miniSO

Características

- código-fonte reduzido (*assembly* e C)
- facilidade de expansão
- para arquitetura Intel x86 em modo real
- uso do BIOS
- suporte a processos leves (threads)
- compilação com ferramentas MS-DOS
- setor de inicialização em Bootlib
- imagem com interpretador de comandos e comandos internos embutidos

Estrutura e código-fonte

- Setor de partida → boot.bl
- Imagem
  - init.c
  - comand.*
  - lib.*
  - scall.*
  - miniSO.asm, miniSO.h, miniSO.def
  - Makefile, miniSO.bxrc

SETOR DE PARTIDA

```
boot(size=512,signature=0xaa55,segment=0x07c0)
{
  fat(oem="miniSO",volume="BOOTDISK",disk=FD1440);
  setstack(0x0000,0x7c00);
  setcolor(0x07);
  putstr("Carregando minisSistema Operacional...");
  readdisk_chs(0x00,0,1,16,3,0x07e0,0x0000);
  readdisk_chs(0x00,1,0,1,18,0x0840,0x0000);
  readdisk_chs(0x00,1,1,1,18,0x0a80,0x0000);
  readdisk_chs(0x00,2,0,1,18,0x0cc0,0x0000);
  jump(0x07e0,0x0000);
}
```

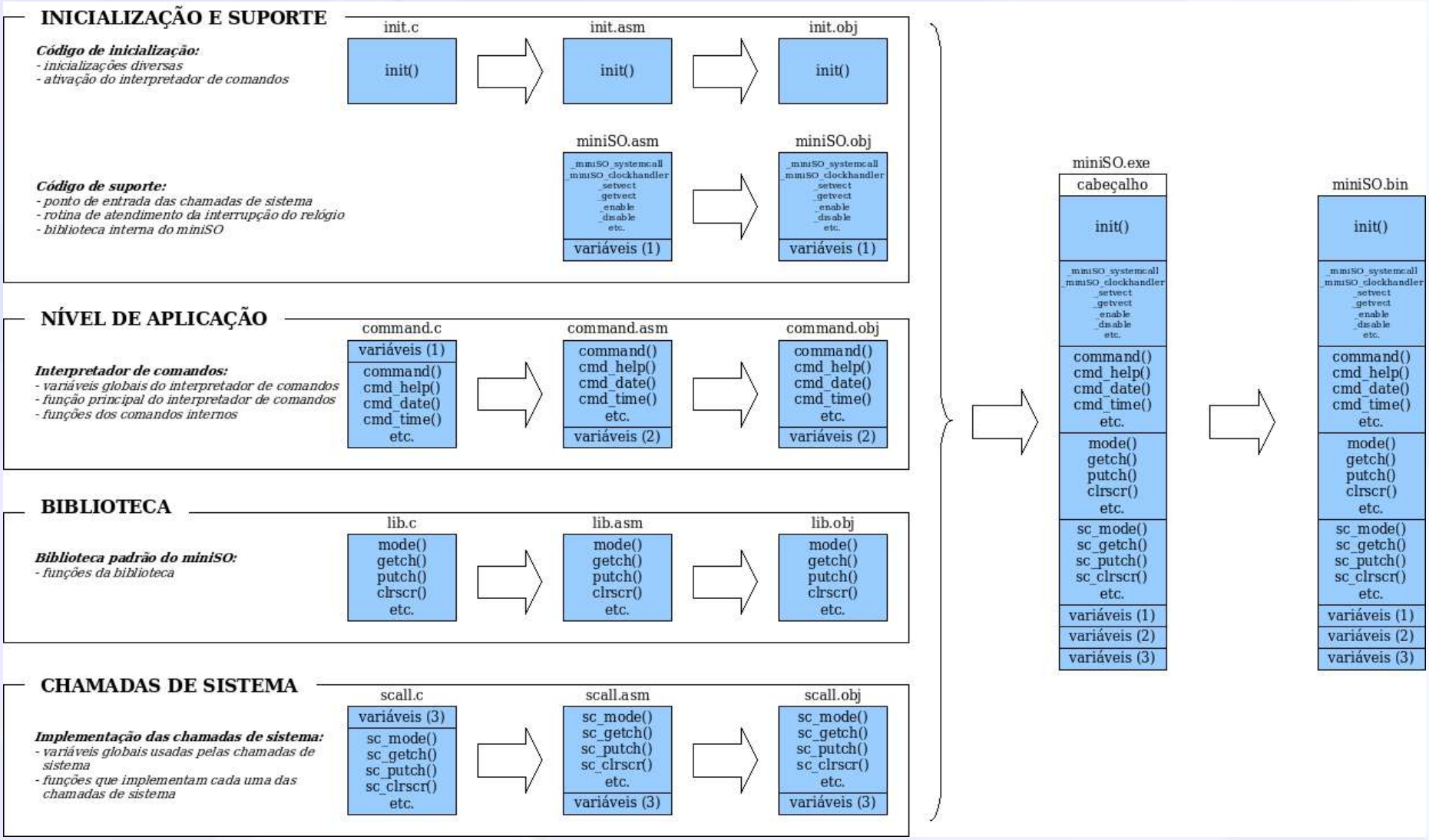
SETOR DE PARTIDA (COMPILAÇÃO)

```
bootlib -o boot.asm boot.bl
tasm boot
tlink boot
exe2bin boot boot.bin
bootcopy
```

IMAGEM

- código de inicialização (init.c), incluindo rotinas internas e a rotina de atendimento da interrupção do relógio (miniSO.asm)
- aplicações, incluindo o interpretador de comandos (command.c)
- biblioteca (lib.c)
- chamadas de sistema (scall.c)
- operações especiais com “inteiros” (miniSO.asm): N_LXMUL@, LXMUL@, etc.

COMPILAÇÃO DA IMAGEM



INICIALIZAÇÃO DO REG. DS

- cada .obj tem segmentos de código e dados
- na formação do .exe todos os segmentos de código são agrupados, da mesma forma que todos os segmentos de dados
- CS deverá apontar para o código
- DS deverá apontar para os dados
- SS deverá apontar para a pilha

INICIALIZAÇÃO DO REG. DS

- CS: apontará automaticamente para a área da memória onde a imagem foi colocada (0x07e00) pois o boot executa um JMP para esta área
- SS: é definido na inicialização (por escolha do desenvolvedor)
- DS: como inicializar DS?
  - DS = área de código+tamanho do código
  - DS=0x07e00+tamanho do código

INICIALIZAÇÃO DO REG. DS

- questões:
  - qual o tamanho do código?
  - isto está no cabeçalho do executável
  - e também no no arquivo MINISO.MAP
- problema:
  - na inicialização, DS tem que ser inicializado
  - mas como saber qual o tamanho do código sem compilar o miniSO

INICIALIZAÇÃO DO REG. DS

- solução:
  - compilar o miniSO
  - descobrir o tamanho do código em MINISO.MAP
  - definir o tamanho do código em miniSO.def
  - recompilar o miniSO e
  - executá-lo

MINISO.MAP

```
Start  Stop   Length Name               Class
 00000H 0278DH 0278EH _TEXT             CODE
 0278EH 03333H 00BA6H _DATA             DATA
 03334H 036D0H 0039DH _BSS              BSS


Detailed map of segments

 0000:0000 00B8 C=CODE   S=_TEXT          G=(none)  M=INIT.C     ACBP=28
 0000:00B8 0EED C=CODE   S=_TEXT          G=(none)  M=COMMAND.C  ACBP=28
 0000:0FA5 07AD C=CODE   S=_TEXT          G=(none)  M=LIB.C      ACBP=28
 0000:1752 01AF C=CODE   S=_TEXT          G=(none)  M=MINISO.ASM ACBP=28
 0000:1901 0E8D C=CODE   S=_TEXT          G=(none)  M=SCALL.C    ACBP=28
 0278:000E 004E C=DATA   S=_DATA          G=DGROUP  M=INIT.C     ACBP=48
 0278:005C 09C2 C=DATA   S=_DATA          G=DGROUP  M=COMMAND.C  ACBP=48
 ...
```

miniSO.def

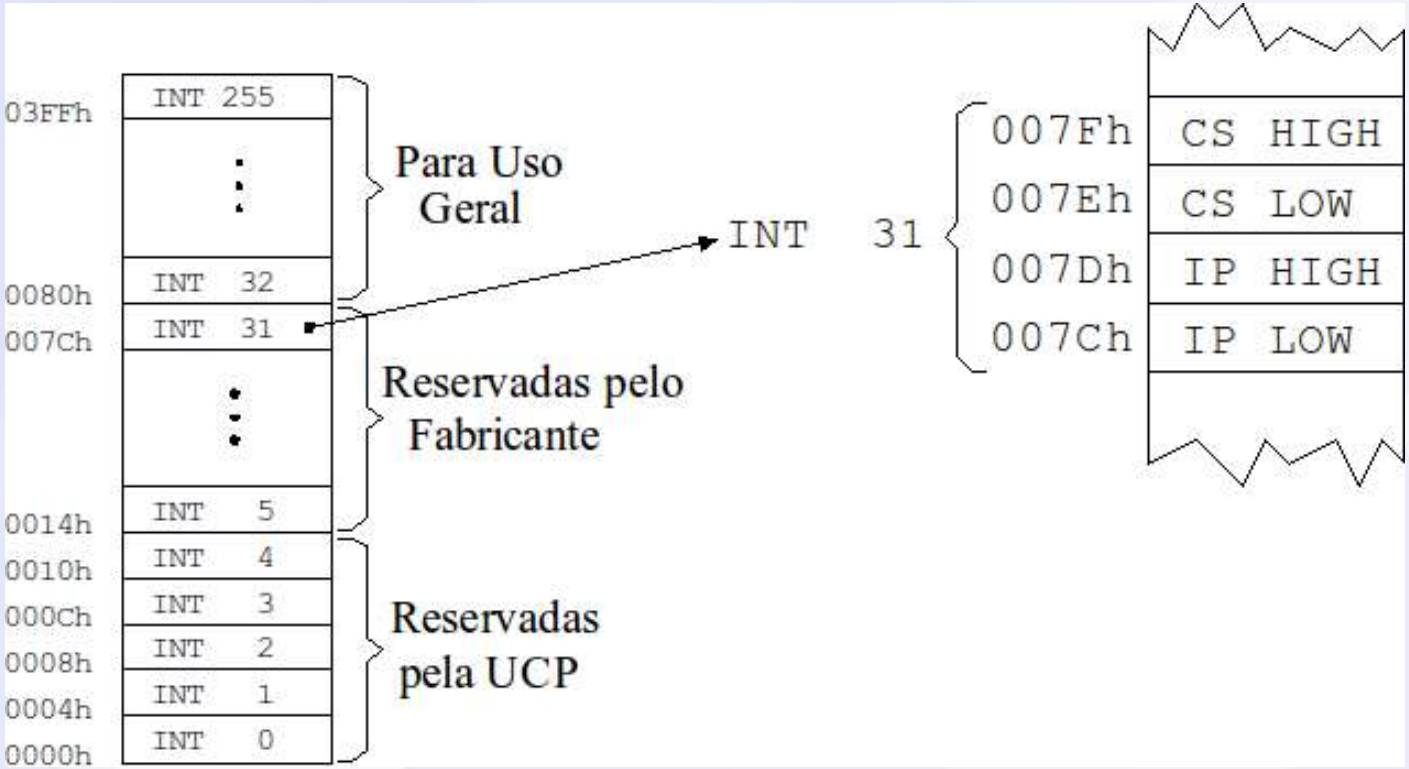
```
# ----- Variaveis --------------------------------
# Deslocamento ate o inicio do segmento de dados (hexadecimal)
DATA    = 0278
# Segmento de codigo onde a imagem do miniSO sera colocada (hexadecimal)
CODE    = 07e0
# Versao do miniSO
VERSION = 2007
```

SUPORTE À EXEC. CONC.

- envolve entendimento do mecanismo de interrupções
- desvio da interrupção do relógio
- implementação do chaveamento de contexto

CONTROLE DE INTERRUP.

- 256 vetores de interrupção, cada um com 4 bytes (0x00000 até 0x003ff)



CONTROLE DE INTERRUP.

- Ao detectar uma interrupção x, a UCP:
  - termina a instrução atual e avança IP para a próxima instrução
  - coloca os flags na pilha (SS:SP)
  - desativa os bits de estado IF e TF
  - coloca CS na pilha
  - coloca IP na pilha
  - executa jump p/ conteúdo de x * 4 (=pos. mem. end. rot. at. int.)

CONTROLE DE INTERRUP.

- IRET encerra a rotina de atendimento de interrupções e:
  - retira IP da pilha
  - retira CS da pilha
  - retira flags da pilha

CRIAÇÃO DE UMA ROT. AT. INT.

- modificador interrupt:

```
void interrupt clockhandler()
{
[...]
}
```

CRIAÇÃO DE UMA ROT. AT. INT.

```
; salva registradores
push ax
push bx
push cx
push dx
push es
push ds
push si
push di
push bp

mov  bp,offset DADOS_ROTINA
mov  ds,bp

mov  bp,sp

sub  sp,TAM_DADOS

; Código da função: ...

mov  sp,bp              ; (*) Só
aparece quando há variáveis
locais
; recupera o valor dos
registradores
pop  bp
pop  di
pop  si
pop  ds
pop  es
pop  dx
pop  cx
pop  bx
pop  ax
iret
```

CRIAÇÃO DE UMA ROT. AT. INT.

```
/* Protótipo da função getvect() */
void interrupt (*getvect(int interruptno))();

/* Declaração de uma variável para receber o
endereço de uma rotina de atendimento de
interrupções */
void interrupt (*ender)();

/* Obtém o endereço da rotina de atendimento de
interrupções 0x08 */
ender = getvect (0x08);
```

CRIAÇÃO DE UMA ROT. AT. INT.

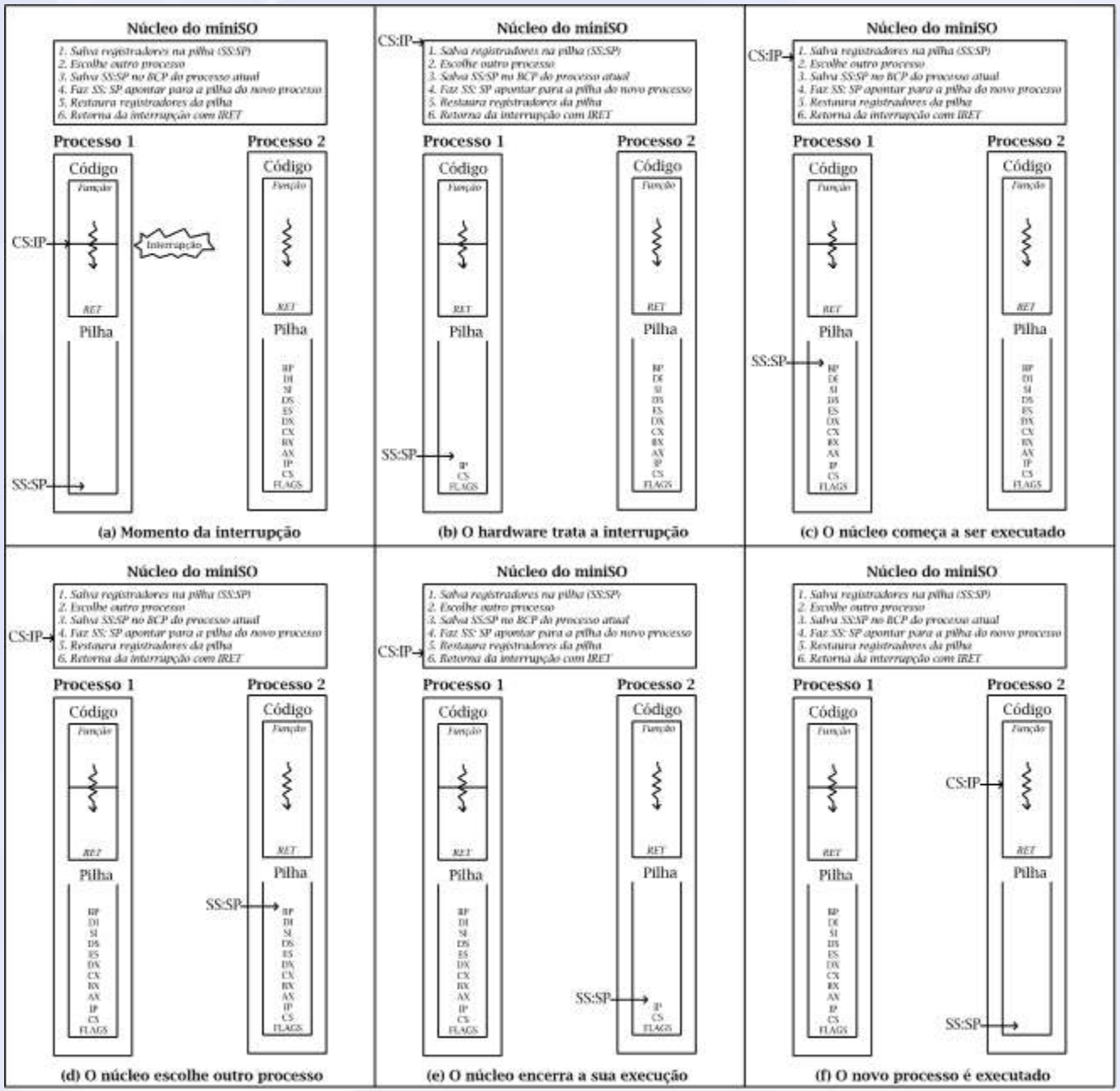
```
/* Protótipo da função setvect() */
void setvect(int intno, void interrupt (*isr)());

/* Código da nova rot. at. da interrupção 0x21 */
void interrupt nova_int_21h()
{
[...]
}

[...]
/* Define end. at. interrupção 0x21 */
setvect(0x21,nova_int_21h);
```

COMPART. DE TEMPO

- consiste em dividir o tempo entre os processos em execução
- é obtido desviando-se a int. do relógio (8)
- a int. do relógio ocorre por padrão 18,2x/s
- passos:
  - obter end. da rot. de at. do rel. original
  - criar uma função de at. nova que fará o chaveamento de contexto
  - desviar a int. 8 para a nova rot. (c/ setvect)

CHAVEAMENTO DE CONTEXTO

- durante a execução da rot. de at., todos os registradores estão na pilha
- para passar a exec. de um processo para outro, basta fazer o ponteiro de pilha apontar para a pilha de outro processo
- ao sair da rot. de at., os registradores do outro processo serão desempilhados e a execução continua no outro processo
- para criar um novo processo, é preciso preencher a sua pilha


Núcleo do miniSO
1. Salva registradores na pilha (SS:SP)
2. Escolhe outro processo
3. Salva SS:SP no BCP do processo atual
4. Faz SS: SP apontar para a pilha do novo processo
5. Restaura registradores da pilha
6. Retorna da interrupção com IRET
Processo 1
Processo 2
Código
Função
RET
Pilha
CS:IP
SS:SP
Interrupção
BP
DI
SI
DS
ES
DX
CX
BX
AX
IP
CS
FLAGS
(a) Momento da interrupção
(b) O hardware trata a interrupção
(c) O núcleo começa a ser executado
(d) O núcleo escolhe outro processo
(e) O núcleo encerra a sua execução
(f) O novo processo é executado

ESTRUTURAS DE DADOS

- tipos simples:

```
typedef int      pid_t;
typedef int      pcb_t;
typedef unsigned signal_t;
typedef int semid_t;
```

ESTRUTURAS DE DADOS

- BCP/PCB e tabela de processos:

```
typedef struct {
  pid_t    pid;
  pid_t    ppid;
  int      status;
  unsigned ss;
  unsigned sp;
  signal_t recvsig;
  signal_t waitsig;
  pcb_t    wait;
  pid_t    waitfor;
  int      waitres;
  pcb_t    zombies;
  pcb_t    prev;
  pcb_t    next;
} miniSO_PCB;
```

```
miniSO_PCB
 miniSO_thread[miniSO_MAXTHREADS];
```

ESTRUTURAS DE DADOS

- Chamadas para gerência de processos:
  - startthread()
  - killthread()
  - waitthread()
  - exitthread()
  - getpid()
  - getppid()
  - sendsignal()
  - waitsignal()

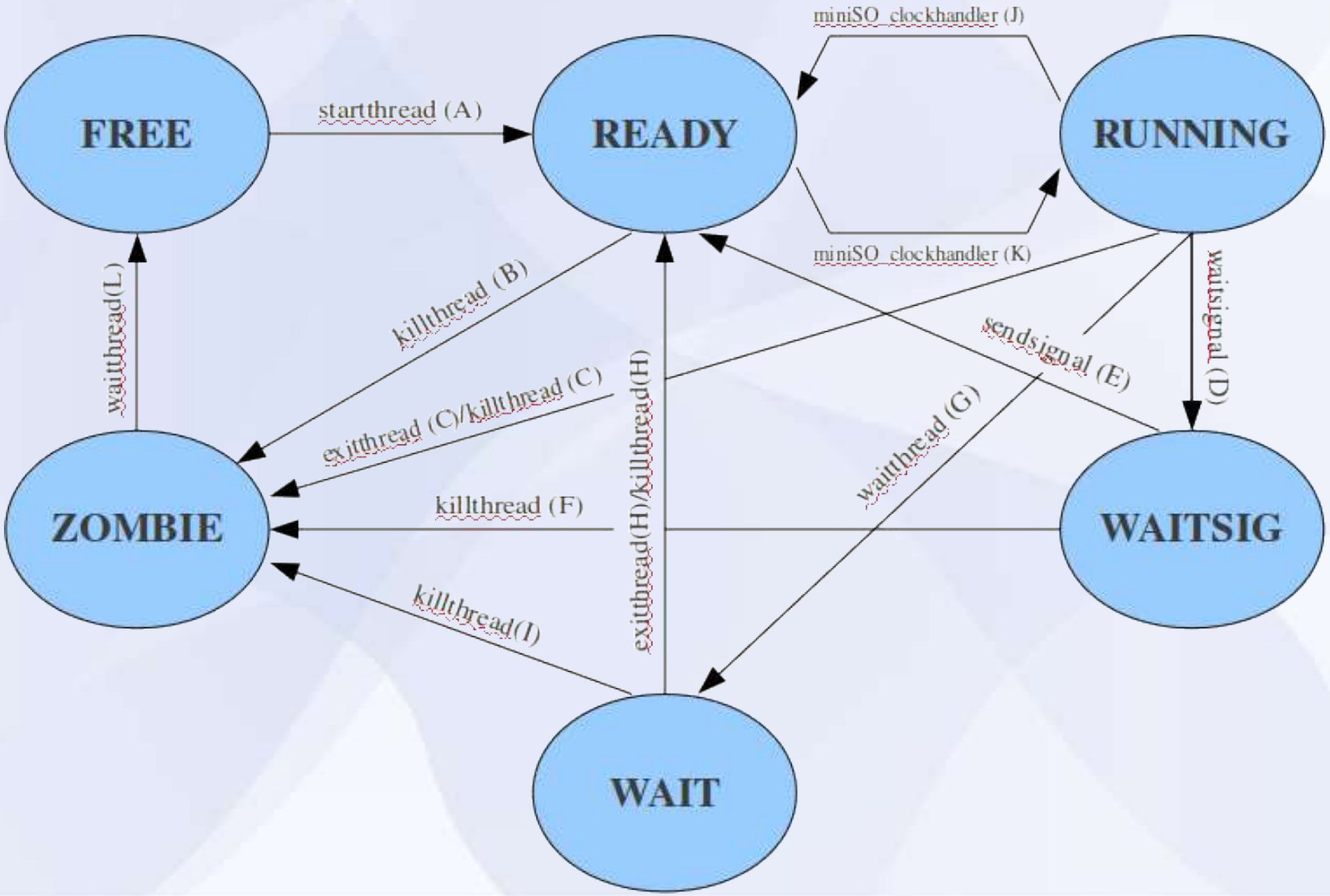
ESTRUTURAS DE DADOS

- Estados de processos
  - FREE
  - READY
  - RUNNING
  - WAITSIG
  - WAIT
  - ZOMBIE
  - WAITSEM

ESTRUTURAS DE DADOS

- Listas e campos de controle de estado
  - lista de BCPs livres (miniSO_free)
  - lista de prontos (miniSO_ready)
  - lista de espera por sinais (miniSO_waitsig)
  - lista de processos-filho em estado zumbi (campo zombies do BCP do processo-pai)
  - campo wait do BCP de cada processo contém pai que está esperando pelo final da execução de um processo-filho

DIAG. DE TRANS. DE EST.



ESTRUTURAS DE DADOS

- Semáforos

```
typedef struct {
  int   status;
  semid_t semid;
  int   value;
  pcb_t queue;
} miniSO_SEM;
```

```
miniSO_SEM
miniSO_sem[miniSO_MAXSEMAPHORES];
```

ESTRUTURAS DE DADOS

- Chamadas para semáforos:
  - semid_t sc_semcreate(int value);
  - int sc_semset (semid_t s,int value);
  - int sc_semup (semid_t s);
  - int sc_semdown (semid_t s);
  - int sc_semdestroy (semid_t s);