UNIVERSITY OF OREGON

Soteria

The Secure Kernel

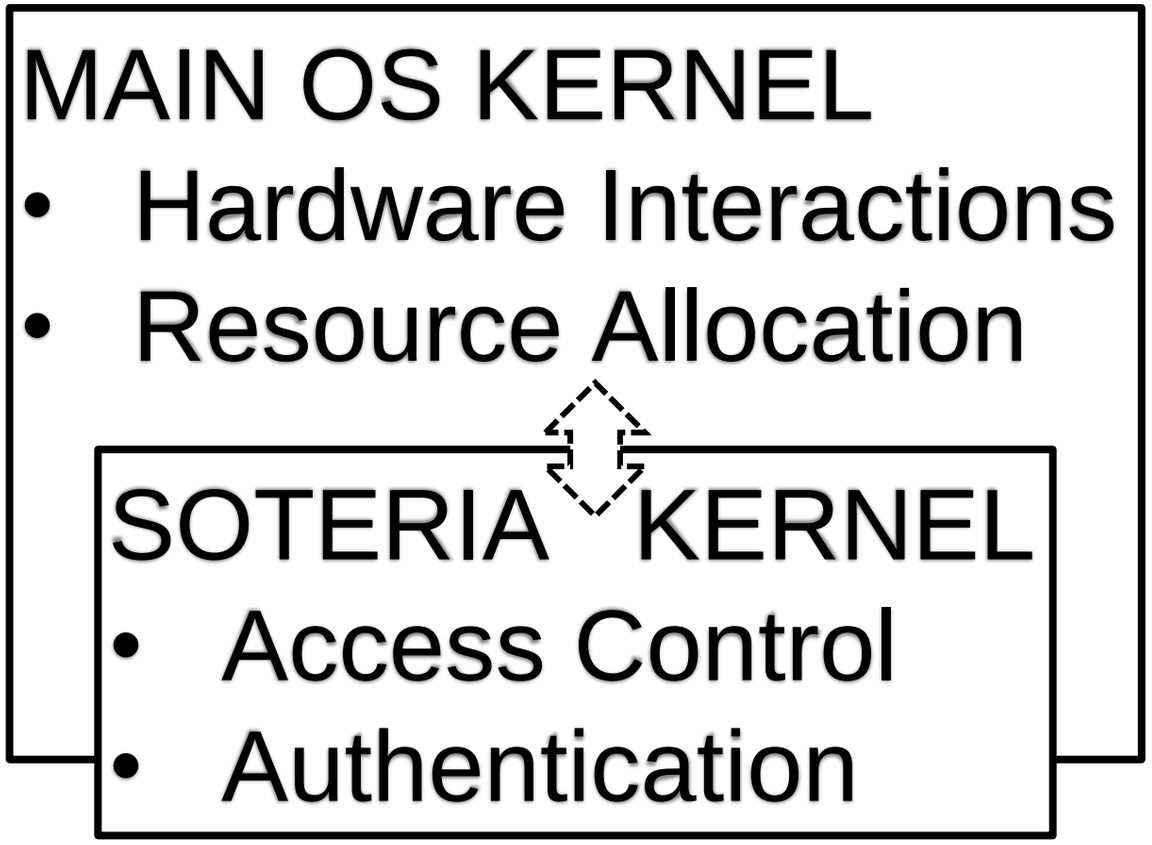
Haley Whitman & Andrew Hill

Computer Architecture and Embedded Systems Laboratory

General Architecture

UNIVERSITY OF OREGON

BIOS starts up, system is capable of multi-level boot.
Main OS kernel is initialized, Soteria soon after.



Soteria

UNIVERSITY OF OREGON

- Small in scale
- Separated from the main kernel driving the OS
- Complete coverage of all accesses from the OS kernel

Deliverables

UNIVERSITY OF OREGON

We need to create two things:

1. A main kernel running the file system and resources of small processes on the system.
2. A security kernel, running separately implementing a resource monitor and other authentication methods which will be used by the main kernel.

The main kernel will be very minimal, the focus is on the security kernel and the interactions between the two kernels.

What we have so far

UNIVERSITY OF OREGON

- We have created a main kernel that can be loaded with QEMU and supports simple input/output with the keyboard.

Next Steps for coding and documentation:

- Load two kernels, and begin simple interactions between the two.
- Create a file system on the main kernel.
- Implement a reference monitor concept on the security kernel

Code examples

```asm
;; Haley Whitman and Andrew Hill
;; Derived from Arjun Sreedharan

;;kernal.asm
bits 32 ;This is NASM specific, specifying a 32bit system.

global start
global entry
global read_port
global write_port
global load_idt
global keyboard_handler


extern kmain ;;Found in kernel.c
extern keyboard_handler_main

section .text
entry: jmp start
        ;To enable multiboot on Linux GRUB based systems, following parameters declared
        align 4
        dd 0x1BADB002 ;Magic field, declares header
        dd 0x00 ;Flags field, no flags set.
        dd - (0x1BADB002 + 0x00) ;Checksum for all parameters, magic + flag + checksum should be zero.


start:
    cli ;block all interrupts, causing OS to hang after execution.
    mov esp, stack_space ;Set stack pointer
    call kmain ;; C code for initializing both the kernel and the keyboard inputs and outputs
    hlt ;halt the cpu

read_port: ;; Initializing ports for individual keyboard inputs
    mov edx, [esp + 4]
    in al, dx
    ret

write_port: ; Initializing ports for individual keyboard outputs on the main window
    mov edx, [esp + 4]
    mov al, [esp + 4 + 4]
    out dx, al
    ret

load_idt: ; Loading the Interrupt Descripter Table (IDT), to handle keyboard interrupts and
          ; avoiding a busy CPU
    mov edx, [esp + 4]
    lidt [edx]
    sti ; this initiates interrupts
    ret

keyboard_handler: ; for all currently implemented interrupts
    call keyboard_handler_main
    iretd

section .bss
resb 8192 ;Specifying 8kb for the kernel's stack
stack_space:
```

```
/*
* Haley Whitman & Andrew Hill
* Derived from Arjun Sreedharan
*/

OUTPUT_FORMAT(elf32-i386)
ENTRY(start)
SECTIONS
{
    . = 0x100000;
    .text : {*(.text)}
    .data : {*(.data)}
    .bss : {*(.bss)}
}
```

These are the current assembly an linker files that are used in initializing the kernel. Kernel running is shown below.

Corresponding kmain.c file not shown.