Device Drivers
(Build your own device driver and Robot control through RT-COM)

RTOS Support

- NOT necessarily responsible everything
- RTOS system calls

Interrupt Management
rtl_request_irq
rtl_free_irq
rtl_hard_enable_irq
rtl_hard_disable_irq

Time Management
clock_gethrtime
clock_gettime
clock_settime
gethrtime
nanosleep

Task Management
pthread_create
pthread_setschedparam// pri. sched
pthread_make_periodic_np
pthread_wait_np
pthread_delete_np
pthread_cancel
pthread_join

Task Communication
FIFO
Shared Memory
Signal

Mutual Exclusion
Lock
Semaphore

Device drivers
rt_com
rtsock

Project A
(Device Driver Designer's Perspective)

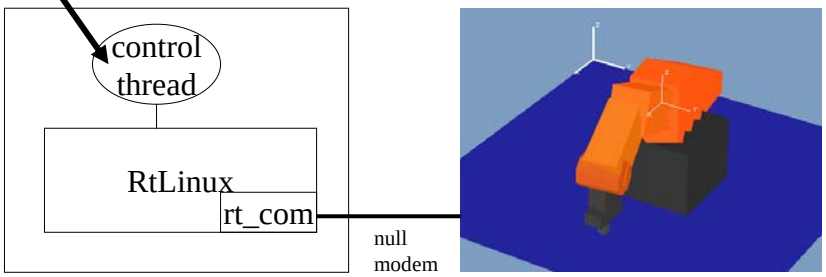
- Build your own rt_com driver called my_rt_com
  - Start with existing sources in your "rtlinux/drivers/rt_com/" directory.
  - Only change you have to make is to implement timeout based read mechanism.
  - Change "rt_com_read" prototype to
    - int rt_com_read(unsigned int com, char *ptr, int cnt, hrtime_t timeout);
  - Timeout works as follows
    - If timeout = 0, it works exactly same as the original rt_com_read, that is, immediately return with the current data in rt_com ibuf.
    - If timeout < 0, it works as a blocked Read, that is, the thread that calls this rt_com_read should suspend until "cnt" bytes are available to return.
    - If timeout > 0, it wait until either "cnt" bytes are available to return or "timeout" happens. If "timeout" happens before "cnt" bytes are available, it returns the number of bytes actually read at that time point.
- Note:
  - For simplicity, assume that only one thread uses the rt_com driver.
  - For suspending and wakeup a thread, we can use pthread_cond_wait, pthread_cond_timedwait, pthread_cond_signal along with mutex if necessary.

Device Driver User
(Control Robot through serial)

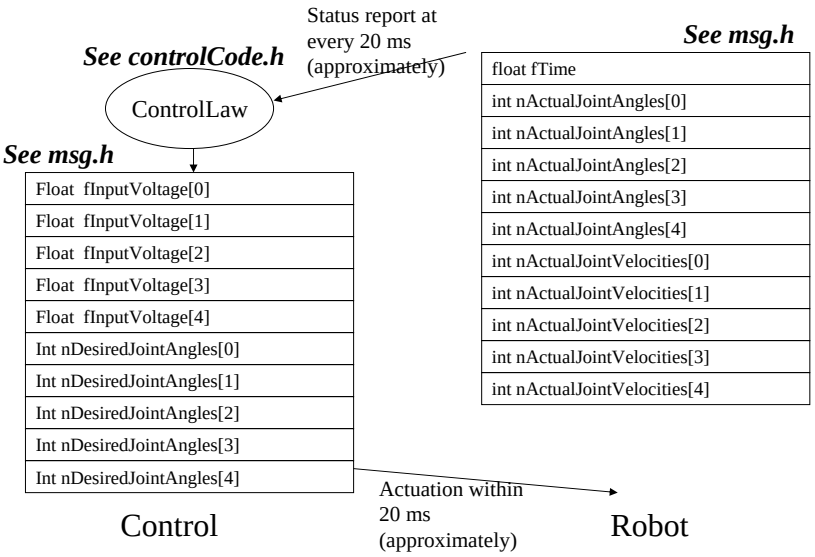
- Only needs to know three standard interface functions
  - rt_com_setup: configure serial com parameters
    - int rt_com_setup(unsigned int com, unsigned int baud, usigned int parity, unsigned int stopbits, unsigned int databits);
  - rt_com read: read data received from COM
    - int rt_com_read(com, char *ptr, byteCountToBeRead);
  - rt_com write: write data to be transmitted to COM
    - void rt_com_write(com, char *ptr, byteCountToBeWritten);

Overview of Robot Control Thread

You have to do this (use template files from “prj4_robotControlSample.tar”)



Message Format



rt_com functions (1)

- Configuration
  - #include <rt_com.h>
  - int rt_com_setup(unsigned int com, unsigned int baud, usigned int parity, unsigned int stopbits, unsigned int databits);
    - Baud rate: 115200
    - No parity
    - Stop bit: 1
    - Data bits: 8
- In cleanup_module: release rt_com
  - rt_com_setup(0, -1, 0, 0, 0);

rt_com functions (2)

- read
  - #include <rt_com.h>
  - int rt_com_read(unsigned int com, char *ptr, int cnt);
    - Try to read cnt bytes
    - Return no. of bytes actually read (Non-blocking I/O)
- write
  - #include <rt_com.h>
  - int rt_com_write(unsigned int com, char *ptr, int cnt);
    - Write cnt bytes

Rt thread

- Periodical sample and actuation
  - At each iteration (each job)
    - Read status message
    - If status message is ready
      - Call ComputeInputVoltage();
      - Write actuation message
- Note: be careful in managing input buffer

One more thing

- For allowing floating point operations
  - #include <rtl_sched.h>
  - Int pthread_setfp_np(pthread_t thread, int flag);
    - thread = pthread_self()
    - If flag = 1, enable FP operations

Download RobotBuilder

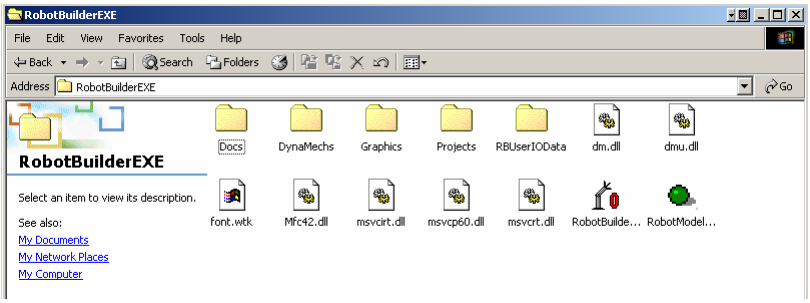
- From ETL, download “RobotBuilder.zip”.
- Unzip “RobotBuilder.zip”.
- No installation needed.
- From ETL, download and unzip “Lab4RobotWindowSide.zip”.

Running RtLinux control thread

- Run RtLinux control thread first
  - If rt_com has been already inserted, remove it
  - Disable Linux com driver
    - Prompt> setserial /dev/ttyS0 uart none
    - Prompt> setserial /dev/ttyS1 uart none (if needed)
  - Insert rt_com
    - Prompt> insmod rt_com.o
  - Insert your controlThread
    - Prompt> insmod robotControl.o (cgleeRobotControl.o is given for your viewing the expected robot motion)

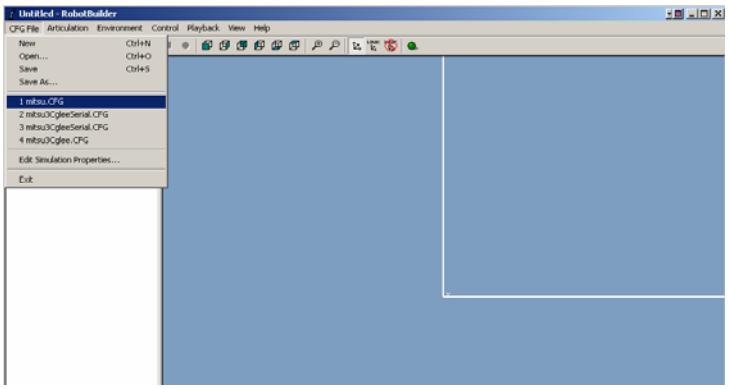
Running RobotBuilder (1)

- Simply run the executable “RobotBuilder.exe”
  - For real-time motion of robot, Pentium 4 with 1.2 GHz and higher is desired.
  - Pentium 3 with 600 MHz seems OK.
  - Pentium 2 with 300 MHz may work but robot motion may be very slow)



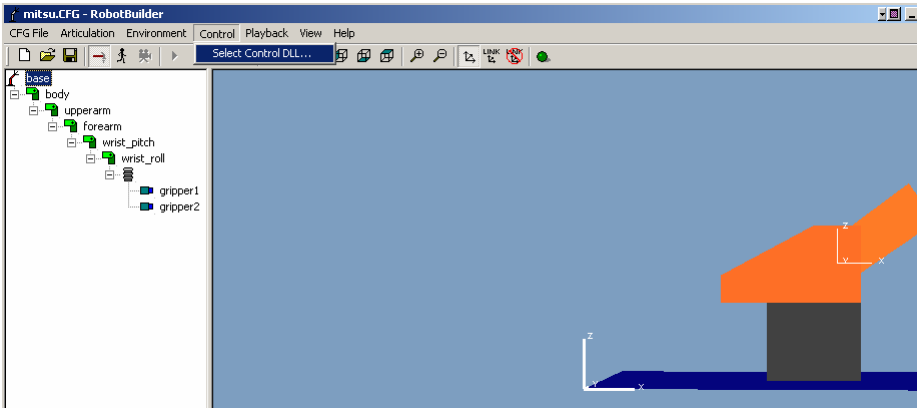
Running RobotBuilder (2)

- Choose CFG file (mitsu.CFG) from directory of “Lab4RobotWindowSide”



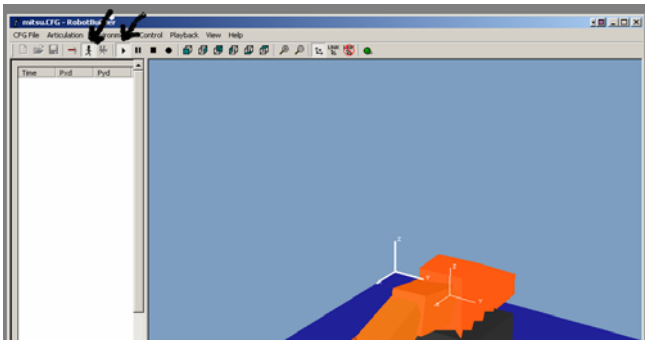
Running RobotBuilder (3)

- Choose dll file from directory of “Lab4RobotWindowSide”
  - mitsuCOM1.dll for COM1 (default)
  - mitsuCOM2.dll for COM2



Running RobotBuilder (3)

- Play
  - Click the “running person” button
  - Click the “play” button



Project B
(Device Driver User's Perspective)

- Implement your control thread (use template in prj4_robotControlSample.tar).
- Measure times necessary for schedulability analysis. Perform the schedulability analysis with the measured times.
- Increase and decrease the sampling period and find
  - The longest possible sampling period (the maximum period is bounded by the stability of control of physical environment)
  - The shortest possible sampling period (the minimum possible period is bounded by rt_linux time granularity, scheduling overheads, etc.)
- While robot is running, run web-browsers, MPEG Players, etc. Explain what happens and why? Then, remove (rmmod) your control module from rt-linux. Explain what happens and why?