the file is found or until the first empty (byte 0 = 0) file pointer is encountered (see section 2.2.1, ENSNV and KTSØG).

The search for an available file pointer continues until the first empty (byte 0 = 0) or deleted (byte 0 = 1) file pointer is encountered (see section 2.2.1, ENSNV and KTSØG).

Catalog sector boundaries are ignored during a catalog search. After the last catalog sector has been examined, the search continues with the first catalog sector until all catalog sectors have been searched.

When a disc is initialized all bytes in the catalog are set to zero. When a file is deleted, byte 0 of the corresponding file pointer is set to 1.

5.3.3 The file area

The file area contains the files described in the catalog. The file format is described in section 5.4. The first byte and the bytes with offset NÆFIL, NÆFIL+1 and NÆFIL+2 (see section 5.4) in the first sector after the last file sector on the disc, contain 0.

5.4 File format

A file consists of a base file followed by zero, one or more extents.

The base file and the extents all consist of the same number of sectors, the extent size. The base file and the extents all start with a 32-byte MIKADOS file information area followed by 224 + (extent size - 1)*256 bytes of user information structured as the user chooses (the record format is described in section 5.4.1). Extent switching is handled automatically by the system.

The layout of the MIKADOS file information area is:

byte 0 - 7		file name; byte 0 = 1 indicates that the file has been purged; byte 0 = 0 marks the start of the available part of the file area
byte 8		file type
byte 9		extent code ('A' in base file, 'B', 'C', ... for extent 1, 2, ...)
byte 10	(FILUD)	(only in base file) total number of extents
byte 11 - 12	(BASIS)	number of sectors in base file (= number of sectors per extent)
byte 13 - 15	(FILNÆ)	absolute disc address of next extent
byte 16 - 18	(FILFO)	absolute disc address of previous extent
byte 19 - 21	(NÆFIL)	absolute disc address of next file on disc
byte 25 - 26	(POSTL)	record length defined when file was created, in bytes
byte 27 - 31		currently not used

The symbol FBUFF has been equated to the length of the file information in bytes (currently FBUFF = 32).

During the OPENing or CREATion of a file, MIKADOS reserves a symbolic ressource corresponding to the file. The first 8 characters of the symbolic ressource name are equal to the file name, the 9th character is the file type, and the 10th character is the last character in the disc identification for the disc on which the file resides. The symbolic ressource is released by CLOSE.

5.4.1 Record format

The user file area in a direct access file does not contain any system information, i.e. the first byte in a record follows immediately after the last byte of the preceding record.

In a sequential file a user record is preceded and followed by a byte containing the length of the user record in bytes. End-of-file is indicated by two subsequent bytes containing a zero.

The algorithm used to read a sequential record is:

1) examine next byte in file; if zero return with an end-of-file indication

2) set file control block record length field (PSLGD) to 1; read next byte (length of sequential record)

3) set file control block record length field (PSLGD) to sequential record length + 1; read record.

5.5 The file control block

The file control block is a data area created by an application program used by file system routines to store and retieve information about absolute addresses and characteristics for a particular file. The file control block is also used to block and unblock records from disc sectors.

The layout of the file control block is as follows:

byte 0 - 7		file name
byte 8		file type
byte 9		extent code corresponding to the extent currently accessed
byte 10	(FILUD)	total number of extents
byte 11 - 12	(BASIS)	number of sectors in base file (= number of sectors per extent)
byte 13 - 15	(FILNÆ)	absolute disc address of next extent
byte 16 - 18	(FILFO)	absolute disc address of previous extent
byte 19 - 21	(SEKT1)	absolute address of first sector currently in the file control block buffer area
byte 22	(SANT)	number of file sectors currently in buffer
byte 23	(BFLGD)	buffer length in sectors
byte 24	(DFLAG)	flags

bit 0: 1 - file opened exclusively
0 - file opened for reading only

bit 1: 1 - a WRITE operation has modified the file contents (the buffer must be written back onto the disc)
0 - the buffer has not been modified

		bits 2-7: currently not used
byte 25 - 27	(FILDE)	absolute disc address of first sector in current extent
byte 28 - 29	(NRPST)	number of next record to be read/written (meaningful only for direct access files); first record in file is number 1
byte 30 - 31	(NÆPST)	relative buffer address of first byte in next record to be read/written
byte 32 - 33	(PLBET)	pointer to disc descriptor for disc on which file resides
byte 34 - 35	(PSLGD)	actual record length in bytes
byte 36 - 40	(ANDET)	used internally by certain file system subroutines
byte 41	(DKÅBN)	open flag; file control block is open if this byte contains the 1´s complement of byte 0
byte 42 - 44	(FILBA)	absolute address of first sector in base file
byte 45 - 47		currently not used
byte 48 - (BFLGD)*256+47		file control block buffer

The symbol BUFF has been equated to the length of the file control block information area in bytes (currently 48).

6. Driver structure

A MIKADOS device driver processes i/o requests to an external device (e.g. display terminal, printer, disc). The driver acts in response to messages received from other MIKADOS processes. The general message format is described in detail in section 2.2 of the "MIKADOS User´s Guide".

A MIKADOS device driver consists of a driver process and usually an interrupt handler.

After system start-up the driver process initializes the device and device interface.

The driver process receives i/o requests from other processes through its main semaphore and checks the requests for validity (operation code, buffer length and device dependent subparameters). If the message is found to be valid the i/o operation is initiated. Upon completion of the i/o request, the driver process constructs the resulting status information and inserts it into the original request, which is then returned to the answer semaphore.

The interrupt handler, if present, handles all device interrupts. Upon encountering an interrupt, the MIKM module immediately transfers control to the start of the interrupt handler, which is responsible for saving the registers on the stack, continuing or completing the operation, and reestablishing the register contents before control is transferred back to the interrupted process. If an interrupt signals the completion of an i/o operation, or if other circumstances so dictate the interrupt handler may invoke the device driver process to continue or complete the i/o operation.

The reader of this manual is encouraged to consult the MIKADOS driver program source listings delivered as a part of the system generation option for concrete examples of the general methods discussed in this chapter.

6.1 Wait for flag transfer

This method is used if the device accepts data so fast that the overhead used in processing interrupts would slow the device considerably, if the device interrupt scheme is too complicated to be handled by a reasonably simple interrupt handler, or if the device is not capable of interrupting the central processor.

Device drivers using this i/o transfer method do not have an interrupt handler and do not use the INTEL 8080/8085 interrupt system.

The transfer takes place with the interrupt system enabled. The driver process activates the device and waits actively (without releasing the cpu for use by other processes of lower priority) until the device has completed the operation, e.g.:

```
(1)     OUT:    OUT     ADDR    ;OUTPUT BYTE
        TEST:   IN      ADDR    ;GET DEVICE STATUS
                ANI     MASK    ;TEST IF DEVICE IS READY TO
                                ; ACCEPT MORE DATA
                JZ      TEST    ;JUMP IF THIS IS NOT THE CASE
                ...             ;GET OR COMPUTE NEXT BYTE TO
                ...             ; BE OUTPUT
                JMP     OUT     ;OUTPUT NEXT BYTE
```

If it is known that the device takes some time to complete an operation (e.g. process a particular control byte), the above construct may be refined in order to avoid excessive waste of cpu time in the active wait loop:

```
(2)     OUT:    OUT     ADDR    ;OUTPUT BYTE
        TEST:   LXI     H,£10   ;WAIT FOR 100 MS
                CALL    VENTT   ;WAIT UNTIL DEVICE READY
                IN      ADDR    ;GET DEVICE STATUS
                ANI     MASK    ;TEST IF DEVICE IS READY TO
                                ; ACCEPT MORE DATA
                JZ      TEST    ;JUMP IF THIS IS NOT THE CASE
                ...             ;GET OR COMPUTE NEXT BYTE TO
                ...             ; BE OUTPUT
                JMP     OUT     ;OUTPUT NEXT BYTE
```

In order to ensure fast servicing of the device the wait time should be approximately one third of the expected total device processing time if the device processing time is unknown to the driver process. If the exact device processing time is known, the wait time should be set to the closest value greater than the device processing time.

Of course, methods (1) and (2) may be combined, using method (1) to transfer data bytes and method (2) to transfer control bytes as demonstrated by the following example.

The Data 100 printer driver (PRIND) uses this method. All bytes except CR are transferred using method (1). The CR byte, which starts the printer, is transferred using method (2) with a wait time of 80 ms. This driver also takes advantage of the fact that after the CR byte has been transferred the whole output line is stored in the internal printer buffer, i.e. the answer message may be transmitted to the user, who may use the wait time used to print the current line to prepare the next line (double buffering).

The Qume printer driver (PRINQ) also uses this method but in a different way. The Qume printer possesses an internal buffer which is used to store the next 16 commands to be executed by the printer. The time to store a command in the internal buffer is 1000 microseconds or less, while the time required to execute a command is of magnitude 10 ms, i.e. the time to empty the buffer is 150 ms or more. The driver uses method (1) to transfer data until the waiting time to transfer one byte exceeds 1 ms. When this situation occurs the driver waits for 150 ms before attempting to transfer another byte.

6.2 Interrupt controlled transfer

In this method the INTEL 8080/8085 interrupt system is used to signal the completion of an operation by an external device.

The device driver process must initiate the i/o operation and enable the interrupt level corresponding to the device interface before waiting for an interrupt message. This must be done using the sequence (for standard INTEL 8080 levels):

```
DI              ;INTERRUPT MASK SHOULD BE
                ; MODIFIED ONLY WHILE INTERRUPT
                ; SYSTEM DISABLED
IN      INTRP   ;READ OLD INTERRUPT MASK
                ; INTRP = OFE - DEFINED IN SYMB
ORI     LEVEL   ;TURN ON APPROPRIATE BIT
                ; (BIT 'N' ENABLES LEVEL 'N')
OUT     INTRP   ;OUTPUT NEW MASK
EI
```

or the sequence (for the special INTEL 8085 levels):

```
        DI
        RIM             ;READ 8085 INTERRUPT MASK
        ANI     MASK    ;PRESERVE OTHER LEVELS
        ORI     LVL     ;LEVEL 5.5, LVL = 08, MASK = 06
                        ;LEVEL 6.5, LVL = 08, MASK = 05
                        ;LEVEL 7.5, LVL = 08, MASK = 03
        SIM             ;SET 8085 INTERRUPT MASK
        EI
```

The driver is notified by the interrupt handler about the occurrence of an event which requires driver process action by a message (slow, but with possibility for transferring information; example: console driver, printer driver), or by a signal to a general semaphore (fast; example: disc driver).

The driver process must always receive its messages from the interrupt handler using the auxiliary semaphore. The auxiliary semaphore may also be used to receive timer messages thus implementing a time out facility (not implemented in any current MIKADOS driver). The communication between interrupt handler and device driver process is discussed in section 6.3.

After the transfer is completed, the disc driver process or the interrupt handler must reset the interrupt level to prevent spurious interrupts.

6.3 Interrupt handlers

When an interrupt occurs, the microprocessor disables the interrupt system (corresponding to a 'DI' instruction), stacks the address of the next instruction to be executed and continues execution at an address determined by the interrupt level.

The contents of the addresses corresponding to the interrupt levels is determined by the MIKM module. Usually, control is immediately transferred to the appropriate interrupt handler, which is responsible for all further processing of the interrupt.

The interrupt routine must start by saving the registers on the stack. If the interrupt routine calls the MIKADOS process scheduler (COMM) the registers must be saved in the standard order

```
PUSH    PSW
PUSH    B
PUSH    D
PUSH    H
```

Otherwise only the registers used by the interrupt handler need to be saved (in any order).

The interrupt handler may signal to a semaphore using the sequence

```
LXI     B,SEMA   ;POINTER TO GENERAL SEMAPHORE
CALL    SIG2     ;SIGNAL - NOTE: NOT   CALL SIGNL
```

The interrupt handler may send a message to the driver process using the sequence

```
LXI     B,POOL   ;GET A MESSAGE BUFFER
INX     B
CALL    FIRST
ANA     A
JNZ     ERROR    ;ERROR EXIT IF 'POOL' EMPTY
...              ;(DE) POINTS TO BYTE -2 IN
...              ; MESSAGE BUFFER
...              ;LOAD MESSAGE CONTENTS
XCHG             ;POINTER TO MESSAGE NOW IN (HL)
LXI     B,SEMA   ;POINTER TO AUX SEMAPHORE IN
                 ; DRIVER PROCESS
CALL    SND2     ;SEND - NOTE: NOT  CALL SEND
```

After a call to SIG2 or SND2, (A) will be non-zero if the MIKADOS scheduler should be invoked.

After performing the necessary operations the interrupt handler returns control to the system. If the driver has not made any system calls and if the registers were saved in standard order, execution of the active process is resumed with the sequence:

```
POP     H        ;RESTORE REGISTER CONTENTS
POP     D
POP     B
POP     PSW
EI
RET
```

If the driver has made one or more system calls, and has been notified by the system that rescheduling is necessary ((A) <> 0 after call of SIG2 or SND2) the following sequence should be used instead:

```
LHLD    RUNN    ;INSERT ACTIVE PROCESS IN
XCHG            ; READY QUEUE
SUB     A
CALL    INTAC
JMP     COMM    ;INVOKE SCHEDULER
```

7. Main storage administration

7.1 Dynamic data allocation

The data area administered by the dynamic data allocation routines ALLOC and DELOC is located in the MIKADOS data module (MDATA).

The structure of the data area is:

```
DLAL1::DS      2        ;pointer to first available data area
                        ; (zero if no such area exists)
       DS      2        ;length of this data area (always zero
                        ; to prevent the allocation of this
                        ; area)
;
       DS      1        ;filler byte - prevents the above area
                        ; from being combined with the
                        ; following
;
DLA1:: DS      2        ;data area - size may be choosen during
DLA1A::DS      SIZE     ; system generation
;
MIKSL:>                 ;end of MIKADOS data area
```

The purpose of the first 5 bytes is to eliminate the need for a special case in the allocation/deallocation algorithms if the whole data area has been allocated.

The structure of an available dynamic data area is

byte 0 - 1: pointer to next available data area with a higher address than this one (zero if no such area exists)

byte 2 - 3: total length of this data area in bytes (including the pointer and length bytes)
byte 4 and following: undefined

When a data area is released the system always checks if the data area lies immediately before or immediately after another available data area. If this is the case the adjoining available data areas are combined to form one available data area.

No check is performed concerning the legality of a data area release, i.e. any memory area may be released.

7.2 Main storage administration process

This section contains a step-by-step description of some of the most complex activities performed inside the MIKADOS system, the program scheduling and termination requests handled by the main storage administration process.

The main storage administration process maintains two important queues:

- a queue of waiting processes, i.e. a queue of processes for which a ´start process´ request has been received, but whose corresponding program has not yet been read into memory. A pointer to the first process control block in this queue may be found in HVENT.

- a queue of running processes, i.e. a queue of processes that have executed one or more instructions but who have not yet issued a ´terminate process´ request. A pointer to the first process control block in this queue may be found in HLAKT.

The processes in these queues are chained using the PKKÆD field in the process control block. The PKKÆD field for the last process in a queue contains 0.

7.2.1 Start a process

This operation, which is initiated by a message to the main storage administration process, comprises the following steps:

1) check if value of requested priority is legal, i.e. less than or equal to MAXPR, and greater than or equal to MINPR (MAXPR and MINPR defined in SYMB)

2) check legality of disc identification

3) open program file; exit if OPEN unsuccessful

4) check that MIKADOS ending address (MIKSL) <=
program starting address <=
program entry point address <
program ending address

5) check that file has sufficient sectors to contain program; number of sectors = (ending address - starting address + 256) / 256 + 1

6) check that requested program bank defined during system generation (BKMSK); if the bank code is zero check that system was generated without provision for bank switching (BKMSK=0); a bit in the BKMSK byte is set if the bank selected by the bit is installed in the computer

7) (bank switch systems only) connect appropriate bank to main storage administration process

8) check that program starting address and (program ending address - 1) correspond to installed RAM memory

9) check that a passive process control block is available (internal variable HLTÆL > 0); if yes decrease HLTÆL by 1, get process control block from semaphore DEAD, and increase semaphore value of DEAD by 1

10) initialize new process stack; set registers to zero and insert program entry point address

11) insert priority and bank code into process control block

12) insert process control block into queue of processes waiting for start-up (pointer to first process in HVENT, processes chained using PKKÆD field in process control block); the processes appear in the queue according to their priority (lowest value of PRIO field first)

13) insert absolute disc address of second sector in program file into process control block

14) insert main storage limits into process control block

15) execute main storage administration scheduler for the bank in which the new process should run (see section 7.2.3)

7.2.2 Terminate a process (EXIT)

A process terminates by calling EXIT. The following describes the steps taken by MIKADOS to terminate a process. Steps 1 - 3 take place in EXIT, the remaining steps take place in the main storage administration process.

1) change the priority of the calling process to be higher than that of the main storage administration process in order to ensure that the main storage administration process does not become active during steps 2 - 3

2) send a 'terminate process' message to the main storage administration process

3) issue a WAIT call to the DEAD semaphore; this causes the calling process to be suspended

4) (this and the following steps are performed by the main storage administration process)
check the contents of the PKBID field of the process control block for the process to be terminated; the field must contain a lower case letter (user process)

5) increase number of available process control blocks by 1 (increase HLTÆL)

6) remove process control block from the queue of running processes

7) execute main storage administration scheduler for bank in which process was running (see section 7.2.3)

7.2.3 Main storage administration scheduler

The main storage administration scheduler (subroutine HLLVA, not externally accessible) is called by the main storage administration process after each 'start process' and 'terminate process' request to check if a waiting process can be activated in the bank which was affected by the operation.

The algorithm is outlined below:

```
repeat for each process in the queue of waiting processes
  if (bank for waiting process = bank where change occurred) then
    begin
    get main storage limits for waiting process;
    ok := true;
    repeat for each process in the queue of running processes
      if (bank for running process = bank where change occurred) then
        begin
        compare main storage limits for running and waiting process;
        if overlapping then ok := false;
        end;
    until (not ok) or (all running processes examined);
    if ok then
      begin
      read program into memory using disc address in PKPRG and
        memory address in PKBEG;
      remove process control block from queue of waiting processes;
      insert process control block at beginning of queue of running
        processes;
      insert process control block into ready queue;
      end;
    end;
until all waiting processes examined;
```

8. Program file formats

This section describes the data format of various important types of system files.

Section 8.1 explains in detail the format of a relocatable file. Relocatable program files are produced e.g. by the assembler during the assembly of a source program. Relocatable program files contain sufficient information about a program to create an executable version of the corresponding program anywhere in memory. All local program addresses are relative and all information about external symbols has been preserved.

Section 8.2 contains a description of the format of an absolute (executable) program file produced by the linker.

8.1 Relocatable program file format

A relocatable file is a sequentially organized type R file. The file may have extents.

The first byte in a record (after the length indicator) contains the record type. This byte is followed immediately by zero, one or more bytes of information.

The record types are described in the following sections.

The first record in a relocatable file will be a Name record. All Entry point and External symbol records will appear immediately after the Name record without any records of other types in between.

A relative program address in a relocatable program file consists of 3 bytes:

byte 0: address type
01 address is relative to the value contained in the RAM base register when linking of this module was started
02 address is relative to the value contained in the ROM base register when linking of this module was started
03 address is absolute
byte 1-2: address value (relative or absolute as indicated by byte 0)

8.1.1 Name record

This record defines module name, module length, and execution start address.

byte 0 type code (2)
byte 1- 8 name of source file used to assemble this program
byte 9-11 relative address of first byte after ROM section of program
byte 12-14 relative address of first byte after RAM section of program
byte 15 execution start address indicator
00 no start address given
FF start address given
byte 16-18 (meaningful only if byte 15 = FF)
relative address where program execution should begin

8.1.2 Entry point record

This record describes an externally accessible symbol (entry point) defined in this module.

byte 0 type code (4)
byte 1- 8 symbol name in ASCII
byte 9-11 address of symbol (relative or absolute)
byte 12 type indicator
08 normal entry point symbol
10 static entry point symbol

8.1.3 External symbol record

Describes a symbol to which references are made in the current module, but which is not defined within the current module.

byte 0 type code (5)
byte 1- 8 symbol name in ASCII
byte 9-10 number used to reference symbol throughout this module. Note: the linker currently does not use this number but assigns numbers to the external symbols in their order of appearance starting with 1
byte 11 (not used)

8.1.4 Define base address record

Indicates where in the absolute program module the following code should be placed.

```
byte  0    type code (7)
byte  1    bit 0-1  new base type
                    01 RAM
                    02 ROM
           bit 2    absolute base address supplied
                    1  yes
                    0  no
byte  2- 3 (meaningful only if bit 2 in byte 1 = 1)
           absolute base address
```

The code is placed either starting at the given absolute address or, if no absolute address is given, at the position indicated by the actual value of the RAM/ROM pointer. After start-up of the linker all code is placed in the ROM section until the first "Define base address" record is encountered.

8.1.5 Data record

```
byte  0    type code (8)
remaining bytes are transferred directly to the executable
           program file
```

8.1.6 Define data area record

Defines a data area of the size specified within the program. The contents of the data area at start-up time are undefined.

byte 0 type code (9)
byte 1- 2 number of bytes in data area

8.1.7 Relative address record

Defines a 2-byte field in the executable program module into which the absolute address corresponding to the relative address should be placed.

byte 0 type code (10)
byte 1- 3 relative address

8.1.8 External address record

Defines a 2-byte field in the executable program module into which the sum of the value of the external symbol and the binary value is placed.

byte 0 type code (11)
byte 1- 2 reference number of external symbol (see description in "External symbol record" section)
byte 3- 4 binary value which should be added to the value of the external symbol

8.1.9 External byte record

Defines a 1-byte field in the executable program module into which the sum of the value of the external symbol and the binary value is placed.

byte 0 type code (12)
byte 1- 2 reference number of external symbol (see description in "External symbol record" section)
byte 3- 4 binary value which should be added to the value of the external symbol

If the binary value falls outside the intervals 0 <= value <= 255, and FF80 <= value <= FFFF the linker will output an error message and terminate.

8.1.10 End record

Defines the end of the relocatable file. Records following the end record will not be processed by the linker. If the linker encounters an end-of-file condition while reading the relocatable file it will output an error message and terminate.

byte 0 type code (15)

8.2 Absolute program file format

An absolute program file is of type 0 - 9. The MIKADOS system will not load executable programs from files having extents.

The file structure is:

first sector, byte 0 - 31: normal file information, as described in section 5.
byte 64 - 65: absolute address of program entry point (symbolic offset HLIND)
byte 66 - 67: program starting address (HLADR)
byte 68 - 69: address of first byte after program area (HLADR+2)

Sector 2 and the following sectors contain the absolute program starting in byte 0 of sector 2. The absolute program is not interrrupted by MIKADOS system information.

Appendix A. Summary of manual changes

The following is a summary of the changes that have occurred in this manual:

21 Sept 1979 original version