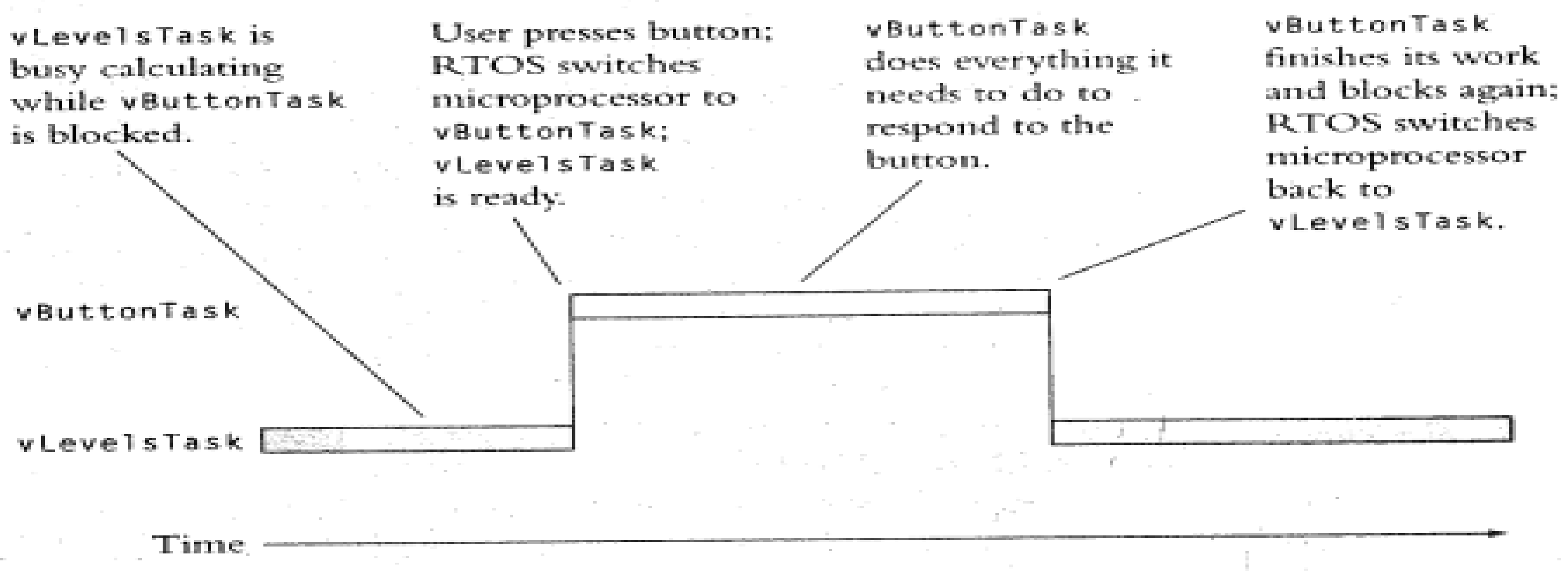
Real Time Operating System II

Task Priority Example

```
/*Button Task*/
void vButtonTask(void)          /*high priority*/
{
        while(True)
        {
                block until user pushes a button
                respond to the user
        }
}
/*Level Task*/
void vLevelTask(void)          /*low priority*/
{
        while(True)
        {
                Read levels of floats in the tank.
                Calculate average float level.
                Do some interminable calculations.
                figure out which tank to do next.
        }
}
```

Task Priority Example

- ❖ RTOS will stop the low-priority **vLevelsTask** in its track and move it to ready state.

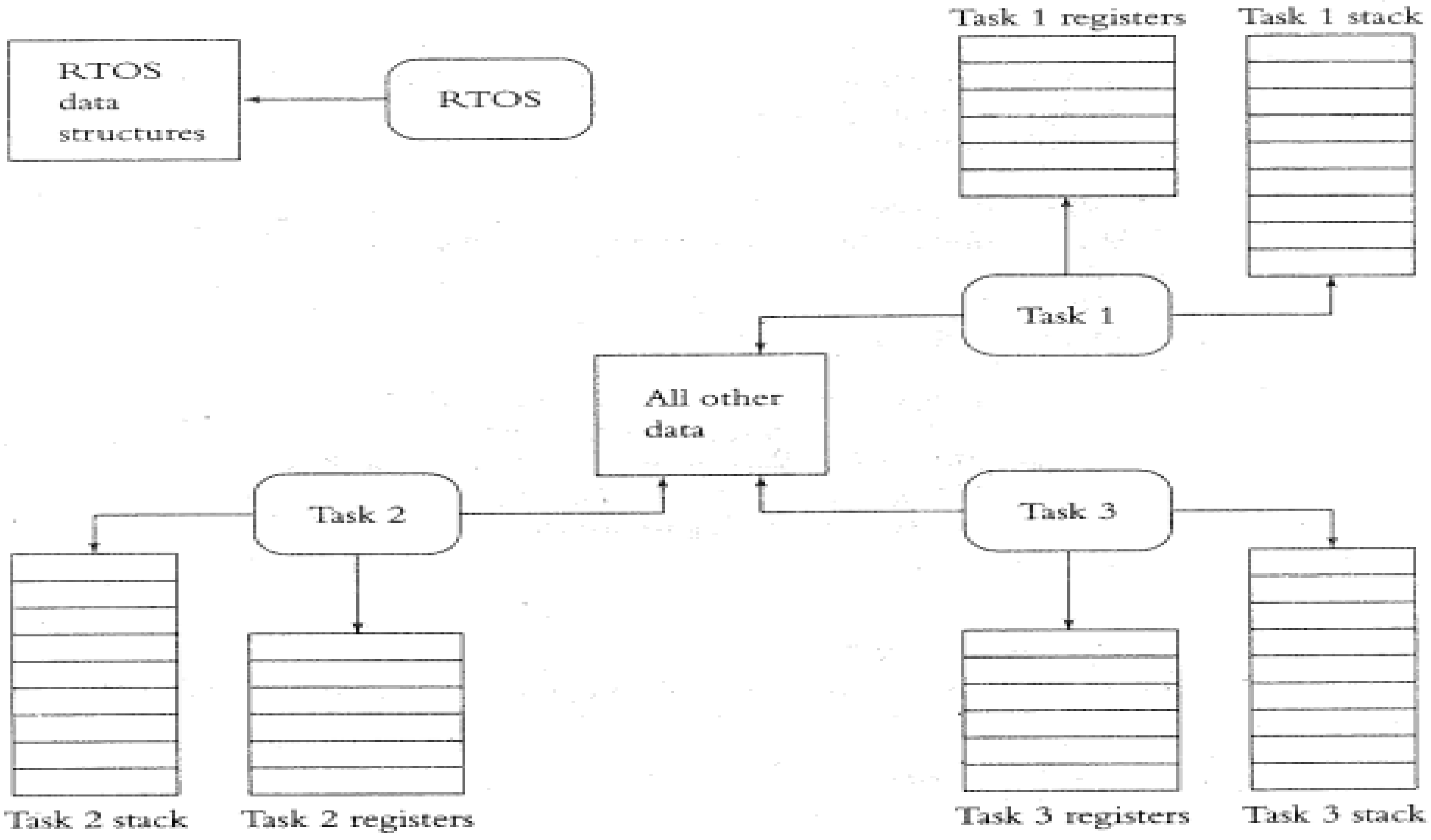

vLevelsTask is busy calculating while vButtonTask is blocked.
User presses button; RTOS switches microprocessor to vButtonTask; vLevelsTask is ready.
vButtonTask does everything it needs to do to respond to the button.
vButtonTask finishes its work and blocks again; RTOS switches microprocessor back to vLevelsTask.
vButtonTask
vLevelsTask
Time

- ❖ Runs the high priority **vButtonTask** task to let it respond to the user.
- ❖ **vButtonTask** task is finished responding, it blocks and the RTOS gives up back to the **vLevelsTask** task once again.

RTOS Initialization Code

```
void main(void)
{
InitRTOS();
StartTask(vButtonTask, HIGH_PRIORITY);
StartTask(vLevelTask,LOW_PRIORITY);
StartRTOS();
}
```

Task and Data



Shared Data problem

```
void Task1(void)
        {
                vCountErrors(9);
        }
void Task2(void)
        {
                vCountErrors(11);
        }
Static int cErrors;
void vCountErrors(int cNewErrors)
{
        cErrors+=cNewErrors;
}
```

Assembly Code

```
void vCountErrors(int  cNewErrors)
        {
                Mov R1, cErrors
                Add R1, cNewErrors
                Mov cErrors, R1
                return
        }
```

Shared Data Problem Contd…

```
Suppose cErrors=5
        Task T1 calls vCountErrors(9)
                Mov R1, cErrors                     ;R1=5
                Add R1, cNewErrors                  ;R1=9+5=14
         RTOS switches to Task T2

        Task T2 calls vCountErrors(11)              ;R1=5
                Mov R1, cErrors                     ;R1=5
                Add R1, cNewErrors                  ;R1=16
                Mov cErrors, R1                     ;cErrors=16

         RTOS switches to task T1
                Mov cErrors, R1                     ;cErrors = 14
```

Reentrancy

Functions that can be called by more than one task and will always work correctly even if the RTOS switches from one task to another in the middle of execution.

or

Function that can be interrupted at any time and resumed at a later time without loss of data.

It either uses local variables or protects data when global variables are used.

It can be used by more than one task without fear of data corruption.

Reentrancy

3 rules to decide if a function is reentrant:

- ❖ It uses all shared variables in an atomic way, unless they are stored on the stack of the task that called the function or are otherwise the private variables.
- ❖ It does not call non-entrant functions.
- ❖ It does not use the hardware in a non-atomic way.

Reentrancy

Suppose two functions each share the same variable foobar. One of the function contains the following:

```
temp=foobar;
temp+=1;
foobar=temp;
```

The code is reentrant because foobar is used non atomically.

Instead of using the above 3 equations, we use the following:

```
foobar+=1;
```

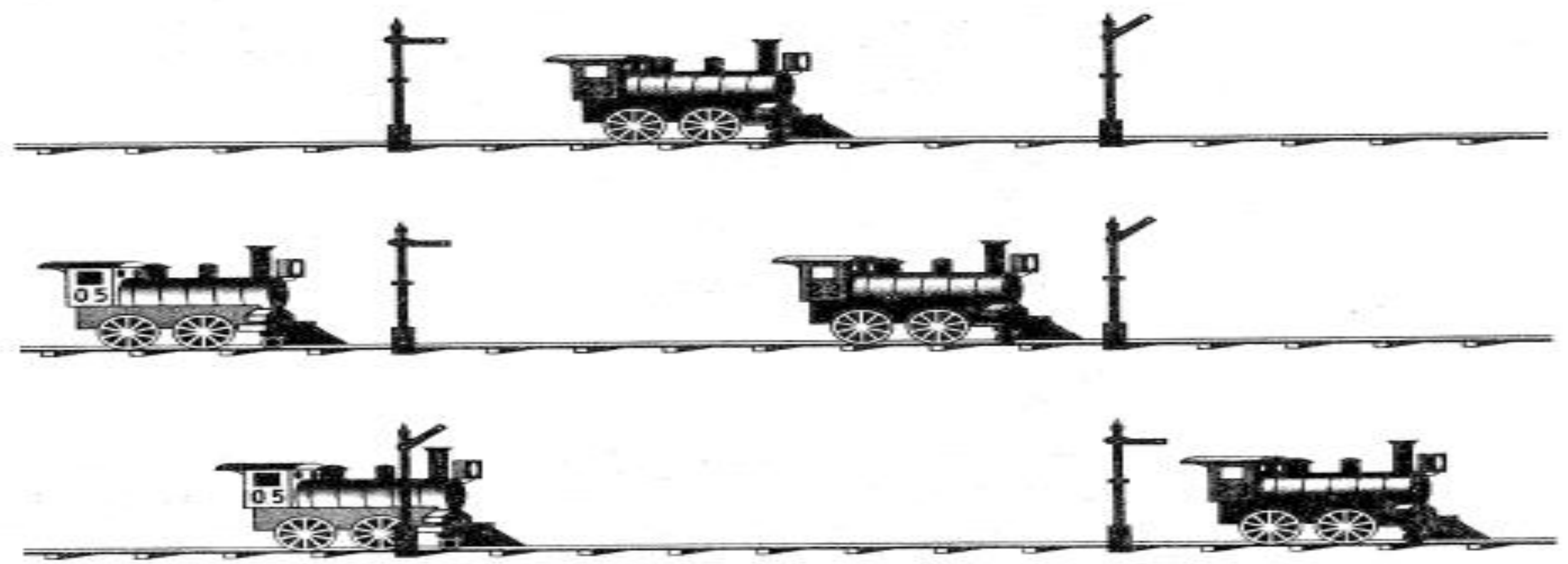
Assembly code:

```
mov ax, [foobar]
    inc ax
    mov [foobar], ax
```

Atomic Version: **inc[foobar]**

Semaphore

- RTOS, new tool to deal with shared data problems.
- Terms used in Semaphore:
  - get and give
  - take and release
  - pend and post
  - p and v
  - wait and signal
- RTOS idea is similar to that of a railroad semaphore.



Semaphore

- It was invented by Edgser Dijkstra in mid 1960s.
  - Control access to shared resources.
  - Signal the occurrence of an event.
  - Allow two tasks to synchronize their activties.
- Commonly used semaphore is called Binary Semaphore.
- Three operations can be performed on a semaphore which are Initialize, wait and signal.
- Two RTOS functions in Binary Semaphore:
  - **TakeSemaphore**
  - **ReleaseSemaphore**
- If a task calls **TakeSemaphore** to take semaphore and has not called **ReleaseSemaphore** to release it, then any other task that calls **TakeSemaphore** will block until the first task calls **ReleaseSemaphore**.
- Only one task can have a semaphore at a time.

Semaphore

Example of Take and Release Semaphore:

```
/*Button Task*/
void vButtonTask(void)          /*high priority*/
{
        while(True)
        {
                block until user pushes a button
                respond to the user
        }
}
/*Level Task*/
void vLevelTask(void)          /*low priority*/
{
        while(True)
        {
                TakeSemaphore();
                Read levels of floats in the tank.
                Calculate average float level.
                ReleaseSemaphore();
                Do some interminable calculations.
                figure out which tank to do next.
        }
}
```

Semaphore

Execution Flow with Semaphores:

Code in Level Task

Level task is calculating tank levels.

TakeSemaphore();
Read tank level.
Calculate average.

Code in Button Task

Button task is blocked waiting for a button.

User pushes a button; the high priority button task unblocks RTOS switches

TakeSemaphore();
This does not return yet.

Semaphore is not available button task blocks RTOS switches back

ReleaseSemaphore();

Releasing semaphore unblocks button task RTOS switches again.

(Now TakeSemaphore returns)
ReleaseSemaphore();
Block unti user pushes a button

The button task blocks, RTOS resumes the level task.

Semaphore

Semaphores makes a function reentrant.

```
void Task1(void)
        {
                vCountErrors(9);
        }
void Task2(void)
        {
                vCountErrors(11);
        }
Static int cErrors;
void vCountErrors(int cNewErrors)
{
        TakeSemaphore();
        cErrors+=cNewErrors;
        ReleaseSemaphore();
}
```