OS-9 – modułowy, wielozadaniowy system czasu rzeczywistego

- elastyczna, modułowa architektura
- 100% romowalność, praca bezdyskowa
- wielozadaniowość i wielodostępność
- podział czasu z wywłaszczaniem
- funkcje czasu rzeczywistego
- we/wy niezależne od sprzętu
- odporny na awarie system plików
- zgodność z UNIX-em na poziomie C
- dostępność języków wyższego rzędu
- narzędzia do uruchamiania programów na różnych poziomach (*User, System, Source*)

Funkcje systemowe

Nadzorowanie procesów *process management*: utworzenie (*create*), zaniechanie (*abort*),zakończenie (*terminate*), pobranie/ustawienie atrybutów (*get/set attributes*), przydział/zwalnianie pamięci (*allocate/free memory*), czekanie czasowe (*wait for time*), czekanie na zdarzenie (*wait for event*), sygnalizacja zdarzenia (*signal event*);

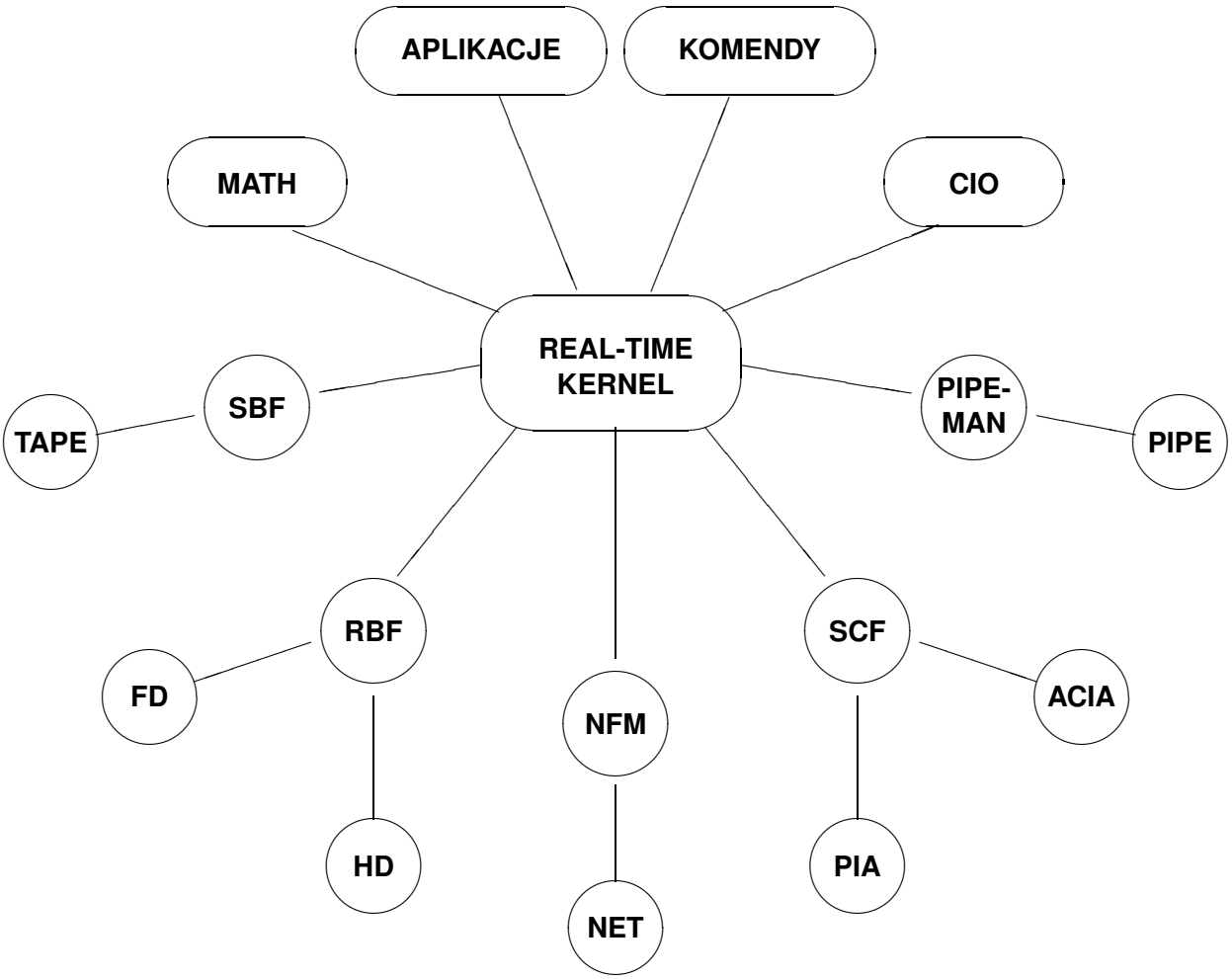
Operacje na plikach *file management*: utworzenie (*create*), usunięcie (*delete*), otwarcie (*open*), zamknięcie (*close*), czytanie (*read*), pisanie (*write*), zmiana położenia (*reposition*), pobranie/ustawienie atrybutów (*get/set attributes*);

Operacje na urządzeniach *device handling*: zamówienie (*request*), zwolnienie (*release*), czytanie (*read*), pisanie (*write*), zmiana położenia (*reposition*), pobranie/ustawienie atrybutów (*get/set attributes*);

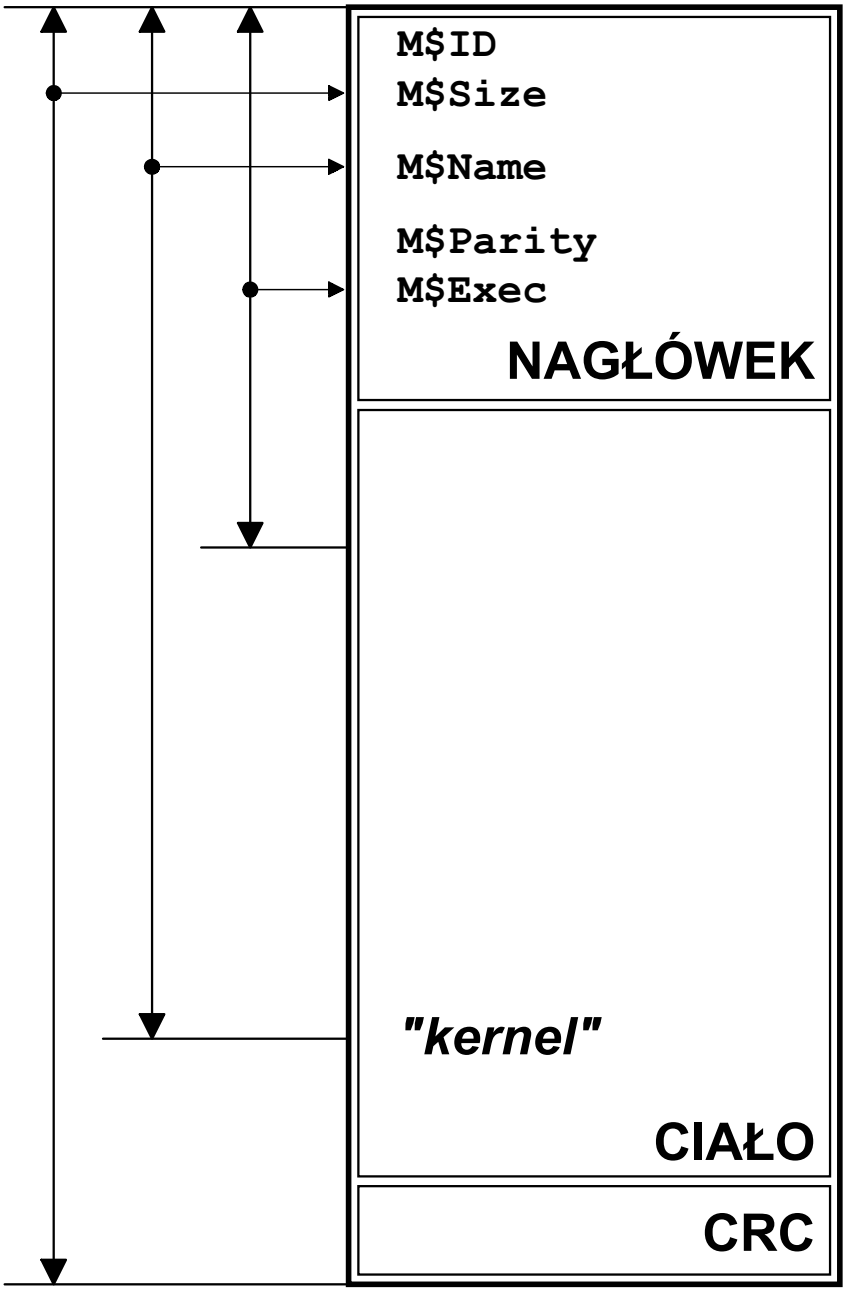
Utrzymywanie informacji *data maintenance*: pobranie/ustawienie daty/czasu (*get/set date/time*), pobranie/ustawienie danych systemowych (*get/set system data*), pobranie/ustawienie parametrów procesu/pliku/urządzenia (*get/set process/file/device attributes*),

Komunikacja *communication*: utworzenie/usunięcie połączenia (*create/delete connection*), nadawanie/odbieranie komunikatów (*send/receive messages*), przekazywanie informacji o stanie (*transfer status information*), przyłączanie/odłączanie urządzeń zdalnych (*attach/detach remote devices*).

Modułowa budowa systemu OS–9



Budowa modułu



Wspólna część nagłówka modułu

adr.	C *module.h*	asm *module.a*	znaczenie
00	_msync	M$ID	Znacznik modułu (4AFC)
02	_msysrev	M$SysRev	Numer wersji systemu
04	_msize	M$Size	Wielkość modułu
08	_mowner	M$Owner	Właściciel modułu
0C	_mname	M$Name	Wskaźnik nazwy modułu
10	_maccess	M$Accs	Zezwolenia na dostęp
12	_mtylan	M$Type,M$Lang	Typ i język
14	_mattrev	M$Attr,M$Revs	Atrybuty i wersja modułu
16	_medit	M$Edit	Numer edycji modułu
18	_musage	M$Usage	Wskaźnik komentarzy
1C	_msymbol	M$Symbol	Wskaźnik tablicy symboli
20	_mident		Kod identyf. modułu
22	_mspare		Zarezerwowane
2E	_mparity	M$Parity	Suma kontrolna nagłówka

Wybrane pola nagłówka

M$Accs:

```
-- -- -- -- -- pe pw pr -- ge gw gr -- ce cw cr
```

p – ***public***, **g** – ***group***, **c** – ***creator***,
w – ***write*** – pozwala na zapis do modułu,
r – ***read*** – pozwala ładować, przyłączeć i odłączać moduł,
e – ***execute*** – pozwala uruchamiać moduł.

M$Type:

1 - Prgm - moduł programowy
2 - Sbrtn - moduł podprogramów
4 - Data - moduł danych
11 - TrapLib - biblioteka
12 - Systm - składnik systemu
13 - Flmgr - moduł zarządzania plikami
14 - Drivr - moduł sterownika urządzeń
15 - Desc - moduł deskryptora urądzeń

M$Lang:

1 - **kod maszynowy**
2 - **kod pośredni Basic**
3 - kod pośredni Pascal
4 - kod pośredni C

M$Attr:

```
sharable sticky supervisor -- -- -- -- --
```

sharable – zezwala na równoczesne używanie modułu przez wiele procesów,
sticky – moduł jest usuwany z kartoteki przy wartości MD$Link=$-1$, a nie 0,
supervisor – moduł pracuje w trybie uprzywilejowanym.

M$Revs:

Pozwala na dynamiczne zastępowanie istniejących modułów ich nowszymi wersjami (nawet w ROM). Jeśli ładowany moduł ma wyższy numer wersji od istniejącego w kartotece, to funkcja **F$Load** podstawia w kartotece nowy moduł w miejsce starego.

Zmienna część nagłówka modułu

Typ modułu	adr.	C	asm	znaczenie
Flmgr, Systm	30	_mexec	M$Exec	Wskaźnik startu
	34	_mexcpt	M$Excpt	Wskaźnik obsługi TRAP
Drivr	38	_mdata	M$Mem	Wielkość obszaru danych
Prgm	3C	_mstack	M$Stack	Wielkość obszaru stosu
	40	_midata	M$IData	Wskaznik inicjalizacji danych
	44	_midref	M$IRefs	Wskaznik inicjalizacji wskaznikow
TrapLib	48	_minit	M$Init	Wskaźnik inicjalizacji TRAP
	4C	_mterm	M$Term	Wskaźnik zakończenia TRAP

Systemowy katalog modułów (*Module Directory*)

adr.	asm	znaczenie
00	MD$MPtr	Adres modułu w pamięci
04	MD$Group	Identyfikator grupy modułów (adres pierwszego modułu grupy)
08	MD$Static	Wielkość pamięci zajętej przez grupę modułów
0C	MD$Link	Licznik użytkowników modułu
0E	MD$MChk	Suma kontrolna nagłówka modułu

F$Link - zwraca adres modułu o podanej nazwie, zwiększa MD$Link;

F$Load - ładuje moduł z pliku o podanej ścieżce, wykonuje F$Link;

F$UnLink - zmniejsza MD$Link dla modułu o podanym adresie, usuwa moduł i zwalnia pamięć przy MD$Link=0;

F$UnLoad - zmniejsza MD$Link dla modułu o podanej nazwie, usuwa moduł i zwalnia pamięć przy MD$Link=0.

Funkcje biblioteki C dla modułów

Funkcja C	Opis
crc()	obliczenie CRC dla modułu
_get_module_dir()	pobranie elementu kartoteki modułów
make_module()	utworzenie modułu
_mkdata_module()	utworzenie modułu danych (typu *Data*)
modcload()	załadowanie modułu do pamięci "kolorowanej" (*colored memory*)
modlink()	dowiązanie do modułu o zadanej nazwie i typie
modload()	załadowanie modułu do pamięci i dowiązanie
modloadp()	załadowanie modułu do pamięci z użyciem zmiennej *PATH* i dowiązanie
munlink()	usunięcie dowiązania do modułu o zadanym adresie
munload()	usunięcie dowiązania do modułu o zadanej nazwie

Przykład dostępu do modułu danych

```
/*
   zalozenie:
        w module typu Data o nazwie "my_module"
        sa umieszczone dane w postaci struktury
        o typie my_data;
        wskaznik dptr ma zostac ustawiony na ich
        poczatek
*/
#include <module.h>

mh_com *mhptr;  /* wskaznik struktury naglowka */
my_data *dptr;  /* wskaznik struktury danych */

/* szukanie modulu */
mhptr = modlink("my_module", 0);
/* wyjscie z bledem */
if(mhptr == -1) return (errno);
/* znalezienie wskaznika do danych */
dptr = (my_data *)((char *)mhptr + mnptr->_mexec);
```

Komendy systemowe dla modułów

mdir

```
Syntax: mdir [<opts>] [<mod names>] [<opts>]
Function: display module directory
Options:
     -a         print language instead of type
     -e         print extended directory listing
     -t=<type> list modules only of type <type>
     -u         print unformatted listing
```

load

```
Syntax: load [<opts>] {<module> [<opts>]}
Function: load a module into memory
Options:
     -d         load file from data directory
     -l         print pathlist of file loaded
     -z         get list of file names from std. input
     -z=<path> get list of file names from <path>
```

link

```
Syntax: link [<opts>] {<modname> [<opts>]}
Function: link a module in memory
Options:
     -z         get list of module names from std. input.
     -z=<path> get list of module names from <path>
```

unlink

```
Syntax: unlink [<opts>] {<modname> [<opts>]}
Function: unlink modules from memory
Options:
     -z         get list of module names from std. input
     -z=<path> get list of module names from <path>
```

ident

```
Syntax: ident [<opts>] {<modname> [<opts>]}
Function: display module information
Options:
   -m           ident module in memory
   -q           quick mode, only one line per module
   -s           silent mode: quick, only disp. bad crcs
   -x           ident module in execution directory
   -z           get list of module names from std. input
   -z=<file>    get list of module names from <file>
```

Ładowanie i start systemu OS–9

W chwili startu systemu OS–9 (od punktu wejścia M$Exec modułu *kernel*) w pamięci muszą się znajdować:

- moduły systemowe ***kernel*** i ***init***,
- program uruchamiany jako pierwszy proces (zazwyczaj ***sysgo***),
- moduł sterownika zegara czasu rzeczywistego,
- moduły tworzące podsystemy we–wy dla konsoli systemowej i dysku systemowego (jeśli są przewidziane w konfiguracji).

UWAGA: Załadowanie tych modułów do pamięci, znalezienie modułu ***kernel***, zainicjalizowanie rejestrów procesora i skok do procedury startowej jądra musi wykonać program spoza systemu, tzw. ***bootstrap loader***, najczęściej rezydujący w pamięci stałej komputera (ROM) i uruchamiany po sprzętowym restarcie (*RESET*).

Budowa i funkcje programu *bootstrap*

plik źródłowy	procedura	wykonywane zadania
sysinit.a	`SysInit`	początkowa inicjalizacja sprzętu
boot.a		budowa tablic skoków do procedur obsługi wyjątków
boot.a		określenie MPUType (typu procesora)
sysinit.a	`SInitTwo`	dokończenie inicjalizacji sprzętu – odblokowanie przerwań
boot.a		poszukiwanie pamięci i budowa listy pamięci RAM i ROM
sysboot.a	`SysBoot`	ładowanie systemu (dołączenie obszaru do listy ROM) i poszukiwanie modułu *kernel*
boot.a		ustawienie rejestrów i skok do miejsca startu *kernel*-a (M$Exec)

UWAGA: Plik ***boot.a*** jest niezależny od sprzętu (dostarczany przez MICROWARE), a pliki ***sysinit.a, sysboot.a*** – zależne (tworzone przez autora implementacji).

Struktura pamięci przygotowanej przed startem jądra systemu (*kernel*)

Obszar zmiennych w pamięci RAM o wielkości 8 kB:

0 – 2539 – tablica skoków do procedur obsługi wyjątków (2540 bajtów),

2540 – 4095 – obszar na stos roboczy *bootloader*-a (wskazywany przez rejestr wskaźnika stosu **a7**),

4096-8191 – obszar na globalne zmienne systemowe (D_), wskazywany przez **a6**.

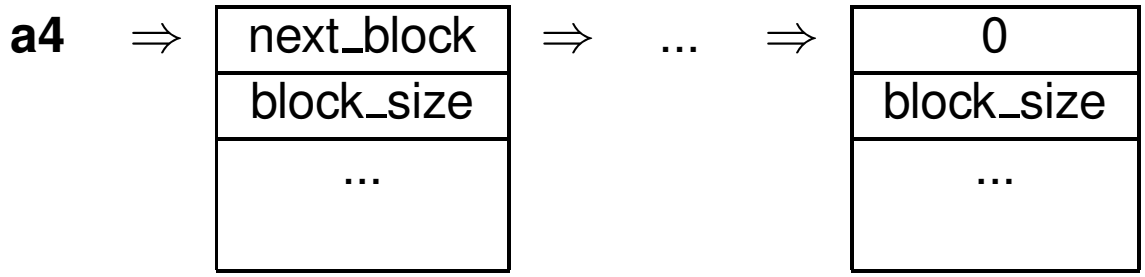
UWAGA: Tablica skoków zawiera 254 elementy stanowiące indywidualne punkty wejścia procedur obsługi błędów:

```
PEA    #N*4       adres wektora na stos
JMP    BadExcpt   skok do obslugi w boot.a
```

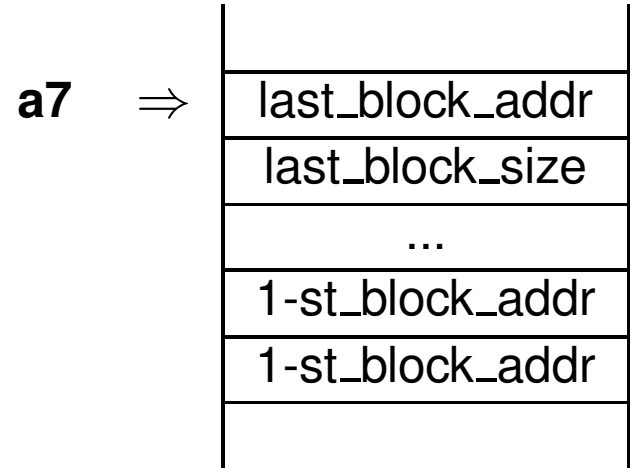
W sytuacji wyjątkowej na stosie składowany jest adres wektora i nastepuje skok do procedury obslugi (początkowo wszystkie prowadzą do procedury `BadExcpt` z pliku ***boot.a***).

Lista pamieci RAM i ROM przygotowana przez *bootstrap* dla *kernel*

Znalezione bloki pamięci RAM są łączone w listę wskazywaną przez a4:



Znalezione bloki pamięci ROM są zapamiętywane na stosie (a7):



Przygotowanie rejestrów do uruchomienia *kernel*-a

d0.l = ilość wolnej pamięci RAM
d1.b = typ procesora (MPUType)
d2.b = flaga debugger-a (0=brak)
d3.l = typ restartu 0-cold (1-quick)
d4.l = 0
d5.l = 0
d6.l = 0
d7.l = 0

(a0) = adres startu znalezionego modułu *kernel*
(a1) = adres startu *bootstrap* (przy błędzie fatalnym)
(a2) = 0
(a3) = 0
(a4) = wskaźnik listy wolnej pamięci RAM
(a5) = adres tablicy skoków dla wyjątków
(a6) = adres obszaru zmiennych globalnych(D_)
(a7) = mapa pamięci ROM

Procedura startowa jądra systemu (*kernel*)

1. Poszukiwanie modułów w pamięci ROM (z listy na stosie), sprawdzanie ich poprawności (CRC) i dodawanie ich do systemowej kartoteki modułów.
2. Dołączenie (F$Link) do modułu konfiguracyjnego ***init*** (jego brak jest błędem fatalnym).
3. Zainicjalizowanie wszystkich tablic i struktur systemowych, otwarcie domyślnych ścieżek we–wy systemowych (zazwyczaj ***/term***) i domyślnych katalogów na dysku systemowym (zazwyczaj ***/dd***).
4. Uruchomienie (F$Fork) pierwszego procesu (zazwyczaj ***sysgo***).
5. Uruchomienie procesu systemowego (kod w ***kernel***, ID=1), niewidocznego dla użytkownika, obsługującego funkcje czasu rzeczywistego (przerwania od RTC).
6. Koniec procedury startowej.

Moduł konfiguracyjny *init*

```
typedef struct {
  struct modhcom _mh;    /* common header info */
  long   _mmaxmem;       /* top limit of free ram */
  ushort _mpollsz,       /* number of IRQ polling
                         /* tbl entries */
         _mdevcnt,       /* number of device table
                         /* entries */
         _mprocs,        /* number of process table
                         /* entries */
         _mpaths,        /* number of path table
                         /* entries */
         _msysparam,     /* offset to parameter
                            string for _msysgo */
         _msysgo,        /* offset to initial module
                            name */
         _msysdrive,     /* offs. to sys. dev. name */
         _mconsol,       /* offset to console name */
         _mextens,       /* offset to customization
                            module name list */
         _mclock,        /* offset to clock module
                            name */
```

```
        _mslice,        /* number of clock ticks
                           per time slice */
        _mip_id;        /* interprocessor identif. */
  long  _msite;         /* installation site code */
  ushort _minstal;      /* installation name offs. */
  long  _mcputyp;       /* cpu class (000/.../070) */
  char  _mos9lvl[4];    /* system lvl/ver./edition */
  ushort _mos9rev,      /* offs. to lvl/rev string */
        _msyspri,       /* initial system priority */
        _mminpty,       /* initial min. executable
                           priority */
        _maxage;        /* initial maximum natural
                           process age */
  long  _mmdirsz;       /* nbr of moddir entries */
  ushort _mevents;      /* number of system event
                           table entries */
  char  _mcompat,       /* ver. change byte #1 */
        _mcompat2;      /* ver. change byte #2*/
  ushort _mmemlist,     /* offs. to col. mem. list */
        _stacksz,       /* IRQ stack size (lwords) */
        _mcoldretrys,   /* coldstart chd retry ctr */
        _mreserved[10];/* reserved space */
}mod_config;
```

Typowy program startowy *sysgo*

```
Entry
    lea     Intercpt(pc),a0
    os9     F$Icpt
En0
* WELCOME message
    move.w  #1,d0           std IO path
    move.l  #Msg1Siz,d1
    lea     Msg1Str(pc),a0
    os9     I$WritLn

* F$STime
    move.l  #$00080000,d0   00hhmmss
    move.l  #$07c30000,d1   yyyymmdd rtc --> mm = 0:
    os9     F$STime         set from chip
    bcc.s   En1
    move    #1,d0
    move.l  #TimESiz,d1
    lea     TimEStr(pc),a0
    os9     I$WritLn
```

```
En4
* shell -p=" sh> "
    moveq   #0,d0           any type module
    moveq   #0,d1           default memory size
    moveq   #ShellPSiz,d2   parmameter size
    moveq   #3,d3           copy std I/O paths
    move.w  #Priority,d4    medium priority
    lea     ShellStr(pc),a0 process to create
    lea     ShellPar(pc),a1 parameter string
    os9     F$Fork          fork to new primary mod.
    bcc.s   En41
    move    #1,d0
    move.l  #ShellESiz,d1
    lea     ShellEStr(pc),a0
    os9     I$WritLn
    bra     En1
En41
    os9     F$Wait          wait for process to die
    bra     En0
```

```
*************************************************
Msg1Str   dc.b " sysgo: WELCOME to OS-9 V2.4",C$CR
Msg1Siz   equ  *-Msg1Str

TimEStr   dc.b " sysgo: can't set sys clock",C$CR
TimESiz   equ  *-TimEStr

ChxEStr   dc.b " sysgo: can't change exec dir to "
ChxStr    dc.b "/dd/CMDS",C$CR
ChxESiz   equ  *-ChxEStr

ChdEStr   dc.b " sysgo: can't change work dir to "
ChdStr    dc.b "/dd",C$CR
ChdESiz   equ  *-ChdEStr

ShellEStr dc.b " sysgo: ERROR - can't fork to   "
ShellStr  dc.b " shell",0," "
ShellPar  dc.b -p= ",$22,śh> ",$22,C$CR
ShellPSiz equ  *-ShellPar
ShellESiz equ  *-ShellEStr
```