Kernel Multitasking

From OSDev Wiki

This page is a work in progress and may thus be incomplete. Its content may be changed in the near future.

Difficulty level

Medium

In this tutorial, we'll cover the creation of a kernel-level multitasking system. That is, kernel threads and processes. The code here is specific to the i386 architecture.

Contents

Requirements

- Paging

task.h

This header defines the function and types used in here. Its code is

```
#ifndef __TASK_H__
#define __TASK_H__

#include <stdint.h>

extern void initTasking();

typedef struct {
  uint32_t eax, ebx, ecx, edx, esi, edi, esp, ebp, eip, eflags, cr3;
} Registers;

typedef struct Task {
  Registers regs;
  struct Task *next;
```

```
} Task;

extern void initTasking();
extern void createTask(Task*, void(*)(), uint32_t, uint32_t*);

extern void preempt(); // Switch task frontend
extern void switchTask(Registers *old, Registers *new); // The function w

#endif /* __TASK_H__ */
```

task.c

This file defines the actual wrappers that *switchTask()* uses, *createTask()* and *preempt()*:

```
#include "task.h"

static Task *runningTask;
static Task mainTask;
static Task otherTask;

static void otherMain() {
  printk("Hello multitasking world!"); // Not implemented here...
  preempt();
}

void initTasking() {
  // Get EFLAGS and CR3
  asm volatile("movl %%cr3, %%eax; movl %%eax, %0;":"=m"(mainTask.regs.cr
  asm volatile("pushfl; movl (%%esp), %%eax; movl %%eax, %0; popfl;":"=m'

  createTask(&otherTask, otherMain, mainTask.regs.eflags, (uint32_t*)mair
  mainTask.next = &otherTask;
  otherTask.next = &mainTask;

  runningTask = &mainTask;
}

void createTask(Task *task, void (*main)(), uint32_t flags, uint32_t *pag
  task->regs.eax = 0;
  task->regs.ebx = 0;
  task->regs.ecx = 0;
  task->regs.edx = 0;
  task->regs.esi = 0;
  task->regs.edi = 0;
  task->regs.eflags = flags;
  task->regs.eip = (uint32_t)main;
  task->regs.cr3 = (uint32_t)pagedir;
  task->regs.esp = (uint32_t)allocPage() + 0x1000; // Not implemented her
```

```
  task->next = 0;
}

void preempt() {
  Task *last = runningTask;
  runningTask = runningTask->next;
  switchTask(&last->regs, &runningTask->regs);
}
```

switch.S

This is the file that actually changes between tasks. It defines a function, *switchTask()*, which does all the magic. It saves all registers to *from* and loads them from *to*. It's more trickier than what you'll think. Its function prototype is

```
void switchTask(Registers *from, Registers *to);
```

Its code is

```
.text
.globl switchTask
switchTask:
pusha
pushf
mov %cr3, %eax ;Push CR3
push %eax
mov 44(%esp), %eax ;The first argument, where to save
mov %ebx, 4(%eax)
mov %ecx, 8(%eax)
mov %edx, 12(%eax)
mov %esi, 16(%eax)
mov %edi, 20(%eax)
mov 36(%esp), %ebx ;EAX
mov 40(%esp), %ecx ;IP
mov 20(%esp), %edx ;ESP
add $4, %edx ;Remove the return value ;)
mov 16(%esp), %esi ;EBP
mov 4(%esp), %edi ;EFLAGS
mov %ebx, (%eax)
mov %edx, 24(%eax)
mov %esi, 28(%eax)
mov %ecx, 32(%eax)
mov %edi, 36(%eax)
pop %ebx ;CR3
mov %ebx, 40(%eax)
push %ebx ;Goodbye again ;)
mov 48(%esp), %eax ;Now it is the new object
```

```
mov 4(%eax), %ebx ;EBX
mov 8(%eax), %ecx ;ECX
mov 12(%eax), %edx ;EDX
mov 16(%eax), %esi ;ESI
mov 20(%eax), %edi ;EDI
mov 28(%eax), %ebp ;EBP
push %eax
mov 36(%eax), %eax ;EFLAGS
push %eax
popf
pop %eax
mov 24(%eax), %esp ;ESP
push %eax
mov 44(%eax), %eax ;CR3
mov %eax, %cr3
pop %eax
push %eax
mov 32(%eax), %eax ;EIP
xchg (%esp), %eax ;We do not have any more registers to use as tmp storag
mov (%eax), %eax ;EAX
ret ;This ends all!
```

Doing it

Put this on some source file

```
#include "task.h"

void doIt() {
  printk("Switching to otherTask... \n");
  preempt();
  printk("Returned to mainTask!\n");
}
```

Now, from your *kernel_main()* call *doIt()*!

Congratulations! You've just implement kernel multitasking! If you want to call *preempt()* from your IRQ #0 handler, you should modify *switchTask()* to work with interrupts and *iret*.

Categories: Level 2 Tutorials | In Progress
