Implementace multithreadingu

- Následující text je pouze stručný nástin toho, jak se přepínají procesy
  - Neřeší např. vícenásobné přerušení, které také může nastat
- Cílová platforma jsou x86 kompatibilní procesory
- Funkce nejsou zapsány optimálně
  - V reálném projektu je toho třeba
  - Viz přednáška o urychlení běhu vlákna
  - Je třeba vzít do úvahy další komponenty systému – např. správce paměti

MS-DOS Compatible, Uniprocessor

- Semestrálka KIV/ZOS 1998
- Princip
  - Pomocí služby DOSu se nainstaluje nový handler přerušení 8, které generují hodiny
  - Když je volána obsluha přerušení, v zásobníku jsou uloženy registry CS, IP a Flags kódu, jehož vykonávání bylo přerušeno
  - Obsluha přerušení uloží stávající registry
  - Plánovač vybere novou úlohu a zapíše do zásobníku její hodnoty uvedených registrů (CS, IP a Flags)
  - Obsluha přerušení obnoví zbývající registry procesoru pro plánovačem vybranou úlohu
  - Provede se instrukce iret, kterou se spustí naplánovaný proces díky přepsání hodnot v zásobníku

```
//User space
procedure Task1; far;
begin
  //dělá něco ohromně užitečného
end;

//Kernel space

type
  TTask = record
    FPU:array[0..FPUSize-1] of byte
    ALU:array[0..ALUSize-1] of byte;
    Stack:pointer;
    State:integer;
  end;

var tasks:array[0..MaxTasks-1]
          of TTask;
    oldHandler:procedure;
    currentTask:0..MaxTasks-1;

procedure SaveRegisters; assembler;
asm
  ;instrukcemi mov a fsave uloží
  ;registry do tasks[currentTask]
  ;flags se uloží přes zásobník a ax

  fsave es:[si]         //FPU
  mov es:[si+114], ax   //ALU
  mov es:[si+116], bx
```

```
    ...
    pop ax        ;záloha CS, IP a Flags
    pop bx
    pop cx

    mov es:[si+130], ax

    ...
end;

procedure RestoreRegisters;assembler;
asm
    //analogicky k SaveRegisters
end;

procedure Scheduler;
begin
    //do proměnné currentTask vybere
    //index nové úlohy ke spuštění
    //tj. naplánuje ji
end;
```

```
procedure ClockHandler; assembler;
asm
  pushf
  call oldHandler

  cli

  call SaveRegisters
  call Scheduler
  call RestoreRegisters

  sti

  iret
end;


procedure Init;
begin
  InitTasks;
  InstallTask(@Task1);
  GetIntVec($8, @oldHandler);
  SetIntVec($8, @ClockHandler);
end;
```

```
procedure InstallTask(task:pointer);
begin
 with Tasks[FindFreeSlot] do
 begin
   State:=stCreating;

   Move(ALU, DefaultALU, sizeofALU);
   Move(FPU, DefaultFPU, sizeofFPU);

   //CS:IP
   ALU[130]:=Seg(Task^);
   ALU[132]:=Ofs(Task^);


   //SS:SP
   GetMem(Stack, StackSize);
   ALU[130]:=Seg(Stack^);
   ALU[132]:=Ofs(Stack^)+StackSize;

   State:=stRunnable;
 end;
end;
```

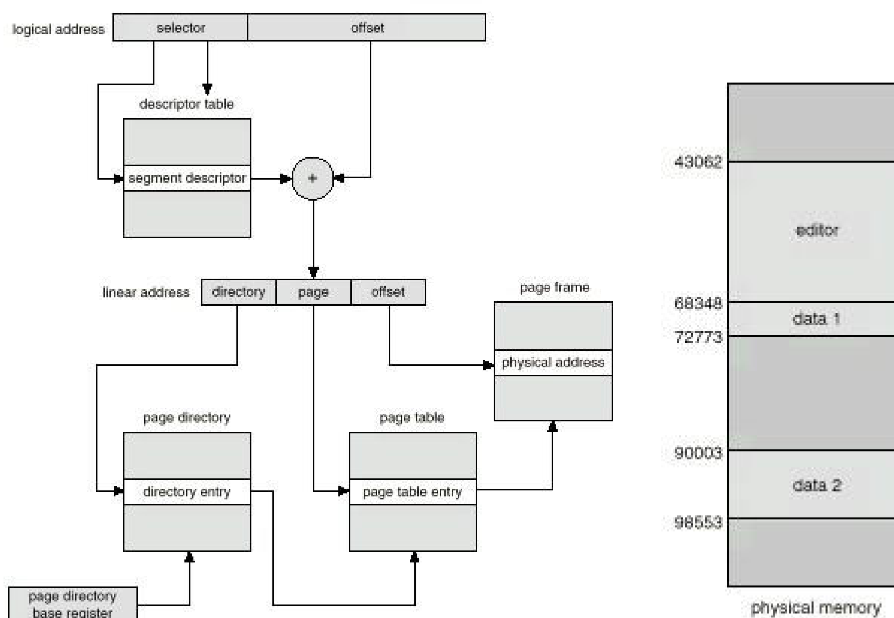
Protected Mode

- Umožnil x86 odstínit user-space od
- kernel-space na hw úrovni
  - Úrovně oprávnění
    - CPL – code privilege level
    - DPL – data privilege level

  - Deskriptor privilege level
    - 00_2 – největší oprávnění – kernel space
    - 11_2 - nejmenší oprávnění – user space
      - WNT používá jenom 2 kombinace kvůli kompatibilitě s Alphou, která měla jen 1b

  - Některé instrukce lze vykonat pouze s CPL=0
  - Také říká, komu je paměť přístupná podle PL

- (Virtuální) Paměť počítače
  - RAM + odkládací soubor
  - a dělí se na bloky zvané segmenty (a ty na stránky)



http://data.uta.edu/~ramesh/cse3320/chap8.html

- Každý spuštěný program vlastní několik segmentů
  - V jednom má uložený programový kód
    - Tj. popis, co má program dělat
  - V ostatních data

- Každý segment má svůj popisovač
  - Segment deskriptor
  - Segmentové registry (cs, ds, es, fs, gs, ss) pak obsahují segment selektor

- Popisovač segmentu má výše uvedené bity, které určují privilege level

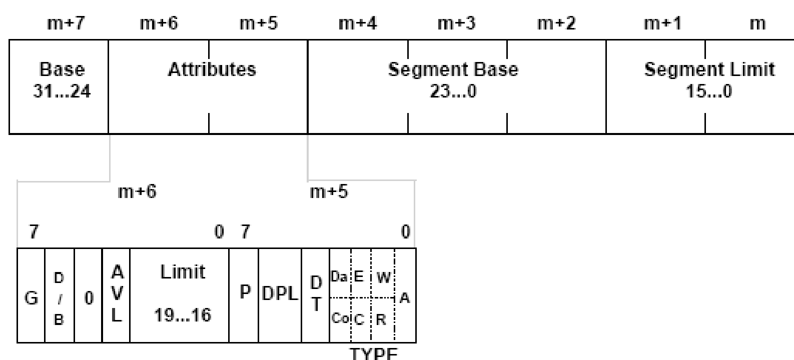


Figure 2. Segment Descriptor Layout

Bit Name	Bit Meaning
G	Granularity bit, used to determine if the limit is check on byte or 4KB page granularity.
D/B	Default/Big bit, for code segments represent the default operand size (16 or 32 bit). For expand down data segments it affects the operation of limit checking (this maintains compatibility with 286 protected mode expand down segments).
AVL	Available bit, this bit is available for use by the system designer.
P	Present bit, indicates that the specified segment is present in memory. Can be used to help with the implementation of virtual memory system.
DPL (2-bits)	Descriptor Privilege Level, Used by the protection mechanism.
DT	Descriptor Type, for systems descriptors DT=1 for segment descriptors DT=0.
Code/Data	Segment Type bit, indicates if the segment is a Code (=1) or Data (=0) segment. The setting of this bit effects the meaning of bits 1 and 2
E (Data)	If E=1 the data segment is an expand down data segment (typically used for stacks).
W (Data)	If W=1 the data segment is both read and write, else segment is read-only.
C (Code)	If C=1 then the code segment is a conforming code segment
R (Code)	If R=1 then the code segment is executable and readable, else it is execute-only.
A	Access bit, The processor automatically sets this bit whenever a descriptor is referenced. The bit is cleared by software.

Table 1. Segment Descriptor Attributes

ftp://download.intel.com/design/intarch/papers/exc_ia.pdf

Protected-Mode, Uniprocessor

- Příklad pro MS-DOS běžel v reálném režimu, málo paměti
  - => přepnutí procesoru do chráněného režimu umožní adresovat paměť nad 1 MB
    - Ovšem také znemožní některé věci, které byly možné v reálném režimu
    - A „přijdeme o paměť/amnézie“, protože se ze segmentových registrů stanou registry ne se segmenty, ale se segment selektory
  - => přepnutí umožňuje hw odstínění user-mode a kernel-mode přístupu k paměti
    - § x86-64 používá long mode

- V principu to funguje stejně jako v reálném režimu, ale je třeba se postarat o více věcí
  - Instrukcí cli zakázat přerušení
  - Připravit si novou tabulku přerušení
  - Přepnout procesor do chráněného režimu
  - Instrukcí lgdt nahrát global descriptor table
  - Instrukcí lidt „aktivovat“ novou tabulku přerušení
  - Nastavit registry SS:ESP na nový zásobník
  - „Vrátit se“ přes instrukce ljmp a ret do chráněného režimu

```
EnablePM:

  cli            ;zamaskování přerušení

  mov eax, cr0
  bts eax, 0        ;přepneme do
  mov cr0, eax      ;chráněného režimu

  lgdt [newGDT]     ;nastavíme selektory
  lidt [newIDT]     ;přerušení a vyjímky

  mov eax, selSS
  mov ss, eax       ;zásobník

  ljmp selCS, @pm ;nastav segment
                    ;selektor kódu:EIP
pm:
   ret              ;a vrátíme se v pm
```

- Obsluha přerušení musí pochopitelně počítat s tím, že je v chráněném režimu
- V chráněném režimu je dostupný TSS – Task State Segment
    - Usnadňuje implementaci multithreadingu

Vlastní multithreading pomocí fcí Win32

- Jen na ukázku, jak by se řešilo výše uvedené
  - Prakticky to zřejmě ani nemá využití
    - Možná nějaký velmi nestandardní důvod

- Princip
  - Jedno vlákno dělá hodiny – tj. interrupt 8
    - Tj. volá SwitchTask
  - Ve druhém vláknu se postupně spouštějí úlohy

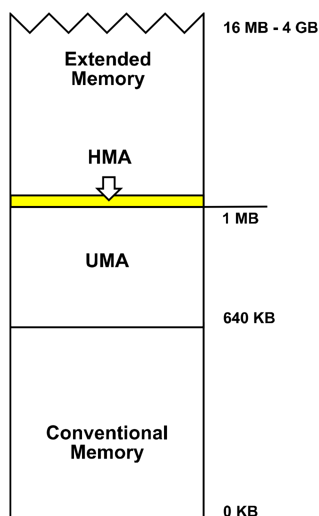
```
//user-space

procedure MyTask;
begin
  //dělá něco úžasného
end;

//kernel-space

type
  TTask = record
    Context:TContext;
    State:integer;
  end;

var FOriginalContext:TContext;
    tasks:array[0..MaxTasks-1]
                          of TTask;
    currentTask:0..MaxTasks-1;
```

```
procedure Init;
begin
  FOriginalContext.ContextFlags:=
     CONTEXT_FULL or
     CONTEXT_FLOATING_POINT or
     CONTEXT_DEBUG_REGISTERS;
  GetThreadContext(Handle,
          FOriginalContext);
end;


procedure InstallTask(task:pointer);
begin
  with Tasks[FindFreeSlot] do
  begin
   State:=stCreating;

   Context:=FOriginalContext;
   Context.EIP:=task;
   Context.ESP:=AllocateStack;

   State:=stReady;
  end;
end;
```

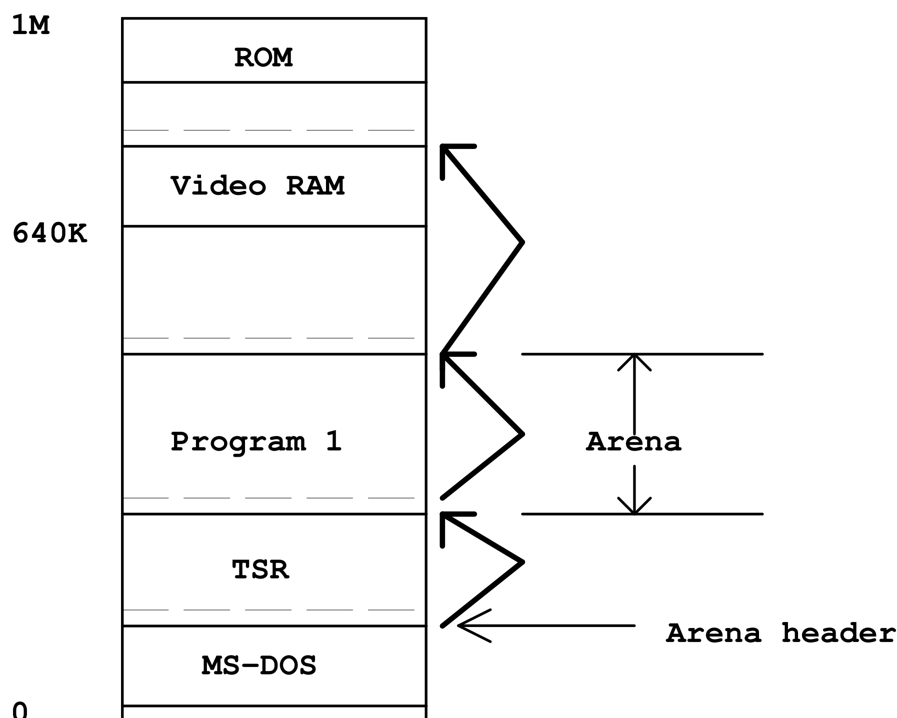
```
procedure SwitchTask;
begin
  SuspendThread(FTaskingThread);

  GetThreadContext(FTaskingThread,
         Tasks[currentTask].Context);
  Tasks[currentTask].State:=stReady;

  Scheduler;
  Tasks[currentTask].State:=stRunning
  SetThreadContext(FTaskingThread,
         Tasks[currentTask].Context);

  ResumeThread(FTaskingThread);
end;
```

Paměť

- Procesy se v reálném režimu nedostanou nad 1MB



http://en.wikipedia.org/wiki/Extended_memory



http://www.ceng.metu.edu.tr/courses/ceng334/MSDOS.doc

- Konvenční paměť aka Base Memory
  - Dolních 640kB
  - „640K ought to be enough for anybody." - Bill Gates, 1981

- UMB alias UMA
  - Upper Memory Blocks/Area
  - Mezi 640kB a 1MB
  - Vyhrazeno pro
    - ROM
    - RAM periferií
    - Do paměti mapované I/O

- High Memory Area
  - 64kB-16B nad 1MB
  - Muselo se povolit A20 (21. adresní bit na sběrnici)

- Aréna
    - V reálném režimu se paměť spravuje v 16B blocích – paragrafech
    - Paměťovému bloku přidělenému procesu se říká aréna
    - Aréna se sestává z několika paragrafů a začíná hlavičkou (1. paragraf), kde je uveden počet paragrafů arény
        - MS-DOS v hlavičce ukládal PSP (Program Segment Prefix) a jméno souboru

- EBDA
    - Extended Bios Data Area
    - Přidáno kvůli PS/2
        - Nový buffer pro PS/2 port myši
    - Ve skutečnosti není standardizováno, pouze existuje proprietární popis např. pro původní IBM BIOS EBDA
    - Použito pro Plug-and-Play

- Unreal mode
    - i386+
    - Procesor se přepne do chráněného režimu
        - Zároveň povolí A20
    - Nastaví se limity segmentů na maximum
        - Respektive se naplní s deskriptorem datového segmentu s limitem na 4GB
    - Procesor se přepne zpět do reálného režimu
    - Nastavené limity zůstanou aktivní
    - A program v reálném režimu může adresovat 4GB flat

MP Floating Pointer Structure

- Datová struktura u SMP systémů
- Začíná signaturou _MP_
- Popisuje dostupné procesory
- Pokud ji kód jádra OS nenajde, předpokládá se uniprocessor
  - Tj. podle její detekce se rozhoduje, jaké jádro se vlastně nainstaluje – viz protected mode
  - Hledá signaturu na určených místech v paměti
    - 1. kB EBDA
    - Poslední kb základní paměti
    - Arény v adresním rozsahu ROM-BIOSu

MP Floating Pointer Structure			
Field	**Offset**	**Length**	**Description/Use**
Signature	0	4B	This 4 byte signature is the ASCII string "_MP_" which the OS should use to find this structure.
MPConfig Pointer	4	4B	This is a 4 byte pointer to the MP configuration structure which contains information about the multiprocessor configuration.
Length	8	1B	This is a 1 byte value specifying the length of this structure in 16 byte paragraphs. This should be 1.
Version	9	1B	This is a 1 byte value specifying the version of the multiprocessing specification. Either 1 denoting version 1.1, or 4 denoting version 1.4.

Checksum	10	1B	The sum of all bytes in this floating pointer structure including this checksum byte should be zero.
MP Features 1	11	1B	This is a byte containing feature flags.
MP Features 2	12	1B	This is a byte containing feature flags. Bit 7 reflects the presence of the ICMR, which is used in configuring the IO APIC.
MP Features 3-5	13	3B	Reserved for future use.

Processor Entry			
Field	**Offset**	**Length**	**Description/Use**
Entry Type	0	1B	Since this is a processor entry, this field is set to 0.
Local APIC ID	1	1B	This is the unique APIC ID number for the processor.
Local APIC Version	2	1B	This is bits 0-7 of the Local APIC version number register.
CPU Enabled Bit	3:0	1b	This bit indicates whether the processor is enabled. If this bit is zero, the OS should not attempt to initialize this processor.
CPU Bootstrap Processor Bit	3:1	1b	This bit indicates that the processor entry refers to the bootstrap processor if set.

CPU Signature	4	4B	This is the CPU signature as would be returned by the CPUID instruction. If the processor does not support the CPUID instruction, the BIOS fills this value according to the values in the specification.
CPU Feature flags	8	4B	This is the feature flags as would be returned by the CPUID instruction. If the processor does not support the CPUID instruction, the BIOS fills this value according to values in the specification.

- Dále existují
    - MP Configuration Table
    - MP Configuration Table Entries
    - IO APIC Entry
    - APIC Memory Mappings
    - Local APIC Register Addresses

- Viz http://www.osdever.net/tutorials/view/multiprocessing-support-for-hobby-oses-explained
- A manuály od Intelu

SMP Bootstrap

- Pojmy
  - BSP – Bootstrap Procesor
    - Vykonává programový kód po spuštění počítače
  - AP – Application Procesor
    - Každé další jádro SMP systému
    - APID – id procesoru
    - Aka Auxiliary Processor
- Princip
  - Po zapnutí jsou všechny procesory v reálném režimu
  - BIOS vybere BSP a ostatní procesory zastaví
  - Kód SMP jádra běžící na BSP prohledá paměť na _MP_
  - Pokud nenašel, zavede se jednoprocesorové jádro
  - Pokud našel, inicializuje APIC BSP
    - K tomu je nutné se přepnout do chráněného režimu
  - Kód vykonávaný BSP postupně vzbudí AP pomocí Init-IPI (Inter-Processor Interrupt)
  - AP se přepne do chráněného režimu a začne svoji další činnost synchronizovat s kódem, který ho spustil a běží na BSP
  - Jakmile jsou inicializovány všechny AP, BSP přepne I/O APIC do symetrického IO režimu
    - Routovací tabulka, která přesměruje přerušení od sběrnic periferií na některý lokální APIC AP
  - SMP jádro pokračuje dál s vlastní inicializací

```
broadcast_AP_startup:
#
# This procedure is executed only by the BootStrap
# Processor, to awaken the Auxilliary Processors so
# that they each can display their Local-APIC ID-
# number (and their CR0 register's value, so we can
# verify that the cache-related bits are setup
# properly). We use code here which follows the MP
# Initialization Protocol.
#

# point FS:EBX to the Local-APIC's memory-mapped page

xor %ax, %ax          # address segment zero
mov %ax, %fs          #   with FS register
mov $APIC_BASE, %ebx # APIC address in EBX

# compute the page-number(where each AP should start)

mov $realCS, %edx    # arena segment-address
shl $4, %edx         # multiplied by sixteen
add $tos, %edx       #  plus entry's offset
shr $12, %edx        # divided by page-size
and $0xFF, %edx      # must be in bottom 1MB

# issue an 'INIT' Inter-Processor Interrupt command

mov $0x000C4500, %eax    # broadcast INIT-IPI
mov %eax, %fs:0x300(%ebx)# to all-except-self

# do 10ms delay, enough time for APs to awaken

mov $10000, %eax     # ten-thousand microseconds
call delay_EAX_microseconds# execute programmed delay

# wait for indication of the command's completion

spin1:  bt  $12, %fs:0x300(%ebx) # command-in-
                                  # progress?
jc  spin1                     # yes, spin until done

#-----------------------------------------------------
```

```
# now we complete the Intel 'MP Initialization
# Protocol'
#-----------------------------------------------------

mov $2, %ecx     # protocol's repetitions nxIPI:

# issue a 'Startup' Inter-Processor Interrupt command
mov $0x000C4600, %eax    # issue 'Startup-IPI'
mov %dl, %al             # page is the vector
mov %eax, %fs:0x300(%ebx)# to all-except-self

# delay for 200 microseconds

mov $200, %eax                 # number of microseconds
call delay_EAX_microseconds    # for a programmed delay

# wait for indication of the command's completion

spin2:  bt  $12, %fs:0x300(%ebx) # command-in-
                                 # progress?
jc  spin2                      # yes, spin until done

# repeat this 'Statup-IPI' step twice (per the
# protocol)

loop nxIPI             # again for MP protocol

ret
#-----------------------------------------------------
```

```
#----------------------------------------------------------
initAP:
#
# This procedure will be executed by each Application
# Processor as it is awakened by the BootStrap
# Processor sending Startup-IPI's. In order that each
# processor can call subroutines, it requires a
# private stack-area, which we setup sequentially
# using the 'xadd' instruction (to guarantee that
# stack-areas are non-overlapping). But until its
# stack is ready, this CPU cannot handle interrupts.
#

cli                 # disable interrupts
mov %cs, %ax        # address program arena
mov %ax, %ds        #   using DS register
mov %ax, %es        #    and ES register

# increment the count of processors that have
# awakened

lock                # insure 'atomic' update
incwn_APs           # increment the AP count

# setup an exclusive stack-region for this processor

mov $0x1000, %ax        # paragraphs in segment
xadd%ax, newSS          # 'atomic' xchg-and-add
mov %ax, %ss            # segment-address to SS
xor %esp, %esp          # top-of-stack into ESP

# call subroutines to display this processor's
# Local-ID

callallow_4GB_addressing # adjust FS seg-limit
calldisplay_APIC_LocalID # display this CPU's ID

# increment the count of processors that have
# finished
lock                # insure 'atomic' update
incwn_fin               # when modifying counter
#----------------------------------------------------------
```

http://www.cs.usfca.edu/~cruse/cs630f08/mphello.s

Proposed boot sequence for an RTOS on SMP systems.

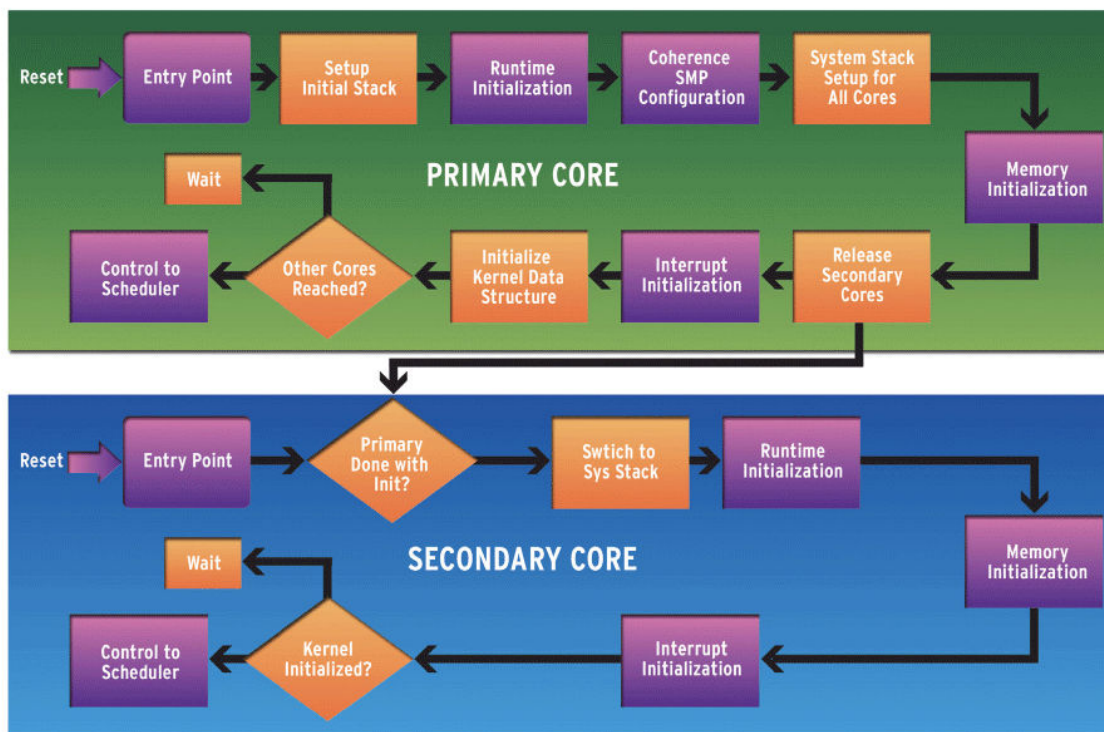


Figure 2

http://www.embeddedinternetdesign.com/design/227500241;jsessionid=EX4DHZLAFHP01QE1GHRSKHWATMY32JVN?pgno=2

Typical boot sequence operations in an embedded boot monitor.

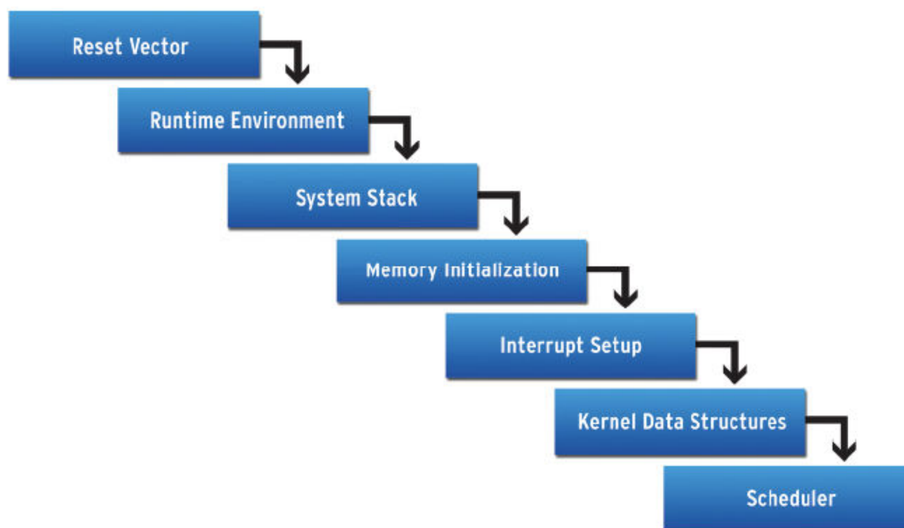


Figure 1

http://www.embeddedinternetdesign.com/design/227500241;jsessionid=QLTXJP4HWV3JTQE1GHRSKHWATMY32JVN?pgno=1

Režimy procesoru

- Real-mode
  - Kód může cokoliv a kamkoliv, kam může adresovat
  - Neexistuje žádná ochrana

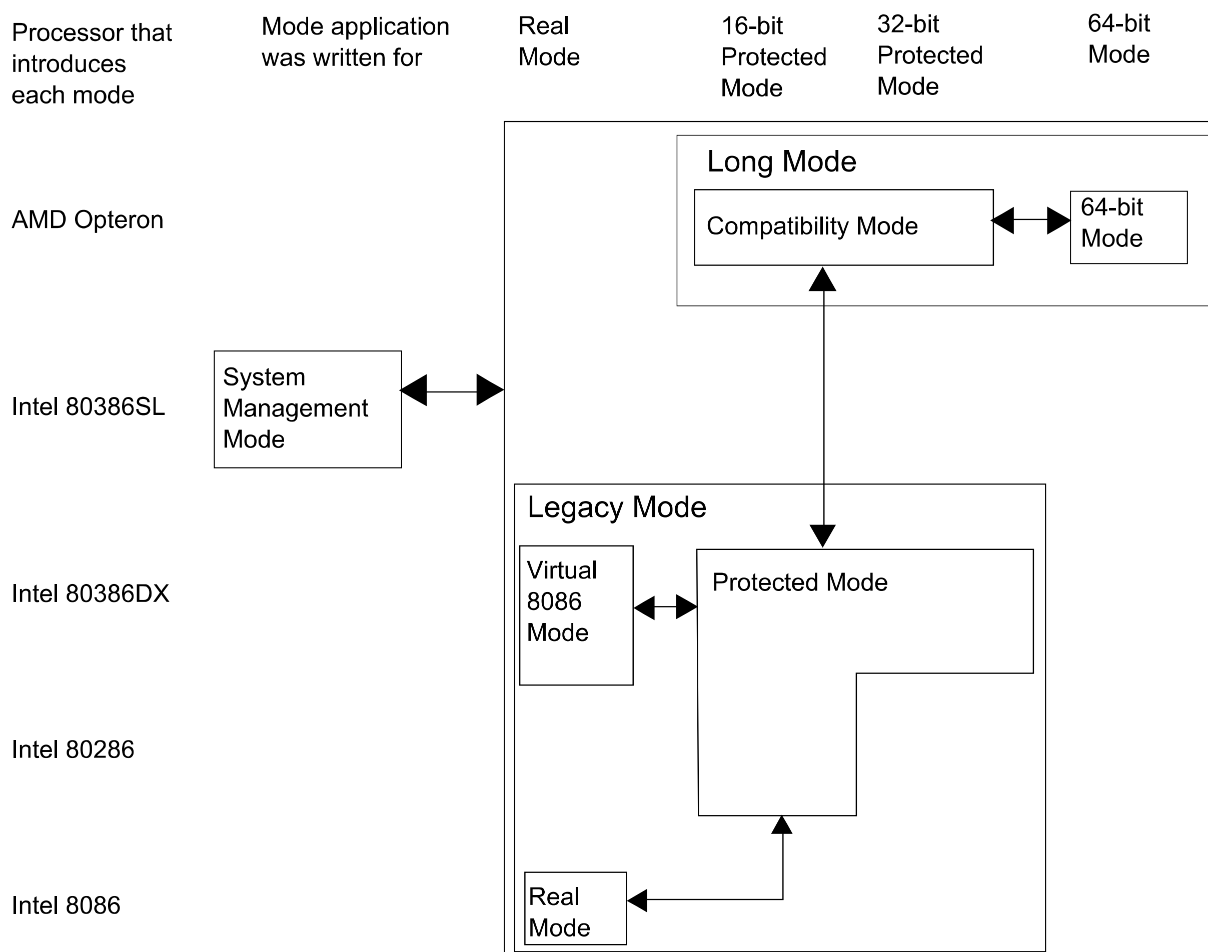
- Protected-mode
  - Viz výše
- Unreal mode
  - Neoficiální režim
  - Viz výše

- VM86
  - Virtual Mode
  - Zpřístupňuje real-mode kód v chráněném režimu
  - Např. volání služeb BIOSu z chráněného režimu

- SMM
  - System Management Mode
  - Žádný kód, jak ho obvykle známe, neběží
  - Pouze běží speciální kód, firmware nebo hw debugger, v privilegovaném režimu

- 64bit Compatibility Mode
  - Umožňuje 64bit OS spouštět 32bitové aplikace
  - Nejsou k dispozici 64bit registry

- 64bit Long Mode
  - Většinu 32bit aplikací prý stačí jen překompilovat
  - Viz dále



http://en.wikipedia.org/wiki/X86-64

- http://www.codeproject.com/KB/system/asm.aspx

Long Mode

- Některé systémové instrukce jsou v Long Mode a Kompatibility Mode nedostupné
  - Segmenty – viz protected mode
  - TSS – Task State Segment
    - Hw podpodora pro multithreading

- Adresování
  - Flat režim
  - Až na FS a GS, jako kdyby všechny segmentové registry byly rovny nule

  - Vytvoří se segment deskriptor s bity D=0 (default) a L=1 (long mode)
    - Jejich kombinace říká 64bitový segment
    - Hodnoty DS, SS a ES jsou ignorovány, ať už do nich dáte cokoliv

- Obsluha přerušení
  - Pozor, nové adresy obsluh jsou už 64bit

- Multi-threading
  - Stále ten samý princip jako u protected-mode
  - Což jsou privileges, APIC a prakticky ta samá virtualizace jako u real-modu
    - Která se dá použít i u unreal modu

- Přepnutí do long-mode
  - http://www.codeproject.com/KB/system/asm.aspx
  - http://www.ijack.org.uk/HTML/S/78.html

```
EnterLM:

; Disable paging
mov eax, cr0          ; Read CR0.
and eax,7FFFFFFFh     ; Set PE=0
mov cr0, eax          ; Write CR0.

; Set Page Address Etension
; by setting CR4's 5th bit.
mov eax, cr4
bts eax, 5
mov cr4, eax

; Create the new page tables and load CR3
; with them. Because CR3 is still 32-bits
; before entering long mode, the page
; table must reside in the lower 4GB.

call SetupMemory

; Enable long mode (note, this does not
; enter long mode, it just enables it)
; EFER = Extended Feature Enable Register

mov ecx, 0c0000080h ; EFER MSR number.
rdmsr                 ; Read EFER.
bts eax, 8            ; Set LME=1.
wrmsr                 ; Write EFER.

; Enable Paging to activate Long Mode.
; Assuming that CR3 is loaded with the
; physical address of the page table.
mov eax, cr0          ; Read CR0.
or eax,80000000h      ; Set PE=1.
```

```
mov cr0, eax        ; Write CR0.

; Now you are in compatibility mode.
; Enter 64-bit mode by jumping to an 64-
; bit code segment:

db 0eah
dd LinearAddressOfStart64
dw code64_idx

; The only thing you have to do in 64-bit
; mode is to reset the RSP:

mov rsp,stack64

; Hotovo, běžíme v long-mode
; i s nastaveným zásobníkem
```