Project 3 Report

seL4 Capabilities Project Report

Introduction

This report summarizes the implementation and testing of capability-based access control mechanisms in seL4. The project involved manipulating capabilities through the seL4 microkernel API to understand core concepts of capability management. All tasks were completed with proper error handling and verification, showing the practical application of capability-based access control in seL4.

Implementation Details

Task 1: CSpace Size Calculation

```
size_t initial_cnode_object_size_bytes = BIT(info->initThreadCNodeSizeBits +
seL4_SlotBits);
```

- This calculation determines the total bytes occupied by the initial thread's CSpace by combining CNode size bits (initThreadCNodeSizeBits) and individual slot size (seL4_SlotBits). The BIT macro performs left shift to calculate the actual size.

Task 2: Capability Copy

```
error = seL4_CNode_Copy(seL4_CapInitThreadCNode, last_slot, seL4_WordBits,
                        seL4_CapInitThreadCNode, seL4_CapInitThreadTCB,
seL4_WordBits,
                        seL4_AllRights);
```

- This implementation first copies the initial TCB capability to the last available slot, then maintains full rights for the copied capability. It uses direct CSpace addressing for both source and destination.

Task 3: Capability Deletion

```
error = seL4_CNode_Delete(seL4_CapInitThreadCNode, first_free_slot,
seL4_WordBits);
```

```
error = seL4_CNode_Delete(seL4_CapInitThreadCNode, last_slot, seL4_WordBits);
```

- Implements cleanup by space by deleting copied capabilities from both slots and ensuring a clean state verification through subsequent empty slot checks.

Task 4: Thread Suspension

```
error = seL4_TCB_Suspend(seL4_CapInitThreadTCB);
```

- This demonstrates capability usage by invoking thread control operations as well as using the original TCB capability for suspension.

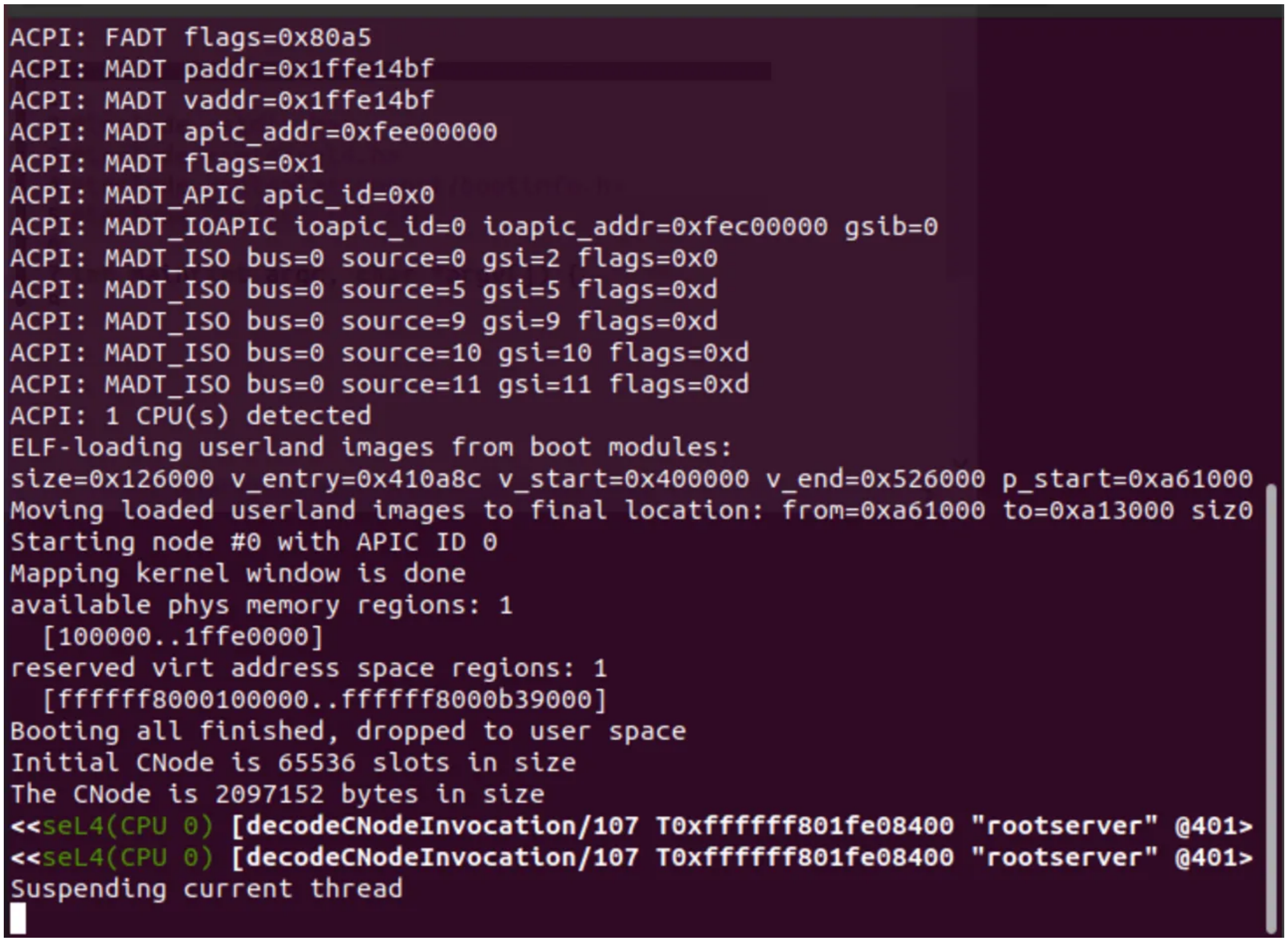
Execution Results

1. Initial Output:

```
Initial CNode is 65536 slots in size
The CNode is 2097152 bytes in size
```

2. Capability Operations:

- Successfully copied TCB capability
- Set thread priority without errors
- Verified slot emptiness after deletion
- Achieved thread suspension



```
ACPI: FADT flags=0x80a5
ACPI: MADT paddr=0x1ffe14bf
ACPI: MADT vaddr=0x1ffe14bf
ACPI: MADT apic_addr=0xfee00000
ACPI: MADT flags=0x1
ACPI: MADT_APIC apic_id=0x0
ACPI: MADT_IOAPIC ioapic_id=0 ioapic_addr=0xfec00000 gsib=0
ACPI: MADT_ISO bus=0 source=0 gsi=2 flags=0x0
ACPI: MADT_ISO bus=0 source=5 gsi=5 flags=0xd
ACPI: MADT_ISO bus=0 source=9 gsi=9 flags=0xd
ACPI: MADT_ISO bus=0 source=10 gsi=10 flags=0xd
ACPI: MADT_ISO bus=0 source=11 gsi=11 flags=0xd
ACPI: 1 CPU(s) detected
ELF-loading userland images from boot modules:
size=0x126000 v_entry=0x410a8c v_start=0x400000 v_end=0x526000 p_start=0xa61000
Moving loaded userland images to final location: from=0xa61000 to=0xa13000 siz0
Starting node #0 with APIC ID 0
Mapping kernel window is done
available phys memory regions: 1
  [100000..1ffe0000]
reserved virt address space regions: 1
  [ffffff8000100000..ffffff8000b39000]
Booting all finished, dropped to user space
Initial CNode is 65536 slots in size
The CNode is 2097152 bytes in size
<<seL4(CPU 0) [decodeCNodeInvocation/107 T0xffffff801fe08400 "rootserver" @401>
<<seL4(CPU 0) [decodeCNodeInvocation/107 T0xffffff801fe08400 "rootserver" @401>
Suspending current thread
█
```

Error Handling

The implementation includes comprehensive error checking:

- Capability operation validation
- Slot state verification
- Proper error reporting through ZF_LOGF_IF

Conclusions

Through this project, I was able to successfully demonstrates my new understanding of seL4's capability system. With this, came the proper manipulation of CSpace structures and capability management. (And thread control through capability invocation)