/\*

\* cmpe283-1.c - Kernel module for CMPE283 assignment 1

\*/

#include <linux/module.h> /\* Needed by all modules \*/

#include <linux/kernel.h> /\* Needed for KERN\_INFO \*/

#include <asm/msr.h>

#define MAX\_MSG 80

/\*

\* Model specific registers (MSRs) by the module.

\* See SDM volume 4, section 2.1

\*/

#define IA32\_VMX\_PINBASED\_CTLS 0x481

#define IA32\_VMX\_BASIC 0x480

#define IA32\_VMX\_PROCBASED\_CTLS 0x482

#define IA32\_VMX\_PROCBASED\_CTLS2 0x48B

#define IA32\_VMX\_EXIT\_CTLS 0x483

#define IA32\_VMX\_ENTRY\_CTLS 0x484

#define IA32\_VMX\_TRUE\_PINBASED\_CTLS 0x48D

#define IA32\_VMX\_TRUE\_PROCBASED\_CTLS 0x48E

#define IA32\_VMX\_TRUE\_EXIT\_CTLS 0x48F

#define IA32\_TRUE\_ENTRY\_CTLS 0x490

/\*

\* struct caapability\_info

\*

\* Represents a single capability (bit number and description).

\* Used by report\_capability to output VMX capabilities.

\*/

struct capability\_info {

uint8\_t bit;

const char \*name;

};

/\*

\* Pinbased capabilities

\* See SDM volume 3, section 24.6.1

\*/

struct capability\_info pinbased[5] =

{

{ 0, "External Interrupt Exiting" },

{ 3, "NMI Exiting" },

{ 5, "Virtual NMIs" },

{ 6, "Activate VMX Preemption Timer" },

{ 7, "Process Posted Interrupts" }

};

struct capability\_info truepinbased[5] =

{

{ 0, "External Interrupt Exiting" },

{ 3, "NMI Exiting" },

{ 5, "Virtual NMIs" },

{ 6, "Activate VMX Preemption Timer" },

{ 7, "Process Posted Interrupts" }

};

struct capability\_info basic[5] =

{

{ 55, "True controls capability" }

};

struct capability\_info procbased[21] =

{

{ 2, "Interrupt-window exiting" },

{ 3, "Use TSC offsetting" },

{ 7, "HLT exiting" },

{ 9, "INVLPG exiting" },

{ 10, "MWAIT exiting" },

{ 11, "RDPMC exiting" },

{ 12, "RDTSC exiting" },

{ 15, "CR3-load exiting" },

{ 16, "CR3-store exiting" },

{ 19, "CR8-load exiting" },

{ 20, "CR8-store exiting" },

{ 21, "Use TPR shadow" },

{ 22, "NMI-window exiting" },

{ 23, "MOV-DR exiting" },

{ 24, "Unconditional I/O exiting" },

{ 25, "Use I/O bitmaps" },

{ 27, "Monitor trap flag" },

{ 28, "Use MSR bitmaps" },

{ 29, "MONITOR exiting" },

{ 30, "PAUSE exiting" },

{ 31, "Activate secondary controls" }

};

struct capability\_info trueprocbased[21] =

{

{ 2, "Interrupt-window exiting" },

{ 3, "Use TSC offsetting" },

{ 7, "HLT exiting" },

{ 9, "INVLPG exiting" },

{ 10, "MWAIT exiting" },

{ 11, "RDPMC exiting" },

{ 12, "RDTSC exiting" },

{ 15, "CR3-load exiting" },

{ 16, "CR3-store exiting" },

{ 19, "CR8-load exiting" },

{ 20, "CR8-store exiting" },

{ 21, "Use TPR shadow" },

{ 22, "NMI-window exiting" },

{ 23, "MOV-DR exiting" },

{ 24, "Unconditional I/O exiting" },

{ 25, "Use I/O bitmaps" },

{ 27, "Monitor trap flag" },

{ 28, "Use MSR bitmaps" },

{ 29, "MONITOR exiting" },

{ 30, "PAUSE exiting" },

{ 31, "Activate secondary controls" }

};

struct capability\_info procbased2[22] =

{

{ 0, "Virtualize APICaccesses" },

{ 1, "Enable EPT" },

{ 2, "Descriptor-table exiting" },

{ 3, "Enable RDTSCP" },

{ 4, "Virtualize x2APIC mode" },

{ 5, "Enable VPID" },

{ 6, "WBINVD exiting" },

{ 7, "Unrestricted guest" },

{ 8, "APIC-register virtualization" },

{ 9, "Virtual-interrupt delivery" },

{ 10, "PAUSE-loop exiting" },

{ 11, "RDRAND exiting" },

{ 12, "Enable INVPCID" },

{ 13, "Enable VM functions" },

{ 14, "VMCS shadowing" },

{ 15, "Enable ENCLS exiting" },

{ 16, "RDSEED exiting" },

{ 17, "Enable PML" },

{ 18, "EPT-violation #VE" },

{ 19, "Conceal VMX from PT" },

{ 20, "Enable XSAVES/XRSTORS" },

{ 22, "Mode-based execute control for EPT" },

{ 25, "Use TSC scaling" }

};

struct capability\_info exitcontrol[11] =

{

{ 2, "Save debug controls" },

{ 9, "Host addressspace size" },

{ 12, "Load IA32\_PERF\_GLOB AL\_CTRL" },

{ 15, "Acknowledge interrupt on exit" },

{ 18, "Save IA32\_PAT" },

{ 19, "Load IA32\_PAT" },

{ 20, "Save IA32\_EFER" },

{ 21, "Load IA32\_EFER" },

{ 22, "Save VMXpreemption timer value" },

{ 23, "Clear IA32\_BNDCFGS" },

{ 24, "Conceal VMX from PT" }

};

struct capability\_info trueexitcontrol[11] =

{

{ 2, "Save debug controls" },

{ 9, "Host addressspace size" },

{ 12, "Load IA32\_PERF\_GLOB AL\_CTRL" },

{ 15, "Acknowledge interrupt on exit" },

{ 18, "Save IA32\_PAT" },

{ 19, "Load IA32\_PAT" },

{ 20, "Save IA32\_EFER" },

{ 21, "Load IA32\_EFER" },

{ 22, "Save VMXpreemption timer value" },

{ 23, "Clear IA32\_BNDCFGS" },

{ 24, "Conceal VMX from PT" }

};

struct capability\_info entrycontrol[9] =

{

{ 2, "Load debug controls" },

{ 9, "IA-32e mode guest" },

{ 10, "Entry to SMM" },

{ 11, "Deactivate dualmonitor treatment" },

{ 13, "Load IA32\_PERF\_GLOBA L\_CTRL" },

{ 14, "Load IA32\_PAT" },

{ 15, "Load IA32\_EFER" },

{ 16, "Load IA32\_BNDCFGS" },

{ 17, "Conceal VMX from PT" }

};

struct capability\_info trueentrycontrol[9] =

{

{ 2, "Load debug controls" },

{ 9, "IA-32e mode guest" },

{ 10, "Entry to SMM" },

{ 11, "Deactivate dualmonitor treatment" },

{ 13, "Load IA32\_PERF\_GLOBA L\_CTRL" },

{ 14, "Load IA32\_PAT" },

{ 15, "Load IA32\_EFER" },

{ 16, "Load IA32\_BNDCFGS" },

{ 17, "Conceal VMX from PT" }

};

/\*

\* report\_capability

\*

\* Reports capabilities present in 'cap' using the corresponding MSR values

\* provided in 'lo' and 'hi'.

\*

\* Parameters:

\* cap: capability\_info structure for this feature

\* len: number of entries in 'cap'

\* lo: low 32 bits of capability MSR value describing this feature

\* hi: high 32 bits of capability MSR value describing this feature

\*/

void

report\_capability(struct capability\_info \*cap, uint8\_t len, uint32\_t lo,

uint32\_t hi)

{

uint8\_t i;

struct capability\_info \*c;

char msg[MAX\_MSG];

memset(msg, 0, sizeof(msg));

for (i = 0; i < len; i++) {

c = &cap[i];

snprintf(msg, 79, " %s: Can set=%s, Can clear=%s\n",

c->name,

(hi & (1 << c->bit)) ? "Yes" : "No",

!(lo & (1 << c->bit)) ? "Yes" : "No");

printk(msg);

}

}

/\*

\* detect\_vmx\_features

\*

\* Detects and prints VMX capabilities of this host's CPU.

\*/

void

detect\_vmx\_features(void)

{

uint32\_t lo, hi;

/\* Pinbased controls \*/

rdmsr(IA32\_VMX\_PINBASED\_CTLS, lo, hi);

pr\_info("\n----->Pinbased Controls MSR: 0x%llx\n",

(uint64\_t)(lo | (uint64\_t)hi << 32));

report\_capability(pinbased, 5, lo, hi);

/\* True Pinbased controls \*/

rdmsr(IA32\_VMX\_TRUE\_PINBASED\_CTLS, lo, hi);

pr\_info("\n----->True Pinbased Controls MSR: 0x%llx\n",

(uint64\_t)(lo | (uint64\_t)hi << 32));

report\_capability(truepinbased, 5, lo, hi);

/\* Basic controls \*/

rdmsr(IA32\_VMX\_BASIC, lo, hi);

pr\_info("\n----->Basic MSR: 0x%llx\n",

(uint64\_t)(lo | (uint64\_t)hi << 32));

report\_capability(basic, 1, lo, hi);

/\* Procbased controls \*/

rdmsr(IA32\_VMX\_PROCBASED\_CTLS, lo, hi);

pr\_info("\n----->Procbased MSR: 0x%llx\n",

(uint64\_t)(lo | (uint64\_t)hi << 32));

report\_capability(procbased, 21, lo, hi);

/\* True Procbased controls \*/

rdmsr(IA32\_VMX\_TRUE\_PROCBASED\_CTLS, lo, hi);

pr\_info("\n----->True Procbased MSR: 0x%llx\n",

(uint64\_t)(lo | (uint64\_t)hi << 32));

report\_capability(trueprocbased, 21, lo, hi);

/\* Secondary Procbased controls \*/

rdmsr(IA32\_VMX\_PROCBASED\_CTLS2, lo, hi);

pr\_info("\n----->Secondary Procbased MSR: 0x%llx\n",

(uint64\_t)(lo | (uint64\_t)hi << 32));

report\_capability(procbased2, 22, lo, hi);

/\* Exit controls \*/

rdmsr(IA32\_VMX\_EXIT\_CTLS, lo, hi);

pr\_info("\n----->Exit controls MSR: 0x%llx\n",

(uint64\_t)(lo | (uint64\_t)hi << 32));

report\_capability(exitcontrol, 11, lo, hi);

/\* True Exit controls \*/

rdmsr(IA32\_VMX\_TRUE\_EXIT\_CTLS, lo, hi);

pr\_info("\n----->True Exit controls MSR: 0x%llx\n",

(uint64\_t)(lo | (uint64\_t)hi << 32));

report\_capability(trueexitcontrol, 11, lo, hi);

/\* Entry controls \*/

rdmsr(IA32\_VMX\_ENTRY\_CTLS, lo, hi);

pr\_info("\n----->Entry controls MSR: 0x%llx\n",

(uint64\_t)(lo | (uint64\_t)hi << 32));

report\_capability(entrycontrol, 11, lo, hi);

/\* True Control controls \*/

rdmsr(IA32\_TRUE\_ENTRY\_CTLS, lo, hi);

pr\_info("\n----->True Entry controls MSR: 0x%llx\n",

(uint64\_t)(lo | (uint64\_t)hi << 32));

report\_capability(trueentrycontrol, 11, lo, hi);

}

/\*

\* init\_module

\*

\* Module entry point

\*

\* Return Values:

\* Always 0

\*/

int

init\_module(void)

{

printk(KERN\_INFO "CMPE 283 Assignment 1 Module Start\n");

detect\_vmx\_features();

/\*

\* A non 0 return means init\_module failed; module can't be loaded.

\*/

return 0;

}

/\*

\* cleanup\_module

\*

\* Function called on module unload

\*/

void

cleanup\_module(void)

{

printk(KERN\_INFO "CMPE 283 Assignment 1 Module Exits\n");

}