



UNIVERSITY OF DHAKA

Department of Computer Science and Engineering

CSE 3211: Operating System Lab

OS LAB ASSIGNMENT 03

Submitted By:

Joty Saha (Roll-51)

Ahona Rahman (Roll-59)

Submitted To:

Dr. Mosaddek Hossain Kamal

Submitted On: 19 March 2025

Operating System Lab Assignment 03

Design and Deployment of Syscall, Thread Creation, and Multi-tasking in DUOS

March 19, 2025

Contents

1	Introduction	2
2	Objectives	2
3	Implementation Details	2
3.1	Syscall Exception Handling	2
3.2	Task Scheduling	3
3.3	Memory Management	3
4	System Design	4
4.1	Task Control Block (TCB)	4
4.2	Syscall Handling	4
4.3	Scheduling Algorithm	5
5	Testing and Results	5
6	Conclusion	7
7	References	7

1 Introduction

This report provides a comprehensive analysis of the design, implementation, and deployment of system calls, thread management, and multitasking within the DUOS operating system. DUOS, a simplified OS designed for ARM-based embedded systems, incorporates essential kernel functionalities such as system calls, task scheduling, and memory management. The assignment aimed to build a robust system that efficiently handles multiple tasks while ensuring efficient resource utilization. The implementation involves creating a system call interface using SVC, scheduling tasks using SysTick and PendSV, and managing stack and heap memory effectively.

2 Objectives

The primary objectives of this assignment were:

- Implement system call exception handling using SVC and PendSV to allow user applications to safely access kernel-level functionalities.
- Develop a structured memory management system, including stack allocation for tasks and heap management for dynamic memory allocation.
- Implement thread creation and process scheduling using First-Come, First-Served (FCFS) and time-sharing policies to ensure efficient multitasking.
- Design an OS capable of handling at least 20 concurrent tasks while maintaining system stability and responsiveness.

3 Implementation Details

3.1 Syscall Exception Handling

System calls (syscalls) are a critical interface between user applications and the operating system kernel, allowing applications to request services such as I/O operations, process control, and memory management. In DUOS, syscalls are implemented using the SVC (Supervisor Call) instruction, which safely transitions the processor from user mode to privileged mode, allowing access to kernel functions.

The following syscalls were implemented:

- **SYS_exit**: This syscall terminates a process. It involves cleaning up resources allocated to the process, updating process tables, and possibly triggering a scheduler to select a new process to run.
- **SYS_getpid**: Retrieves the process ID of the calling process. This is useful for process management and debugging.
- **SYS_read**: Reads input from a device. This involves interfacing with device drivers to fetch data from hardware peripherals.
- **SYS_write**: Writes output to a device. Similar to **SYS_read**, this syscall interacts with device drivers to send data to hardware.

- **SYS_malloc:** Allocates memory dynamically from the heap. It updates the memory allocation table to track allocated blocks and ensures memory alignment.
- **SYS_free:** Frees previously allocated memory, updating the memory allocation table to mark the block as available.
- **SYS_fork:** Creates a new process by duplicating the calling process. This involves copying the process's memory space and creating a new task control block (TCB).
- **SYS_execv:** Replaces the current process image with a new program, effectively changing the process's code and data segments.

3.2 Task Scheduling

Task scheduling in DUOS is implemented using the SysTick and PendSV exceptions, which are part of the ARM Cortex-M architecture. These exceptions facilitate periodic interrupts and context switching, essential for multitasking.

- **SysTick Timer:** Configured to generate periodic interrupts, the SysTick timer serves as the heartbeat of the scheduler. It decrements a system tick counter and, when it reaches zero, triggers the PendSV exception to initiate a context switch.
- **PendSV Exception:** This exception is used for context switching between tasks. When triggered, it saves the context of the current task (registers, stack pointer) and restores the context of the next task in the ready queue.
- **Round-Robin Scheduling:** A simple yet effective scheduling policy, round-robin ensures fair CPU time distribution among tasks. Each task is given a fixed time slice, and tasks are cycled through the ready queue. The ready queue is implemented as a circular buffer, storing Task Control Blocks (TCBs) for each task.

3.3 Memory Management

Memory management in DUOS involves handling both stack and heap memory regions to ensure efficient resource utilization and task isolation.

- **Stack Memory:** Each task is allocated a dedicated stack space, used for local variables, function calls, and syscall processing. The stack pointer is part of the task's context and is saved/restored during context switches.
- **Heap Memory:** The heap is used for dynamic memory allocation, managed by the `SYS_malloc` and `SYS_free` syscalls. A memory allocation table tracks allocated and free blocks, ensuring efficient use of available memory and preventing fragmentation.

4 System Design

4.1 Task Control Block (TCB)

The Task Control Block (TCB) is a data structure that encapsulates all necessary information about a task, enabling the scheduler to manage and switch between tasks efficiently. The TCB structure includes:

Listing 1: Task Control Block Definition

```
1 typedef struct {  
2     uint32_t task_id;           // Unique identifier for the task  
3     void *stack_pointer;       // Pointer to the current top of the task's stack  
4     uint16_t status;           // Status of the task (e.g., running, ready, blocked,  
5     uint32_t execution_time;   // Total execution time of the task  
6     uint32_t waiting_time;     // Total waiting time in the ready queue  
7 } TCB_TypeDef;
```

- **task_id**: A unique identifier assigned to each task, used for tracking and managing tasks.
- **stack_pointer**: Points to the current top of the task's stack, crucial for context switching.
- **status**: Indicates the current state of the task, such as running, ready, or blocked.
- **execution_time**: Tracks the cumulative CPU time consumed by the task, useful for performance analysis.
- **waiting_time**: Measures the time spent by the task in the ready queue, aiding in scheduling decisions.

4.2 Syscall Handling

The syscall mechanism is organized into three key files, each serving a distinct purpose:

- **syscall_def.h**: This header file defines unique syscall numbers, ensuring each syscall can be identified and dispatched correctly.
- **syscall.h**: Contains function prototypes for all syscalls, providing a clear interface for user applications to invoke kernel services.
- **syscall.c**: Implements the actual kernel service functions. Each function corresponds to a syscall and performs the necessary operations, such as interacting with hardware or managing resources.

4.3 Scheduling Algorithm

The round-robin scheduler is implemented using a ready queue, which stores TCBs of tasks ready to execute. The scheduling process involves:

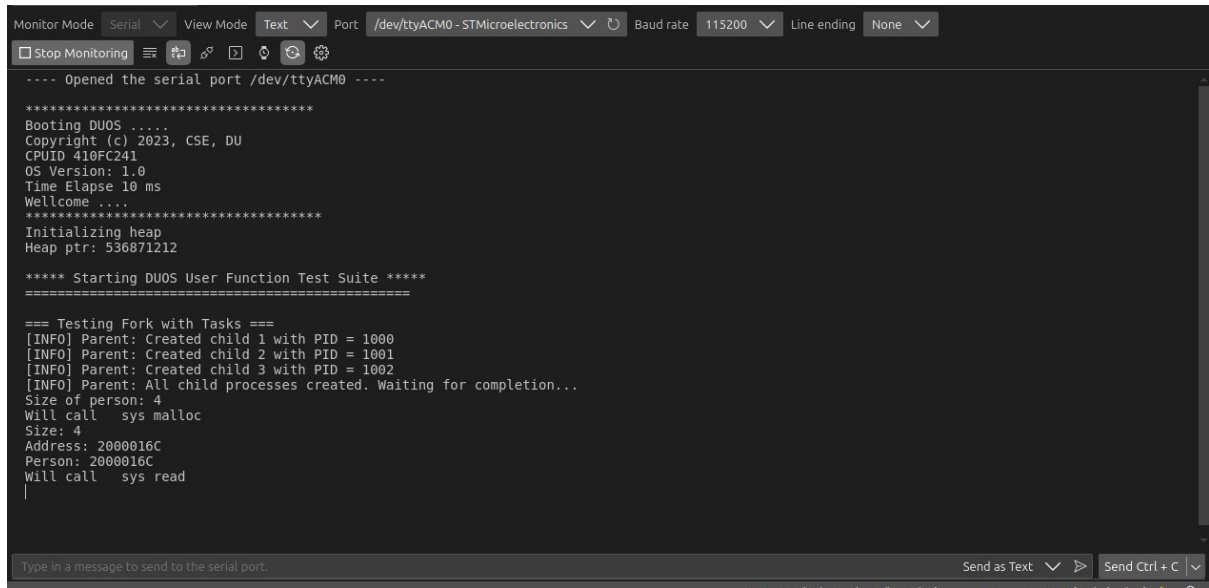
- **Ready Queue Management:** The ready queue is a circular buffer that holds TCBs. When a task's time slice expires, its TCB is moved to the end of the queue, and the next task is selected for execution.
- **Context Switching:** Triggered by the PendSV exception, context switching involves saving the current task's context (registers, stack pointer) and loading the next task's context. This ensures seamless task transitions without data loss.
- **Time Slice Allocation:** Each task is allocated a fixed time slice, determined by the SysTick timer interval. This ensures fair CPU time distribution and prevents any single task from monopolizing the processor.

5 Testing and Results

A test program was developed to verify syscall execution, task switching, and memory allocation. The following scenarios were tested:

- Creating multiple threads and verifying correct execution.
- Checking memory allocation and deallocation.
- Testing system calls for read, write, and process management.

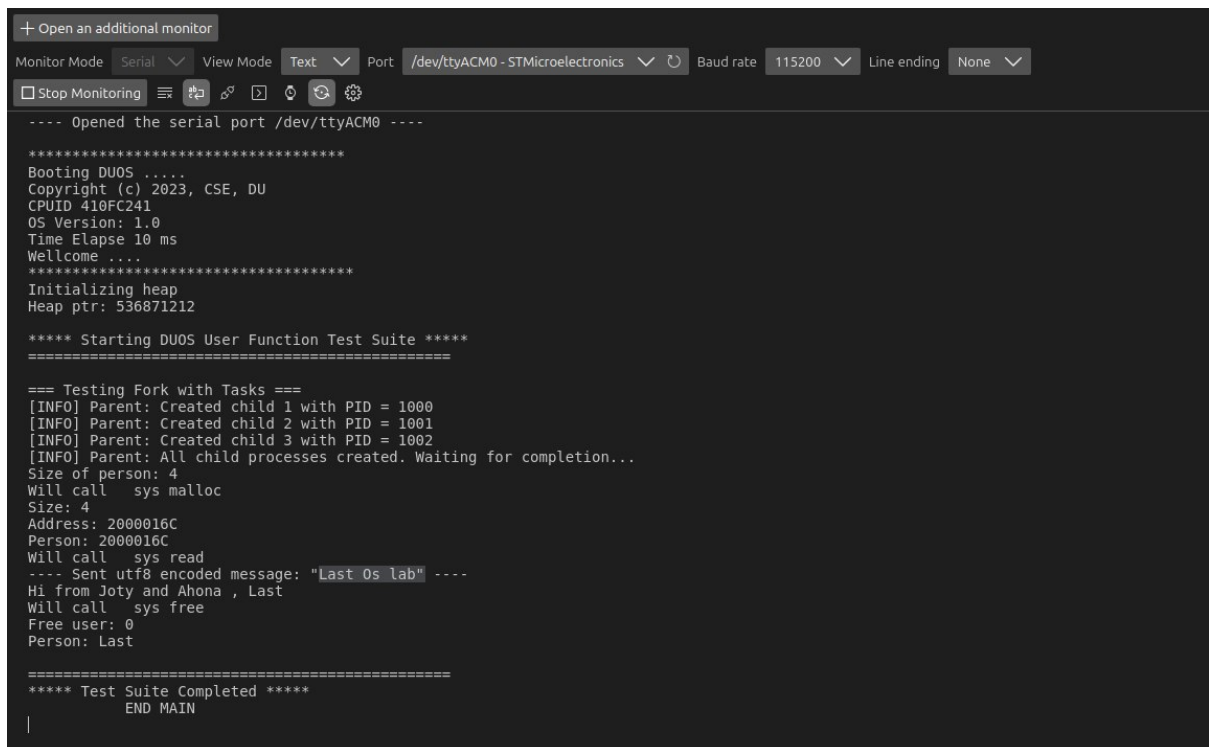
The results confirmed correct syscall behavior and successful task scheduling.



The image shows a Serial Monitor window with the following content:

```
Monitor Mode Serial View Mode Text Port /dev/ttyACM0 - STMicroelectronics Baud rate 115200 Line ending None
[Stop Monitoring] [Icons]
---- Opened the serial port /dev/ttyACM0 ----
*****
Booting DUOS .....
Copyright (c) 2023, CSE, DU
CPUID 410FC241
OS Version: 1.0
Time Elapse 10 ms
Wellcome ....
*****
Initializing heap
Heap ptr: 536871212
**** Starting DUOS User Function Test Suite ****
=====
=== Testing Fork with Tasks ===
[INFO] Parent: Created child 1 with PID = 1000
[INFO] Parent: Created child 2 with PID = 1001
[INFO] Parent: Created child 3 with PID = 1002
[INFO] Parent: All child processes created. Waiting for completion...
Size of person: 4
Will call sys malloc
Size: 4
Address: 2000016C
Person: 2000016C
Will call sys read
|
```

Figure 1: Serial Monitor before input



The image shows the same Serial Monitor window after some input. The output has been updated:

```
+ Open an additional monitor
Monitor Mode Serial View Mode Text Port /dev/ttyACM0 - STMicroelectronics Baud rate 115200 Line ending None
[Stop Monitoring] [Icons]
---- Opened the serial port /dev/ttyACM0 ----
*****
Booting DUOS .....
Copyright (c) 2023, CSE, DU
CPUID 410FC241
OS Version: 1.0
Time Elapse 10 ms
Wellcome ....
*****
Initializing heap
Heap ptr: 536871212
**** Starting DUOS User Function Test Suite ****
=====
=== Testing Fork with Tasks ===
[INFO] Parent: Created child 1 with PID = 1000
[INFO] Parent: Created child 2 with PID = 1001
[INFO] Parent: Created child 3 with PID = 1002
[INFO] Parent: All child processes created. Waiting for completion...
Size of person: 4
Will call sys malloc
Size: 4
Address: 2000016C
Person: 2000016C
Will call sys read
---- Sent utf8 encoded message: "Last Os lab" ----
Hi from Joty and Ahona , Last
Will call sys free
Free user: 0
Person: Last
=====
**** Test Suite Completed ****
END MAIN
|
```

Figure 2: Serial Monitor after input

6 Conclusion

This assignment provided hands-on experience in OS design, focusing on syscall implementation, task management, and scheduling. The developed system successfully supports multi-tasking with efficient context switching and memory management.

7 References

- ARM Cortex-M4 Technical Reference Manual
- Operating System Concepts, Silberschatz et al.
- DUOS OS Design Documentation