Title: Implementing a Scheduler on STM32F401CCU6

Abstract

This project presents the design and implementation of a round-robin scheduler on the STM32F401CCU6 microcontroller, a high-performance foundation line, ARM Cortex-M4 core with DSP and FPU, 512 Kbytes of Flash memory, 84 MHz CPU, and ART Accelerator. The scheduler operates by allocating time slices to each task in equal portions and in a circular order, handling all tasks without priority. The project includes four user tasks, each of which has its own stack to create local variables when it runs on the CPU. The context or state of the task is saved in the task's private stack when the scheduler decides to remove a task from the CPU.

The SysTick timer, a basic timer present in the Cortex-M processors, is used to generate an exception every 1ms to run the scheduler code. The SysTick handler is initially used to carry out the context switch operation between multiple tasks. Later, the code is modified to use the PendSV handler, which is an exception type that has the lowest priority among all exception types, making it most suitable for performing a context switch in an OS kernel.

Each task in this project can consume a maximum of 1 KB of memory as a private stack. This stack is used to hold tasks' local variables and context. When a task is getting scheduled for the very first time, it doesn't have any context. So, the programmer should store dummy SF1 and SF2 in the Task's stack area as a part of the "task initialization" sequence before launching the scheduler.

The project introduces a blocking state for tasks. When a task has got nothing to do, it should simply call a delay function which should put the task into the blocked state from running state until the specified delay is elapsed. The scheduler should schedule only those tasks which are in the Running state. The scheduler also should unblock the blocked tasks if their blocking period is over and put them back to running state.

The project also includes an idle task that runs on the CPU if all the user tasks are blocked. The idle task is like user tasks but only runs when all user tasks are blocked, and you can put the CPU to sleep. The project demonstrates the successful implementation of a scheduler on the STM32F401CCU6 microcontroller. The scheduler effectively schedules multiple user tasks in a round-robin fashion and introduces a blocking state for tasks. The inclusion of an idle task ensures efficient use of the CPU. The project provides a practical example of real-time operating system scheduling, demonstrating the potential for further exploration and development in this field.

Introduction

The project revolves around the STM32F401CCU6 microcontroller, a high-performance foundation line, ARM Cortex-M4 core with DSP and FPU, 512 Kbytes of Flash memory, 84 MHz CPU, and ART Accelerator. The primary objective is to implement a scheduler that operates in a round-robin fashion.

In the realm of computing, a scheduler is a fundamental component of an operating system that manages the execution of tasks. The round-robin scheduling algorithm, used in this project, is one of the simplest and most widely used algorithms. It allocates time slices to

each task in equal portions and in a circular order, handling all tasks without priority. This method ensures fairness, as each task gets an equal share of the CPU time.

The project includes four user tasks, each of which has its own stack to create local variables when it runs on the CPU. The context or state of the task is saved in the task's private stack when the scheduler decides to remove a task from the CPU. This process is known as context switching, a critical feature of multitasking operating systems. Context switching allows a single CPU to be shared by multiple tasks, enhancing the overall efficiency of the system.

The SysTick timer, a basic timer present in the Cortex-M processors, is used to generate an exception every 1ms to run the scheduler code. The SysTick handler is initially used to carry out the context switch operation between multiple tasks. Later, the code is modified to use the PendSV handler, which is an exception type that has the lowest priority among all exception types, making it most suitable for performing a context switch in an OS kernel.

The project introduces a blocking state for tasks. When a task has got nothing to do, it should simply call a delay function which should put the task into the blocked state from running state until the specified delay is elapsed. The scheduler should schedule only those tasks which are in the Running state. The scheduler also should unblock the blocked tasks if their blocking period is over and put them back to running state.

The project also includes an idle task that runs on the CPU if all the user tasks are blocked. The idle task is like user tasks but only runs when all user tasks are blocked, and you can put the CPU to sleep. This feature ensures that the CPU is never idle and is always performing useful work, thereby improving the overall efficiency of the system.

In conclusion, this project provides a practical example of real-time operating system scheduling, demonstrating the potential for further exploration and development in this field. The successful implementation of a scheduler on the STM32F401CCU6 microcontroller effectively schedules multiple user tasks in a round-robin fashion and introduces a blocking state for tasks. The inclusion of an idle task ensures efficient use of the CPU. The project serves as a foundation for more complex scheduling algorithms and techniques, paving the way for future advancements in this area.

Methodology

The methodology of this project involves several steps, each of which contributes to the successful implementation of a round-robin scheduler on the STM32F401CCU6 microcontroller.

1. **SysTick Timer Configuration**: The SysTick timer, a basic timer present in the Cortex-M processors, is used to generate an exception every 1ms to run the scheduler code. The SysTick handler is initially used to carry out the context switch operation between multiple tasks. The SysTick timer count clock is brought down from 84MHz to 1KHz using a divisor (reload value). The reload value is calculated based on the processor clock and the desired exception frequency.
2. **Task Stack Initialization**: Each task in the project can consume a maximum of 1 KB of memory as a private stack. This stack is used to hold tasks' local variables and context. When a task is getting scheduled for the very first time, it doesn't have any

context. So, the programmer should store dummy SF1 and SF2 in the Task's stack area as a part of the "task initialization" sequence before launching the scheduler.

3. **Switching Stack Pointer**: The code is initially using MSP (Main Stack Pointer) as the stack pointer. Since tasks should run in the Thread mode, the stack pointer has to be switched to PSP (Process Stack Pointer).
4. **Blocking State for Tasks**: The project introduces a blocking state for tasks. When a task has got nothing to do, it should simply call a delay function which should put the task into the blocked state from running state until the specified delay is elapsed. The scheduler should schedule only those tasks which are in the Running state. The scheduler also should unblock the blocked tasks if their blocking period is over and put them back to running state.
5. **Idle Task**: The project also includes an idle task that runs on the CPU if all the user tasks are blocked. The idle task is like user tasks but only runs when all user tasks are blocked, and you can put the CPU to sleep.
6. **PendSV Handler**: Later in the project, the code is modified to use the PendSV handler to carry out the context switch operation instead of the SysTick handler. The PendSV handler is an exception type that has the lowest priority among all exception types, making it most suitable for performing a context switch in an OS kernel.
7. **Fault Handlers**: The project includes fault handlers for HardFault, MemManage, and BusFault. These handlers print out the type of exception and enter an infinite loop, allowing for easier debugging if any of these faults occur.
8. **Task Handlers**: The project includes four user tasks, each of which toggles an LED at a different frequency. Each task turns on an LED, calls the delay function to block itself for a certain period, then turns off the LED and calls the delay function again.

This methodology ensures that the scheduler effectively schedules multiple user tasks in a round-robin fashion and introduces a blocking state for tasks. The inclusion of an idle task ensures efficient use of the CPU. The project serves as a foundation for more complex scheduling algorithms and techniques, paving the way for future advancements in this area.

Results

The project successfully implements a scheduler on the STM32F401CCU6 microcontroller. The scheduler schedules multiple user tasks in a round-robin fashion by carrying out the context switch operation. The project also introduces a blocking state for tasks and an idle task to run on the CPU when all user tasks are blocked.

Conclusion

In conclusion, this project demonstrates the successful implementation of a round-robin scheduler on the STM32F401CCU6 microcontroller. The scheduler effectively schedules multiple user tasks in a round-robin fashion, ensuring that each task gets an equal share of the CPU time. This approach guarantees fairness and prevents any single task from monopolizing the CPU.

The introduction of a blocking state for tasks is a significant achievement of this project. It allows tasks to efficiently use CPU time, only running when they have work to do. This feature significantly improves the overall efficiency of the system by preventing tasks from unnecessarily consuming CPU time.

The inclusion of an idle task further enhances the efficiency of the system. The idle task runs on the CPU if all the user tasks are blocked, ensuring that the CPU is never idle and is always performing useful work. This feature demonstrates a thoughtful design approach that takes into consideration the efficient use of system resources.

The project also successfully handles context switching using the SysTick handler and later the PendSV handler. Context switching is a critical feature of multitasking operating systems that allows a single CPU to be shared by multiple tasks. The successful implementation of context switching in this project underscores the robustness and reliability of the scheduler.

Overall, this project serves as a foundation for more complex scheduling algorithms and techniques, paving the way for future advancements in this area. It provides a practical example of real-time operating system scheduling, demonstrating the potential for further exploration and development in this field. The methodologies and techniques used in this project could be applied to other similar projects, contributing to the broader field of real-time operating system scheduling. The project not only achieves its objectives but also opens up new avenues for research and development in the field of task scheduling in real-time operating systems. It stands as a testament to the potential of round-robin scheduling in efficiently managing tasks in a multitasking environment.

References

The project uses the STM32F401CCU6 microcontroller and its associated peripherals. The SysTick timer is used to generate exceptions, and the PendSV handler is used for context switching. The project also uses inline assembly for certain operations.

Please review and let me know if there are any sections you would like me to revise or expand upon.