GTU-C312 Operating System Project

Project Overview

This project implements a simulated CPU (GTU-C312) with a custom instruction set and a simple operating system. The project was developed as part of the Operating Systems course at Gebze Technical University. The initial CPU structure and understanding of the instruction set was aided by AI tools, while the actual implementation and development of the operating system was done independently.

System Architecture

Memory Organization

The system uses a sophisticated memory management system with the following structure:

OS Memory Space (0-999)

1. Registers (0-20):
    - Register 0: Program Counter
    - Register 1: Stack Pointer
    - Register 2: System Call Result
    - Register 4: Thread Number (for context switching)
    - Register 5: Thread Start Time
    - Register 6: Thread Instruction Count
    - Register 7: Thread State (0: ready, 1: blocked, 2: running)
    - Register 17: System Call Type (0: HLT, 1: YIELD, 2: PRN)
    - Register 18: Thread Temp PC Switch
    - Register 19: Current Running Thread Number
    - Register 20: Total Instructions Executed
2. Thread Tables (21-324):
    - Each thread table occupies 25 memory locations
    - Thread table 0 (OS): [50-74]
    - Thread table 1: [75-99]
    - Thread table 2: [100-124] And so on...
3. Round Robin Scheduler (398-419):
    - Head pointer: [398]
    - Tail pointer: [399]
    - Thread entries: [400-419]
    - Each thread entry contains:
        - Thread number
        - Next thread pointer

Thread Memory Space (1000-10999)

Each thread has a 1000-size memory block with:

- Data section at the start
- Instruction section
- Stack (grows downwards)

Operating System Implementation

Thread Management

The OS implements a comprehensive thread table system that tracks:

- Thread ID (Register 4)
- Start time (Register 5)
- Instruction execution count (Register 6)
- Thread state (Register 7)
- Program Counter and Stack Pointer

Scheduling System

The OS uses a round-robin scheduling algorithm with:

1. A circular linked list implementation
2. Head and tail pointers for efficient thread rotation
3. Support for up to 10 threads
4. Thread state management (ready, blocked, running)

System Call Handling

The OS implements three system calls:

1. SYSCALL HLT (Type 0): Thread termination
2. SYSCALL YIELD (Type 1): CPU yield for scheduling
3. SYSCALL PRN (Type 2): Memory content printing

Memory Management

The system uses a sophisticated memory management system with:

1. OS Memory Blocks (0-999)
2. Thread Memory Blocks (1000-10999)
3. Separate data and instruction sections
4. Stack management for each thread
5. Memory protection (user mode can only access addresses 1000+)

Project Structure

Source Code Organization

```
src/
├── CPU.cpp/h          - CPU implementation
├── Memory.cpp/h       - Memory management
```

```
├── DataBlock.cpp/h   - Data segment handling
├── InstructionBlock.cpp/h - Instruction handling
├── AMemoryBlock.cpp/h    - Abstract memory block
└── main.cpp          - Program entry point

include/              - Header files
functions/            - OS functions
├── round_robin.os    - Scheduler implementation
└── round_robin_remove_thread.os - Thread management
```

Build System

The project uses a Makefile for building and includes debug modes for:

- Memory content tracing
- Thread table monitoring
- Step-by-step execution

Development Process

The project development followed these steps:

1. Initial CPU structure and instruction set understanding (with AI assistance)
2. Independent implementation of the CPU simulator
3. Development of the operating system in GTU-C312 code
4. Implementation of thread management and scheduling
5. Testing and debugging

Conclusion

This project successfully implements a simulated CPU with a custom instruction set and a simple operating system. The implementation demonstrates key operating system concepts including thread management, scheduling, and system calls. The project was developed with minimal AI assistance, primarily used for understanding the initial CPU structure and instruction set, while the actual implementation was done independently.