Real-Time Operating System User Guide (Motorola 68000)

1. Introduction

The Motorola 68000 (68K) is a chip released by Motorola in the late 1970s, which was based on the complex instructions set computer (CISC) architecture [1]. The 68K was one of the first 32-bit microprocessors introduced, which contributed to its use throughout the 1980s in many computer and arcade systems [1]. According to [1], the 68 K-based controller was commonly used in many industrial applications, as many industrial controllers built in the 1980s may only be replaced decades later. The 68K was found to be used on many gaming consoles, such as the Sega Genesis and Sega Saturn, due to the inexpensive price of 68K [1]. However, the most common application for the 68K was for the embedded system industry from the early 1980s to the 1990s [1].

This project will be a lightweight real-time operating system (RTOS) to handle concurrent tasks. RTOS is an operating system that depends on the execution of functions and tasks within a limited timeframe [2]. This report will explain the implementation details of RTOS and act as a guide for the end user. There are some concepts regarding our system which are discussed further. It is important to note that Motorola 68000 was discontinued in 1996 [1].

The fundamental idea behind the multitasking concept in an operating system is the simultaneous or concurrent execution of multiple tasks. Time-slicing is employed as a technique to allocate the operating system's time among various tasks. The way of implementation varies as processor architecture, and, in the case of Motorola 68000, microprocessors have certain methods. The method of implementation is dependent on their hardware and software mechanisms.

1. Operating Modes: In this case, we are using a user mode with limited authority to access system resources in comparison to the kernel. Furthermore, the kernel has more privileged access to the system's hardware, memory, drivers, and interrupt control.
2. Timer interrupts: set at 1000 milliseconds to generate periodic intervals. During this recurring interval of time, the system switches between several jobs based on our predefined conditions. It is one of the multitasking approaches that assigns specific time periods to multiple jobs. We established two tasks, 1 and 2, which run concurrently every 1000 milliseconds (1 second interval).
3. Interrupt service routine (ISR): This is a key that allows you to access the operating system and initiate a shift in control flow to perform a certain function. It is built primarily on three concepts: interrupt vectors, task switching, and task control blocks. In this assignment, provide six numbers of software interruptions for functions 1–6. The address of the trap name is altered based on the variable

initialization. For example, insert the name of ***trap 0*** into the operating system to get trap ***#scs***. At the same time, ISR saves the current task to the chosen memory location, which includes all registers (D0-D7 and A0-A7), the programme counter (PC), and the status register (SR). Finally, restore the required task content. At the same time, ISR saves the current job to the chosen memory location, which includes all registers (D0-D7 and A0-A7), the programme counter (PC), and the status register (SR). Finally, restore the changed task content. At last, task control blocks ensure that the operating system's data structure remains consistent. All relevant data information in a task is stored as a concept of offset memory address.

2. Memory:

The purpose of the operating system (OS) is to manage memory dynamically. Memory usage and handling is the responsibility of the OS, which relieves the user from direct memory handling. This will enable the user to focus on application development rather than managing low-level details. It is essential that specific areas of the memory should never be written by the user, as it is used by the OS to function correctly.

Memory segment	Address Location
System Memory (OS only)	$0 - $1000
Tasks storage list (OS only)	$1000 - $1900
User Tasks	From $8000

User tasks are stored in memory above the system. Each task has its own memory space, with the programme code at the lowest address, data at the next, and the topof-stack slightly above the task's memory region. For example, task 1 began at memory location 1000H and ended at 1100H, and task 2 and task 3 began with every 100H memory difference. In the context of this programme, accessing memory with an offset is frequently connected with indexed addressing techniques.

For example, ***move.l d0, tcbd0 (a0)*** provides a new memory allocation according to the base value of a0. The effective address is determined by adding the contents of register A0 (an address register) with the displacement or offset provided in parentheses.

1. The source operand is the 32-bit value stored in data register D0.
2. The effective address for the destination is determined by adding the contents of register A0 to the displacement indicated in brackets. The result becomes the destination address.
3. The determined destination address is then used to store the 32-bit value from the source register D0 in memory.

The setup process is straightforward, and the message is displayed in a pop-up window figure. 01. However, all internal variables are reset, including registers (D0D7,

A0-A7), programme counters (PC), status registers (SR), and flags. To begin, a tcb0 is initialised for task 0 and then left unused. This tcb0 set corresponds to a running task in the operating system, which is also regarded as a default user task. Finally, it branches to the user application task area under the task0 name. The task 0 operation is then performed to enter the operating system using the trap #sys call. Task 1 requests that the operating system create a new task using the parameter addresses specified in d0 and d1. The new task's starting address is in the d0 register, and the top-of-stack is in d1.

3. Schedular Information

The schedular will be responsible for choosing the next task to run. It is important to note that only tasks registered and tasks in ready mode will be considered runnable. According to [2], in many operating systems, tasks tend to be blocked or runnable by the operating system. This operating system uses a similar concept as in [2], means a task can be in waiting or ready mode.

The schedular system will be switching tasks based on time intervals. The time interval switch is based on continuous time-based events, which tend to be used in many operating systems, typically called round-robins algorithm [2]. In other words, no tasks have priority over other tasks. All tasks in the scheduler will most likely be given the same time duration as other tasks. The tasks are arranged in a cyclically linked list for ease of implementation and predictability. Circular linked lists are typical linked lists with exceptions for the last item in the list, which is connected to the first item [3].

4. Operating System (OS) services

Our OS provides the following functionalities to the user. The processor can be operated from a maximum internal clock frequency of 25 MHz. The 68000 is available in several frequencies, including 4, 6, 8, 10, 12.5, 16.67, and 25 MHz [4]. In our case, we are considering the maximum frequency to calculate the time required for executing each of the functions.

4.1. Create Task

This function is a two-parameter call to the system that manages the addition of the user’s code as a task. The function will take the address of the code, written by the user, to be registered by RTOS. The function will dynamically manage the memory and add the code address as a task to be executed. All tasks registered will be placed in memory.

There is limited memory to be occupied for task registration. The function will search for empty memory space in the linked list. When the list is full, an error text message will appear on the connected terminal, and the microprocessor will halt

execution until it resets. The error handling feature will enable better feedback to the user of RTOS. It takes 1308 cycles to run (approx. 0.05232 ms) or 86 instructions.

4.2. Delete Task

The task removal function will remove specified tasks from the execution list. The function will accept a single parameter that contains the start address of the code (pointer to label). Then, the removal function will iterate through the list until it finds the task that matches the parameter. Once the task is found, it will be disabled and considered an empty space for new tasks to be added. The final stage is to clean up the links between the tasks to make sure all tasks are accessible. It takes 1302 cycles to run (approx. 0.05208 ms) or 50 instructions.

4.3 Wait Mutex

This function is responsible for moving tasks between the ready list and the wait list based on their availability, i.e. if it's set to zero, then your task will be moved to the wait list and wait for the signal mutex function to be available. If it is set to one, it will change to zero and move your task to the ready list. The whole function takes 164 cycles to execute (approx. 0.00656ms).

4.4 Signal Mutex

The signal mutex is an essential function in the implementation of the mutex. It enables the mutex function for the task requesting it. If it is zero, that means it is available and will move your task to the ready list. If none of the tasks are requested for a mutex function, then it changes its state to one. It takes 670 cycles to execute (approx. 0.0268ms).

4.5 Initialize Mutex

This function simply initializes the mutex variable to be used. It has two parameters, namely 0 and 1. This is called when a user intends to use the mutex function for his task execution. It takes 590 cycles to run (approx. 0.0236 ms).

All the mutex functions combined take about 64 instructions to be implemented.

4.6 Wait Time

If the user requires a type of delay or is in need of applying an interruption to run a high priority task, this is where we provide the functionality of a Wait Timer function. This can be set to a predefined time interval for the succeeding action. This takes 776 cycles to execute (approx. 0.03104 ms) or 24 instructions.

5. API and usage

5.1. User guidelines

The user is required to start writing the code under the label “T0” (main label). This is the first location the execution will jump into after initialization. It is important to note that any code under the main label will only be called once. It is recommended for the user to register at least one task under the main label.

Generally, the user should never jump to labels and sections that are used in other tasks. However, an exception can be made for jumping from the main label to other tasks. Jumping between different code blocks assigned to separate tasks could potentially cause a task-fusion error in the system. Taskfusion is when two tasks get locked in running the same block of code, which will result in double code execution and other unpredictable behaviour.

5.2. System calls

The operating system (OS) provides utilities for the user application. System calls allow the user to call the OS at any time using interrupt calls. Depending on the argument provided by the user, multiple services can be called. Some services include task registration, task removal, and calling sleep timers for the task. The OS can be called by calling the ***trap*** symbol and providing a value of ***#sys*** as the argument. The user will need to fill in the parameters (registers D0 to D2) manually before calling the system.

Function	D0	D1	D2
Create a task	$1	Address of the task (label address)	Address of the stack
Remove a task	$2	Address of the task to be removed	Not used
Move task to waiting list for specific duration	$3	Address of the task to be moved	Time duration
Wait mutex call	$4	Address of the task to be moved	Not Used

Table 1: Functions description that is provided for the user

6. Error handle and User interface

The operating system has built-in error reporting as a part of the user interface. During a call to the system, if the specified address is not found (common logical error), it will print the error to the terminal and halt the program. This is

implemented to make debugging more convenient for the user. For example, when attempting to register a task when the maximum number of tasks is reached, an error will state that no new tasks can be registered. This will give a hint to the user to resolve the issue.

7. Additional Information

Our system overall takes 2036 cycles to run (approx. 0.08144ms), which primarily involves switching between the tasks in a set of 300 instructions. The priority task scheduling is implemented through an interruption, which halts the currently executing task and transfers control to a high-priority task. Apart from the time consumed by the system for running the functions, the user tasks generally have 100ms of time to run, which also depends on the user-given time interval.

References:

[1] "Motorola 68000". *Wikipedia*. Accessed: Jan. 19th, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Motorola_68000

[2] "Real-time operating system". *Wikipedia.* Accessed: Jan. 18th, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Real-time_operating_system

[3] "Introduction to Circular Linked List", *GeeksforGeeks.* Accessed: Jan. 19th, 2024. [Online]. Available: https://www.geeksforgeeks.org/circular-linkedlist/#:~:text=What%20is%20Circular%20linked%20list,no%20NULL%20at%20the%2 0end.

[4] M. Rafiquzzaman. "Microprocessor Theory and Applications with 68000/68020 and Pentium". *Oreilly*. Accessed: Jan. 19th, 2024. [Online]. Available: https://www.oreilly.com/library/view/microprocessor-theoryand/9780470380314/13_ch06.html#:~:text=The%20processor%20can%20be%20op erated,%2C%2016.67%2C%20and%2025%20MHz