DOCUMENTATION

This document describes the features and functionalities of a simple multitasking operating system implemented for mc68000 microprocessor, which is simulated in the easy68k simulator.

**Multitasking and Time Slicing Feature**

The main objective of this feature is to allow multiple tasks to run in parallel by sharing the processor time equally between the tasks.

The implementation of multitasking involves the creation of a list of tasks created by the user, which is maintained by the OS. This list contains the tasks that are ready to run, and the OS switches between tasks based on the timer interrupts. The following structure is used in designing the OS to implement these features.

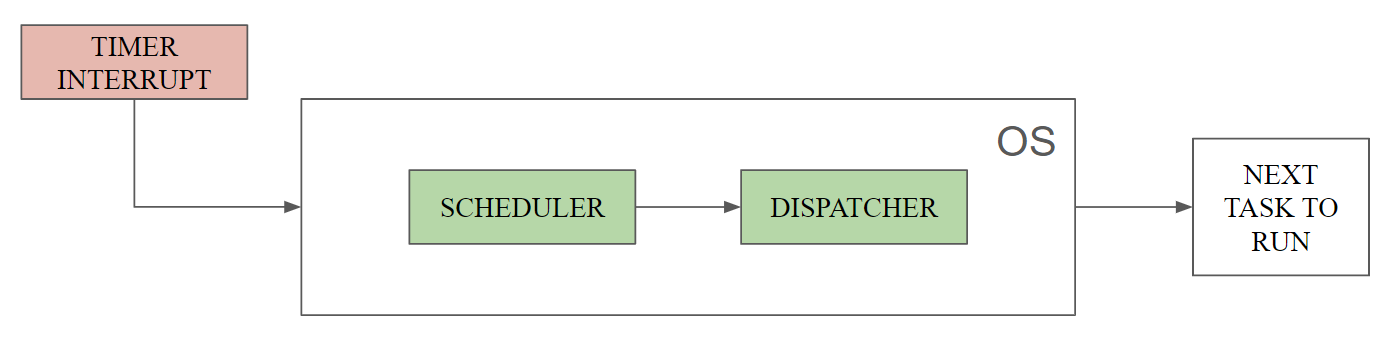


Fig 1: multitasking structure

The scheduler will choose the next task to run within the list of ‘ready to run’ tasks, finally the dispatcher will fetch all the necessary data related to the task chosen by the scheduler and load it to the registers and starts execution of that task.

**Memory Layout**

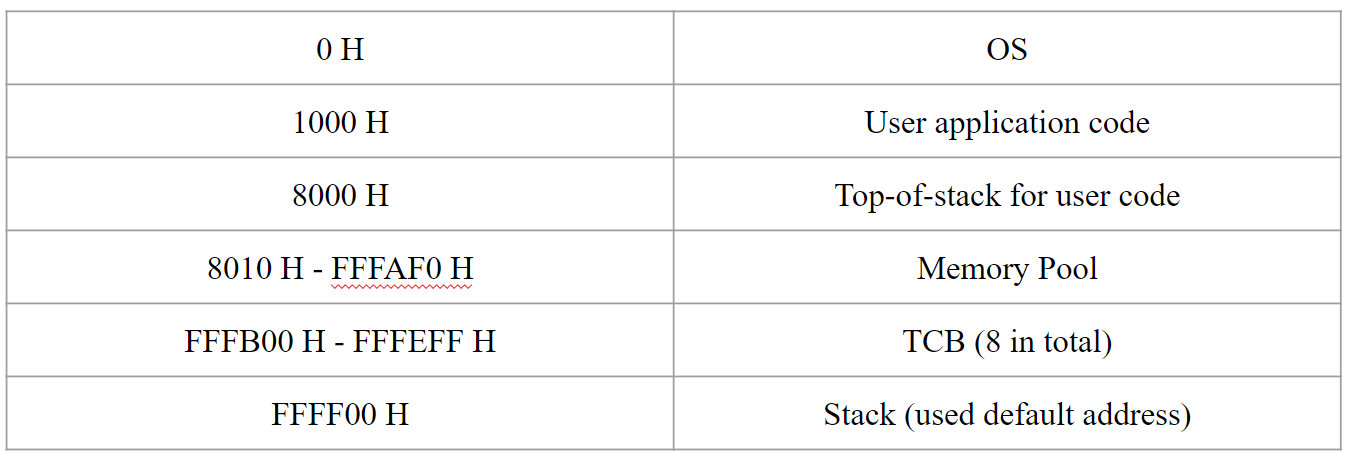


Fig 2: Memory Layout

The main operating system starts from the bottom of the memory, this runs in the foreground (when any interrupt occurs). The TCBs (Task Control Blocks) are at the very top, right below the default stack pointer. The free area in memory is used for memory allocation, mostly assigned to specific tasks upon request, for data storage.

**Startup Behaviour**

The following steps explain the startup behaviour;

1. initializing all the necessary variables to its default values.
2. All the TCBs, except for the first one, are marked as unused.
3. The beginning of the user code is considered as the default ‘task 0’, so the first TCB is initialized with values related to ‘task 0’.
4. Mutex is marked as available.

**User Functions**

These are specific functions, implemented in the operating system, that can be used for various purposes. These functions are described below;

**Create Task**

In our Designed system, the "Create Task" function initializes and manages new tasks. When a new task is created, it is added to a ready list, managed by the OS, using a task control block (TCB). An unused TCB is found and initialized with all the data necessary to run the new task, and it is linked to the current TCB in the ready list by altering its pointers (linked list algorithm).

Parameters: The start address of the new task, The address of its top-of-stack.

Usage:

A group of black text

Description automatically generated

* #syscr: represents the system call code to create a task.
* d0: function to run.
* d1, d2: parameter
* #$4000: is the task’s stack size.
* #sys: a software interrupt is raised to run the OS

This function executes in ’74 – 97’ instructions [including interrupt handler, scheduler and dispatcher].

**Delete task**

The objective of this function is to remove a task from the ready list, such that it will not execute any further. This is accomplished by unlinking the TCB within the ready list (changing the pointers) and marking the TCB as unused, so that any other task can utilize this TCB.

In our system, any task that should be deleted must send a request to the OS within that task. For instance, if ‘task 1’ needs to be deleted, then ‘task 1’ should send a request to the OS as shown below.

Parameters: none

Usage:

A group of black letters

Description automatically generated with medium confidence

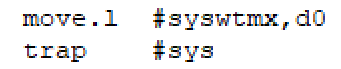
* #sysdel: represents the system call code to delete a task.
* d0: holds the value of id

This function executes in ’70 - 100’ instructions [including interrupt handler, scheduler and dispatcher].

**Wait Mutex Function**

The main objective of this function is to provide a way to use the shared resources effectively. This is achieved by maintaining a variable, internally by the OS, to hold the state of the availability of the shared resources. To use this function, the user must request the OS before accessing any kind of shared resource and the OS will check the status of the mutex and decide whether to let the task to run or hold its execution. Here, the OS maintains a separate list (waiting list) to keep track of the tasks on hold, any TCB in the waiting list is first unlinked from the ready list to hold its execution. There can be a case where multiple TCBs are present in the waiting list for various reasons, so we have given a specific code to the TCBs in the waiting list to identify the reason as to why the TCB is present in the waiting list. The ‘wait code’ used for TCB in case of ‘waiting for mutex’ is ‘C8H’ in our system.

Procedure to request to OS to run this function,



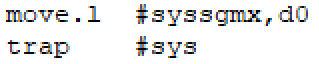
These lines, with the same syntax, should be included in the task to request the OS for mutex availability. Here, ‘d0’ holds the function that the user is requesting (as a specific integer value) and trap is the system call (software interrupt) that starts the OS.

This function executes in ’61 – 105’ instructions [including interrupt handler, scheduler and dispatcher].

**Signal Mutex Function**

The objective of this function is to make the mutex available for any further tasks. The OS maintains a separate list of tasks on hold (as mentioned earlier), and this list is examined by the ‘signal mutex function’ and if a task is present in this list waiting for mutex, it is put back on to the ‘ready to run tasks’ lists, this is verified by checking the ‘wait code’ for each TCB in the waiting list. To use this function, the task must send a request to the OS after the shared resource has been used. In a case where, the ‘holding tasks list is empty’ the mutex is directly made available for any further tasks to use, hence the shared resource is free to use by any other task.

Procedure to request the OS to run this function,



These lines must be included in the task to send a request to the OS. The syntax is same as shown in the previous task.

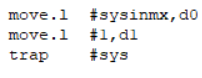
This function executes in ’61 – 105’ instructions [including interrupt handler, scheduler and dispatcher].

**Initialise Mutex**

In our designed system, “initialize Mutex” function is designed to set the mutex as specified by the user.

Parameter used: 0 or 1

Usage:



* #sysinmx: Represents the system call code to initialise mutex
* d0: Holds the system call code to initialise mutex
* d: Parameter
* trap #sys: It is a system call to the OS to handle the task

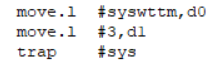
This function executes in ’60’ instructions [including interrupt handler, scheduler and dispatcher].

**Wait Timer**

In our designed system, “Wait Timer” function is designed to calculate and update the expiration time in the Task Control Block (TCB) based on the system counter (Counter) once a wait timer function is requested to OS. After that it removes the TCB from the ready list, adds it to the waiting list, and updates the waiting list length and the reason code for being in the waiting state. The task count in the waiting state is incremented, and the system scheduler is invoked for effective task management. The code periodically checks for tasks in the waiting list with expired timers and moves them back to the ready list. It adjusts the waiting list length, decrements the task count, and triggers the scheduler to manage the execution of tasks efficiently.

Parameter used: Number of time intervals to wait

Usage:



* #syswttm: Represents the system call code to wait on timer
* d0: Holds the system call code to wait on timer
* d1: Parameter
* trap #sys: It is a system call to the OS to handle the task

This function executes in ’81 - 97’ instructions [including interrupt handler, scheduler and dispatcher]. Later on, every time the timer interrupt occurs, we need to check the waiting list to know if any TCB is expired, hence this runs in ‘100-126’ instructions [our system is designed to handle multiple tasks in waiting list waiting for timer expiry].

**Wait I/O**

The main objective of this function is to put any requesting to the waiting list, as it is waiting for an I/O interrupt. Once an I/O interrupt occurs, this task is removed from the waiting list and added back to the ready list.

Parameters: None

Usage:



This line is used in the application program to send a request to the OS to put the task to waiting list.

**Additional System Features**

* interrupt priority handling: This system is designed to handle any interrupt that occurs first, none of the interrupts are given special priority.
* Processing time for tasks: Every task in the ready list receives equal amount of time to run, if there is a special need, by the user, to provide more execution time to any task, they should utilize the wait-timer function to accomplish this.
* Time to switch tasks: Our system takes ’83 instructions’ in total to switch between tasks, when a timer interrupt occurs.