School of Electrical, Electronic & Computer Engineering

UNIVERSITY OF
NEWCASTLE UPON TYNE

User Manual

Real-Time Embedded Systems - EEE8087

Shihao Zhou, ID: 230594553

August 2024

Contents

RTS user manual

Overview

Welcome to this Real time System, developed in assembly language on the Motorola 68000 CPU. Whether you are developing a new application or maintaining an existing system, this manual will provide the detailed information and guidance you need to understand and use this real time operating system. By reading this manual, you will learn about the system's multitasking principles, memory layout, and user functions such as Reset, Create and Delete Task, and Mutex. I hope this manual will provide you with comprehensive support and help to make your development work more efficient and successful.

1. Multitasking and Time-Slicing

Multitasking is the ability to run multiple programs or processes at the same time. This does not mean that the processes are actually running at the same time, it means that the operating system gives the user the sense that they are all running at the same time by switching between them quickly so that each program or process has enough time to run.

There are two main ways that operating systems implement multitasking: time-slice rotation and priority scheduling. Time-slice rotation means that the operating system divides the CPU's time into small segments (called time slices) and then assigns these time slices to each program or process that needs to run. Each program or process can then use the CPU to perform calculations once it has been given a time slice. When the time slice runs out, the operating system switches to the next program or process. This approach is simple and fair, but may lead to frequent interruptions of tasks that require a lot of computation.

Priority scheduling, on the other hand, determines the order in which programs or processes are run based on their importance. Important programs or processes are given a higher priority and thus more CPU time. This approach ensures that important tasks are prioritized but may result in some low priority tasks not being executed for a long time.

The time-slice rotation in this system is achieved by timer interrupt. The system sets

a timer, and when a timer interrupt occurs, the system triggers the scheduler to switch tasks. The scheduler saves the state information of the current task, including all registers, program counters and stack pointers, and then selects the next task from the task list and restores its state. In this way, the system is able to allocate CPU time equitably among multiple tasks, and each task gets a certain amount of execution time.

2. Memory Layout

The code sections of a user task have a defined start address in memory, which in this system is $1000. The code sections of each task are independent of each other. The system sets the entry address of the user task to this start during initialization and loads its code segment to this address when a new task is created.

Each user task has a unique stack with a start address of $8000. The stack is used to store local variables and temporary data while the task is running. The system allocates a stack space for each task when the task is created and initializes the stack pointer to point to the top of that space.

3. Startup Behavior

1. Reset

   I set up the TCB list, task stack and other necessary components. TCB is the critical data structure used to manage the tasks. This system uses 4 TCB. Each task has a relative TCB which is used to save the task's state information including register values, program counter (PC), stack pointer and so on.
2. Branch to user task

   According to the user task, different functions are implemented.

   For example, create a new task, for deleting tasks, waiting for timer, etc.
3. Define function ID:

   Set unique id(1 - 7) for each interrupt and system service function, this system stores this id in d7 register.
4. Allocate TCB

   Allocate a separate TCB for each task to store the status information of the task,

including all data registers, address registers (a7 is used as the stack pointer in this system), program counter (PC) and status register (SR) which are saved to the stack after interrupt.

5. Interrupt Vector Definition

   When timer hardware interrupt and software interrupt triggers system service, the function ID is recorded and the register value is saved to the TCB of the corresponding task.

6. Save Task Data

   When an interrupt happens, the register status of the current task is saved into its TCB. This includes all data registers, address registers (A7 is the stack pointer), and PC and SR saved to the stack during the interrupt, as well as restoring PC and SR from the stack.

7. Handling interrupts

   For timer interrupts, the user service function is not executed and jumps directly to the Scheduler.

   For software interrupts, identify the system service function that needs to be run by identifying the id and then jump to (Dispatcher)

8. Scheduler

   Divides tasks by state into a ready task list and a waiting task list, and sorts them according to priority, which is determined by the parameters used to create the task. The scheduler finds the next task that needs to be run according to the priority and saves its TCB address

9. Dispatcher

   The scheduler loads the TCB of the task that needs to run into the system, restores its saved data registers and address registers, and saves the PC and SR registers to the stack. At the end of the interrupt, the system automatically jumps to the address indicated by the PC value in the stack to execute the task.

10. Cycle Steps

    Repeat the above steps whenever a timer expires or a system service is called.

4. User Functions

Name:	Create task	id:	1	Time Estimate:	30-50 instructions
Parameters:	The start address of the new task. The address of its top-of-stack.				
Function:	A currently unused TCB is marked as in use and set up for a new task. It is placed on the ready list. The requesting task remains on the ready list. Two parameters indicate the start and end of the memory area occupied by the new task and its data.				

Code Example:	`move.l #syscr,d0 ;start task 2`
	`move.l #t2,d1 ;address of task 2 will be save in data register 1 (d1)`
	`move.l #$5000,d2 :5000 is stack pointer(sp or ss) address`
	`trap #sys`				

Name:	Delete task	id:	2	Time Estimate:	30-50 instructions
Parameters:	None				
Function:	The requesting task is terminated, its TCB is removed from the list and marked as unused. Any memory allocated to it is returned to the system.				

Code Example:	`delete_task`
	`move.l #sysdel,d0 ; delete task`
	`trap #sys ; software interrupt`
	`bra t2`				

Name:	Wait mutex	id:	3	Time Estimate:	30-50 instructions
Parameters:	None				
Function:	If the mutex variable is one, it is set to zero and the requesting task is placed back onto the ready list. If the mutex is zero, the task is placed onto the wait list, and subsequently transferred back to the ready list when another task executes a signal mutex.				
Code Example:	`move.l #syswtmx,d0` `trap #sys`				

Name:	Singnl mutex	id:	4	Time Estimate:	30-50 instructions
Parameters:	None				
Function:	If the mutex variable is zero, and a task is waiting on the mutex, then that task is transferred to the ready list and the mutex remains at zero. If the mutex is zero and no task is waiting, the mutex is set to one. In either case, the requesting task remains on the ready list.				
Code Example:	`move.l #syssgmx,d0` `trap #sys ; software interrupt`				

Name:	Initialize mutex	id:	5	Time Estimate:	30-50 instructions
Parameters:	0 or 1				
Function:	The mutex is set to the value 0 or 1, as specified in the parameter.				

Code Example:	`sys_init_mutex`
	`move.l d1,mutex`
	`bra sched`				

Name:	Wait timer	id:	6	Time Estimate:	30-50 instructions
Parameters:	Number of timer intervals to wait.				
Function:	The requesting task is placed onto the wait list until the passage of the number of timer interrupts specified in the parameter, when it is transferred back to the ready list.				

Code Example:	`move.l #syswttm,d0 ; wait time system call`
	`move.l #10,d1 ; wait for 10 timer intervals`
	`trap #sys ; software interrupt`				

5. Other Relevant Aspects

The behaviors of this system involve functions such as task scheduling, multitasking, time slice rotation, mutual exclusion lock management and timer waiting. The detailed description is given below:

1. Task Scheduling

Task scheduling uses a time-slice rotation mechanism to ensure that all ready tasks get a fair runtime on the CPU. The scheduler is triggered at the end of each time slice and is implemented through timer interrupts.

2. Priority Scheduling

Currently RTS does not implement priority scheduling, all ready tasks get similar runtime. Each task rotates at the end of the time slice, ensuring that all tasks get execution time and avoiding any task being left unexecuted for a long time.

3. Time Slice Rotation

Time-slice rotation is implemented through timer interrupts. Each time slice is 100 milliseconds, and when the time slice ends, a timer interrupt triggers the scheduler to switch tasks. Through this mechanism, the system is able to allocate CPU time fairly among multiple tasks.

4. Mutex Lock Management

Mutually exclusive locks are used to protect shared resources and ensure that only one task can access the resource at the same time. Mutual exclusion lock management includes waiting for mutual exclusion locks and releasing mutual exclusion locks.

Waiting for mutex locks (syswtmx):

When a task tries to acquire a mutex lock, the system checks to see if the lock is available. If the mutex lock is already occupied, the task enters the wait queue until the mutex lock is available.

Releasing a mutex lock (syssgmx):

when a task releases a mutex lock, the system wakes up the task waiting for the lock and puts it into the ready state.

Wait Timer (syswttm):

When a task calls this function, the system hangs the current task and sets a timer. When the timer expires, the task is put back into the ready queue.

5. System Performance

Assuming an average execution time of 1 microsecond per instruction, timer interrupt period: 100 milliseconds, and task switching time: about 100 microseconds (100 instructions), each 10-millisecond time slice occupies about 1% of the task switching time, and the remaining 99% is used for executing user tasks.

I hope this manual will help you to better understand and use this real-time system. If you have any questions or need further support, please feel free to contact.

References

[1] Coleman, J. N. (2023, November 20). EEE8087. Course Materials. Course modules: Real Time Embedded Systems - EEE8087 (instructure.com)

[2] M. Jester. (2017, September, 16). *MarkeyJester's Motorola 68000 Beginner's Tutorial.* *MarkeyJester's 68k Tutorial (hapisan.com)*

[3] EASy68K. (2018, April, 07). *EASy68K Home Page.* *EASy68K Home*