STR: aula_5

Trabalho 1 – 2ª parte

RTLib – A realtime kernel library

Planeamento das aulas

- **Semana 21 Set:**
  - **Conceitos básicos de I/O (com experimentação)**
- **Semanas 28 Set, (5 Out), 12 Out:**
  - **1ª parte do trabalho 1 (em C)**
- **Semanas 19 Out, 26 Out:**
  - **2ª parte do trabalho 1 (RTlib)**  **HERE**
- **Semana 2 Nov:**
  - **Exercícios sobre Java**
- **Semanas 9, 16, 23 Nov:**
  - **Trabalho 2: Java em Tempo Real**
- **Semanas 30 Nov, 7 Dez, 14 Dez:**
  - **Trabalho 3 (plc)**

RtLib – Realtime kernel library (review)

Overview

- Allow run programs in a real-time fashion.
- Programs can concurrently sense, react, and perform actions over the system being managed.
- Tasks inside a program can compete for a resource, coordinate, cooperate, communicate and synchronize.

Basic functionality

- Concurrent tasks creation
- Synchronization with SEMAPHORES
- Message Sending/receiving with MAILBOXES
- Works in a cooperative fashion – So, every task should allow other tasks to run by invoking Rtdelay(int miliseconds), or any synchronization or mailbox functions

Example: Simple task (review)

```
#include"c:\rtlib\lib\kernel.h"
#include<stdio.h>
#include <stdlib.h>
#include <conio.h>

RTtask(Task1) {
   printf("\nTask1: Hello...");
   while(1)   {
      int r = 100.0*rand()/RAND_MAX;
      printf("\n\t\tTask1 working..... %02x",r);
      RTdelay(500);
   }
}

int main(int argc, char **argv) {
   InitRTkernel();
   RTcreateTask(Task1,2,"Task1");
   int t;
   do{
         t=RTgetkey();
   } while(t!=27);
   return 0;
}
```

Tasks run indefinidely until main function finishes (see below)

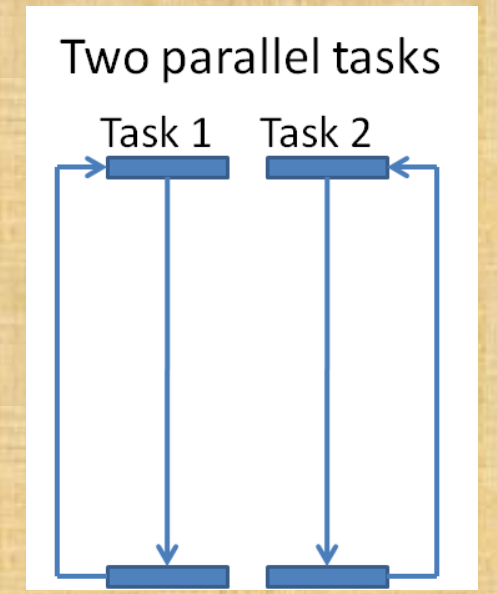
Kernel initialization & Task creation

Goes through a infinite look that if interrupted, finalizes the program

Try it here….

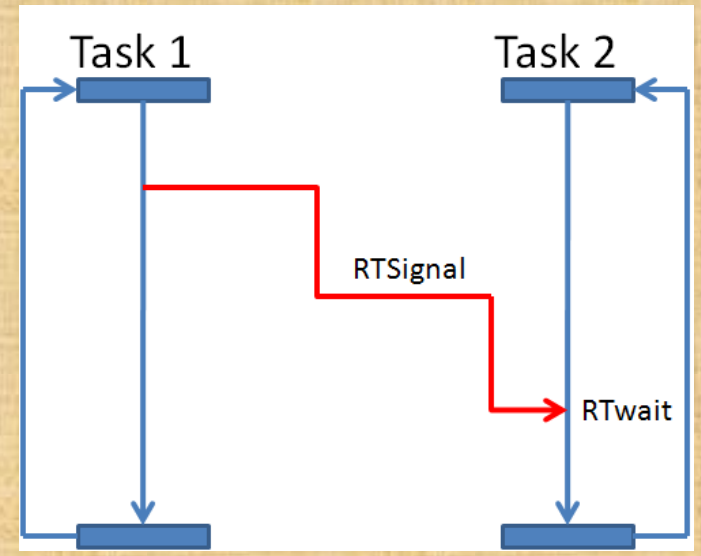
C:\rtlib\demos\vc++6\simple_task

Provided examples



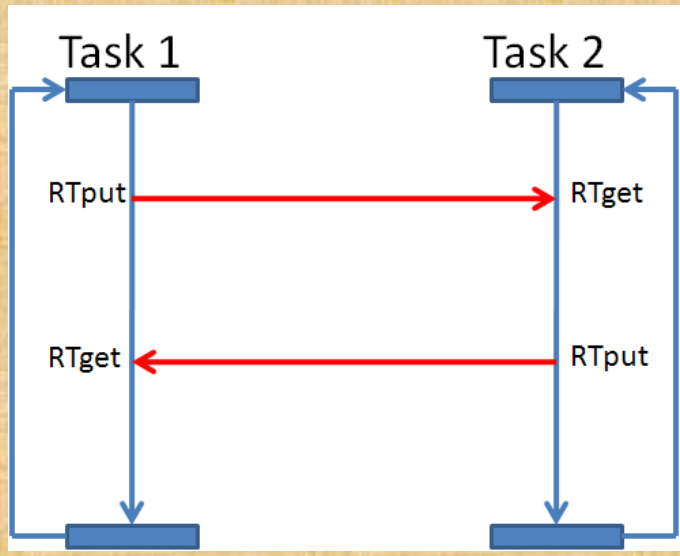
C:\rtlib\demos\vc++6\parallel_tasks

synchronized



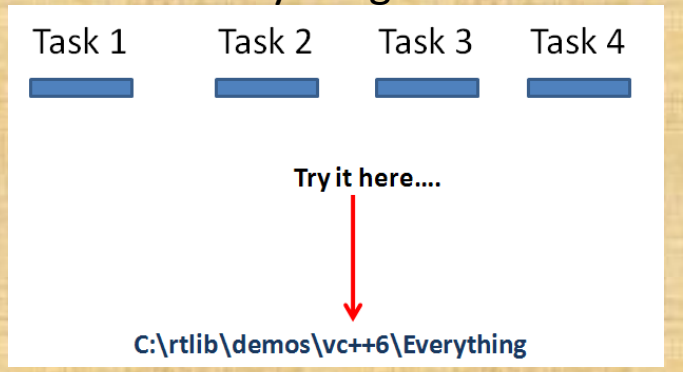
C:\rtlib\demos\vc++6\Sinchronization

MAILBOXES



C:\rtlib\demos\vc++6\MailBox

everything



Approach to add real-time functionality

- Many functions are just execute single access to hardware -> no changes.
- But functions with while(){ } may need to be translated into Tasks.
- Example next…

An example

```
typedef struct
{
  // 0->put_piece, 1->get_piece, ...
  int operation;
  int x;
  int z;
  // 0->operation_ok, 1->busy, 2->error
  int status;
}TRequest;
```

```
RTtask(Task_conveyor)
{
  printf("\n===>Task4: Hello...");
  printf("\n===>Task4: Waiting for Message from Mailbox
M4...");
  TRequest req;
  while(1)
  {
    req=*(TRequest *)RTget(M1);
    printf("\n\n===>Task4: EXECUTING request: op=%d x=%d
              z=%d...“,  eq.operation, req.x,req.z);
    req.status = 10*rand()/RAND_MAX;
                                RTput(M4,&req);
  }
}
```

```
RTtask(Task_keyboard)
{
   printf("\nTask1: Hello...");
   int idx = 0;
    RTdelay(1000); // let other tasks start...
    char str[255];
   while(1)
   {
       TRequest req;
        printf("\n x="); gets(str);req.x=atoi(str);
        printf("\n z="); gets(str);req.z=atoi(str);
       printf("\n op="); gets(str);req.operation=atoi(str);
       if(req.x==9 || req.z==9 || req.operation==9)
            terminate_prog=true;

      RTput(M1,&req);
      RTdelay(1000);
      req=*(TRequest *)RTget(M4);
      printf("\n===>Task1: OPERATION status %d...",req.status);
   }
}
```

```
int main(int argc, char **argv)
{
   InitRTkernel();
   M1=RTcreateMailbox(10);
   M4=RTcreateMailbox(10);
   RTcreateTask(Task_keyboard,2,"Task1");
   RTcreateTask(Task_conveyor,2,"Task4");
  //prevent ending the program
   while(!terminate_prog)
        RTdelay(10);
   return 0;
}
```