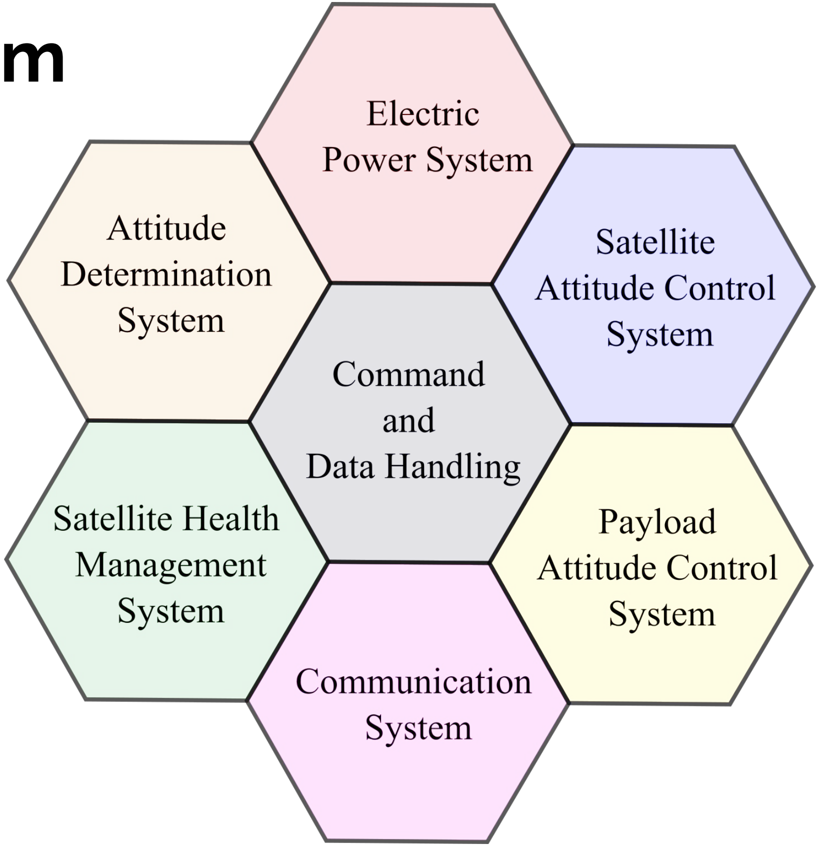
RODOS:
An Introduction

Rishav Mani Sharma

May 29, 2021

Satellite System



***Source*:** Sarrio et al. - Flexible Electrical Power System for Interplanetary and Lunar Cubesats (2018)

Embedded Systems

An embedded system is a computer system—a combination of a computer processor, computer memory, and input/output peripheral devices—that has a dedicated function within a larger mechanical or electronic system.

Examples:

Food processing	Satellite onboard computer
Chemical plants	Jet engine controls
Engine controls	Microwave ovens
Photocopiers	Washing machines
Modems	Thermostat
Robots	Weapons systems

Super Loop: Without RTOS

```
// Your initializations
void setup()
{
    initIron();
}

// Function your device performs
void loop()
{
    readCurrentTemp();
    runControlLaw();
    heatIron();

    certainDelay();
}

// Interrupt Service Routine
// 1. Iron going crazy !!
// 2. Change setpoint
ISR()
{
}
```

Fig. Generic heater control system without RTOS

Super Loop in Satellite

```
electricalPowerSystem()
{
    // EPS functionality
}

attitudeEstimation()
{
    // Attitude estimation functaionality
}

communicationSystem()
{
    // Communication system functionality
}

attitudeControl()
{
    // Attitude control functionality
}

commandAndDataHandling()
{
    // Command & data handling functionality
}
```

Fig. Satellite subsystem software implementations skeleton

```

void main()
{
    while (1)
    {
        commandAndDataHandling(); // 500 millis
        electricalPowerSystem();  // 5 minute
        attitudeEstimation();     // 10 millis
        attitudeControl();        // 500 millis
        communicationSystem();    // 2 seconds
    }
}
```

Fig. main()

Tasks

```
electricalPowerSystem()
{
    while (1)
    {
        // EPS functionality
    }
}

attitudeEstimation()
{
    while (1)
    {
        // Attitude estimation functaionality
    }
}

communicationSystem()
{
    while (1)
    {
        // Communication system functionality
    }
}
```

Fig. Satellite subsystem software implementations skeleton

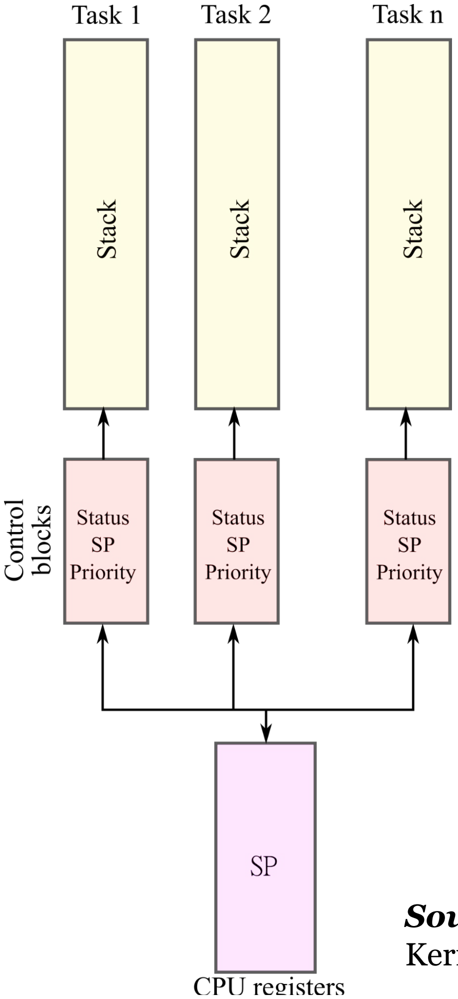
```
void main()
{
    createTask1(&electricalPowerSystem);
    createTask2(&attitudeEstimation);
    createTask3(&communicationSystem);
    createTask4(&attitudeControl);
    createTask5(&commandAndDataHandling);
    startScheduler();
}
```

Fig. main()

Tasks

1. A task, also called a *thread*, is a simple program that thinks it has the CPU all to itself.
2. The design process for a real-time application involves *splitting the work to be done into tasks* which are responsible for a portion of the problem.
3. Receives its *own private memory stack*.
4. Unlike a super loop in main, which was sharing the *system stack*.
5. Each task has its own call stack *without interfering with other tasks*.
6. Each task has a *priority assigned* to it.
7. This priority allows the *scheduler* to make decisions on which task should be running.
8. Like *multiple little* embedded processors with the processor.

Tasks:
A closer look



***Source*:** Labrosse - MicroC OS II_ The Real Time Kernel (2002)

Super loops vs task

```c
void main()
{
    while (1)
    {
        commandAndDataHandling(); // 500 millis
        electricalPowerSystem();  // 5 minute
        attitudeEstimation();     // 10 millis
        attitudeControl();        // 500 millis
        communicationSystem();    // 2 seconds
    }
}
```

Fig. Super loop

```c
void main()
{
    createTask1(&electricalPowerSystem);
    createTask2(&attitudeEstimation);
    createTask3(&communicationSystem);
    createTask4(&attitudeControl);
    createTask5(&commandAndDataHandling);
    startScheduler();
}
```

Fig. Tasks

THE CRUX OF THE PROBLEM:
HOW TO PROVIDE THE ILLUSION OF MANY CPUS?

Although there are only a few physical CPUs available, how can the OS provide the illusion of a nearly-endless supply of said CPUs?

TIP: USE TIME SHARING (AND SPACE SHARING)

Time sharing is a basic technique used by an OS to share a resource. By allowing the resource to be used for a little while by one entity, and then a little while by another, and so forth, the resource in question (e.g., the CPU, or a network link) can be shared by many. The counterpart of time sharing is **space sharing**, where a resource is divided (in space) among those who wish to use it. For example, disk space is naturally a space-shared resource; once a block is assigned to a file, it is normally not assigned to another file until the user deletes the original file.

Source: Operating Systems: Three Easy Pieces

Real Time Operating System

An RTOS is an operating system in which the time taken to process an input stimulus is less than the time lapsed until the next input stimulus of the same type.

Functions:

1. Scheduling tasks - Preemptive*
2. Intertask communication and resource sharing
3. Interrupt handlers
4. Memory allocation

**Preemption* is the act of temporarily interrupting an executing task, with the intention of resuming it at a later time.

RTOS because …

1. Gives the parallel processing-ish result
2. Abstracting Timing Information
3. Priority Based Scheduling
4. Modularity
5. Promotes Team Development
6. Easier Testing
7. Code Reuse
8. Idle Processing

Source: https://www.highintegritysystems.com/

Realtime Onboard Dependable Operating System

RODOS

Real Time Kernel Design for Dependability

Dr. Sergio Montenegro, Frank Dannemann
DLR, Institute of Space Systems, Robert-Hooke-Str. 7, 28359 Bremen, Germany
Web: http://www.dlr.de/irs
E-Mail: Sergio.Montenegro@DLR.de, Frank.Dannemann@DLR.de

Introduction

The NetworkCentric core avionics machine consists of several harmonised components which work together to implement dependable computing in a simple way.

Repository: https://gitlab.com/rodos/rodos

RODOS

Features

1. Hardware abstraction
2. Timing abstraction
3. Preemptive dynamic schedule

Philosophy

1. Dependability*
2. Simplicity

* *Dependability* is the ability to provide services that can defensibly be trusted within a time-period.

Hello World !!

ध्यान दिनुपर्ने कुराहरु !!

```cpp
// The class Thread is extended by a custom run() procedure, which
// writes Hello World to the UART with PRINTF().

#include "rodos.h"

class HelloWorld : public StaticThread<>
{
    void run()
    {
        PRINTF("Hello World!\n");
    }
} hello_world;
```

Threads (Tasks)

```cpp
// The thread LowPriorityThread writes the character "." every second
// and is interrupted every five second by the thread HighPriorityThread,
// which writes the character "*".

#include <rodos.h>

class HighPriorityThread : public StaticThread<>
{
public:
    HighPriorityThread() : StaticThread("HiPriority", 55) {}
    void run()
    {
        while (1)
        {
            suspendCallerUntil(NOW() + 5 * SECONDS);
            PRINTF("*\n");
        }
    }
} highprio;

class LowPriorityThread : public StaticThread<>
{
public:
    LowPriorityThread() : StaticThread("LowPriority", 25) {}
    void run()
    {
        while (1)
        {
            suspendCallerUntil(NOW() + 1 * SECONDS);
            PRINTF(".\n");
        }
    }
} lowprio;
```

Communication between threads

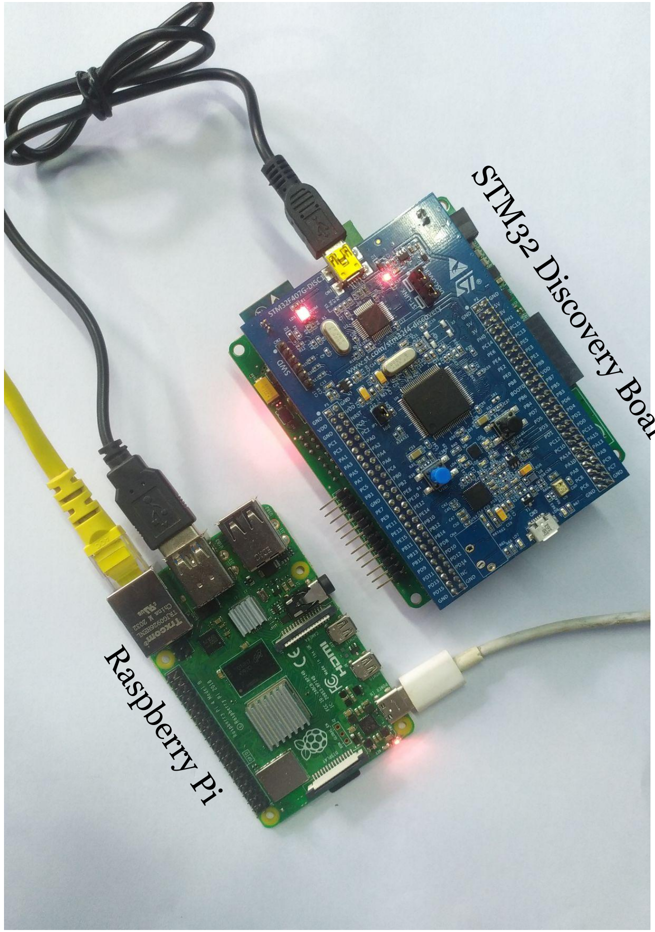
```cpp
// The Publisher-Thread post every one second an ascending counter value,
// while the Subscriber-Thread simply displays the received integer value.

#include <rodos.h>

Topic<long> counter1(-1, "counter1");

class MyPublisher : public StaticThread<>
{
public:
    MyPublisher() : StaticThread("SenderSimple") {}
    void run()
    {
        long cnt = 0;
        TIME_LOOP(1 * SECONDS, 1 * SECONDS)
        {
            PRINTF("Published: %ld\n", ++cnt);
            counter1.publish(cnt);
        }
    }
} publisher;

class MySubscriber : public SubscriberReceiver<long>
{
public:
    MySubscriber() : SubscriberReceiver<long>(counter1) {}
    void put(long &data)
    {
        PRINTF("Received: %ld\n\n", data);
    }
} subscriber;

```

Remote access to OBC

Your computer

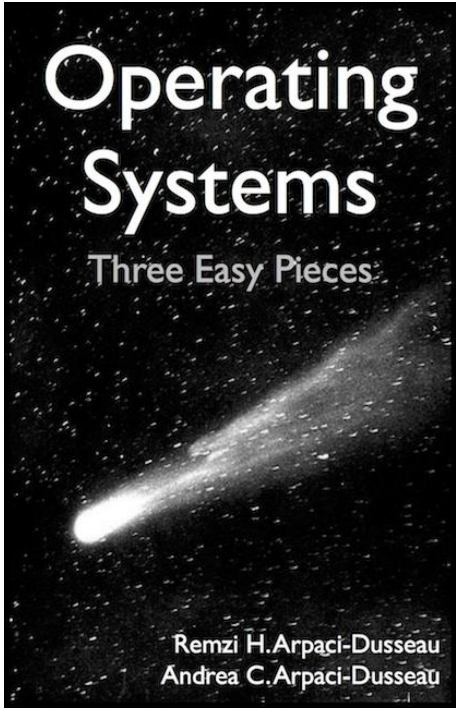
Internet

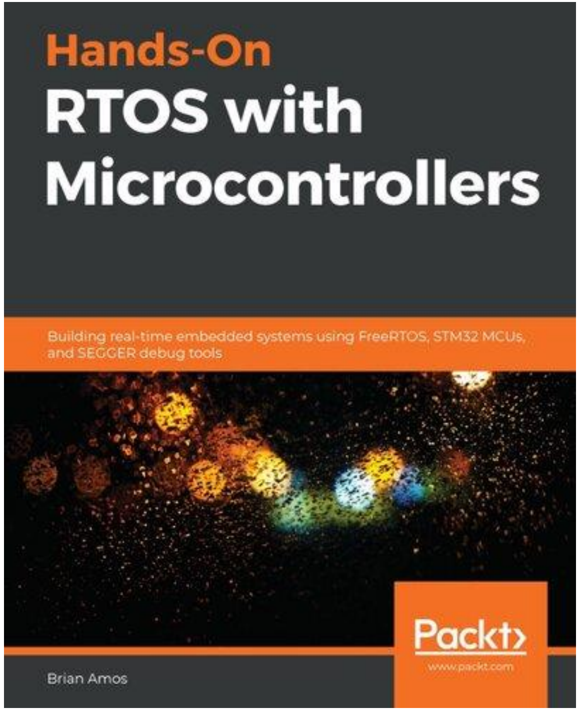
SSH using PuTTy

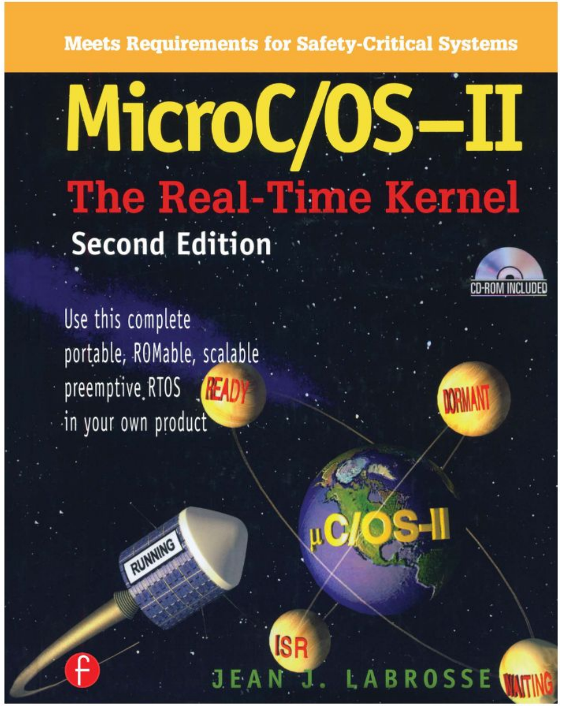


My home

References


Operating Systems
Three Easy Pieces
Remzi H.Arpaci-Dusseau
Andrea C.Arpaci-Dusseau


Hands-On
RTOS with Microcontrollers
Building real-time embedded systems using FreeRTOS, STM32 MCUs, and SEGGER debug tools
Packt>
www.packt.com
Brian Amos


Meets Requirements for Safety-Critical Systems
MicroC/OS-II
The Real-Time Kernel
Second Edition
CD-ROM INCLUDED
Use this complete portable, ROMable, scalable preemptive RTOS in your own product
READY
DORMANT
μC/OS-II
RUNNING
ISR
WAITING
JEAN J. LABROSSE

Thank You !!