Open Source RTOS

Featuring FreeRTOS and eCos

Powering the Supply Chain.SM

(RT)OS for ARM

- Many open source alternatives available today
  - Linux, eCos, CapROS, FreeRTOS, RTEMS etc...
- Parameters when selecting one can be
  - Available packages/APIs (like network stacks, POSIX interfaces etc)
  - Size/Performance
  - Portability
  - Price, licensing
    - Many open source RTOSes are LGPL

Why (RT)OS?

- Separation of 'firmware' and application
  - Applications should be platform independent and use standard (and open) APIs
- Requirement for more advanced functions such as filesystems, network stacks
- Threading / heap management support

Why not RTOS?

- Size penalty
  - Depending on RTOS of choice and configuration
  - Ranging from about 10KiB up to MiB
- Performance penalty
  - Most OSes uses a tick interrupt with 10-100ms resolution
  - Interrupt handling is more cumbersome (context switch etc)
  - Most RTOS have generic ports and doesn't use “special” interrupt controllers

FreeRTOS

- FreeRTOS is small and simple. The kernel itself is comprised of only three or four C files
- Preemptive, heap management
- Open source under LGPL, no source tainting
- Many ports, including bitfire + atmosfire (made by yours truly)

FreeRTOS

- Arm (32bit) size: about 8KiB
- Well worth a look for small “scheduler” like requirements
- Easy to understand and port
- Heap management is a plus

freertos421: .../armschool/software/rtos/FreeRTOS/Demo/ARM7_STR71x_GCC/main.c – Kate

File Edit Document View Bookmarks Tools Sessions Settings Window Help

Documents

Filesystem Browser

- port.c
- portISR.c
- portmacro.h
- 71x_conf.h
- 71x_vect_gnu.s
- FreeRTOSConfig.h
- main.c
- Makefile
- minrtos.c
- minrtos.h

```c
void main( void )
{
  int i;
  char ch='0';
  char buf[6]="TaskX";

  /* Setup any hardware that has not already been configured by the low
    level init routines. */
    prvSetupHardware();

        for (i=0;i<5;i++) {
          buf[4]=ch;
          xTaskCreate(taskcode, (const signed char*)buf, configMINIMAL_STACK_SIZE, NULL,
          tskIDLE_PRIORITY+1, NULL);
          ch++;
        }

        vSemaphoreCreateBinary(uart);
        vTaskStartScheduler();

    /* We should never get here as control is now taken by the scheduler. */
    return;
}

static void prvSetupHardware( void )
{

extern volatile char *pxCurrentTCB; // Internal kernel TCB pointer
extern void vPortEnterCritical( void );
extern void vPortExitCritical( void );

void print(char *str)
{

void taskcode(void *pvParameters)
{
  int i=0;
  char *buf=(char*)(pxCurrentTCB+56); // Taskname is at this offset

  while(1){
    i++;
    if (xSemaphoreTake(uart,(portTickType)0)==pdTRUE) {
      print(buf);
      print("\r\n");
      xSemaphoreGive(uart);
    }
    vTaskDelay(100+(rand()%500));
```

Line: 39 Col: 1 | INS | NORM | .../armschool/software/rtos/FreeRTOS/Demo/ARM7_STR71x_GCC/main.c

Find in Files | Terminal

eCos

- Open source, royalty-free, real-time operating system intended for embedded applications
- Originally developed by Cygnus Software, later Redhat. Now maintained by eCoscentric in UK
- Much more than a simple scheduler, plays more in the “embedded linux” field
- Support for virtual memory

eCos

- Complete solution with bootloader (redboot), debug solutions and RTOS
- Highly configurable at source level with powerful tools
- Behaves like a standard POSIX (unix) system, but with it's own codebase for c lib etc
- A plethora of packages to choose from
- Runs on almost everything, including PCs!

eCos, basic concepts

- Every building block is called a package
- Component repository
  - Contains all packages with source and tests
- Build tree
  - Contains a eCos system configuration and build files of a specific “libtarget.a”
- Application tree
  - Contains you application sources

eCos, basic concepts

- HAL (Hardware Abstraction Layer)
  - "BSP" package containing the core port to the specific board
- Kernel
- Other drivers typically placed under devs/ in component repository
- Many many more packages for file systems, network stack etc etc.

eCos configtool

ecos - eCos Configuration Tool

File Edit View Build Tools Help

- Configuration
  - Redboot for ARM options
  - Global build options
  - Redboot HAL options
  - FLASH device drivers — current
    - Hardware cannot support direct acces
    - Display status messages during flash
    - Verify data programmed to flash
    - Platform has flash soft DIP switch v
    - Instantiate in I/O block device API
    - Flash device driver build options
    - m25pxx SPI FLASH memory support — current
  - eCos HAL — current
  - I/O sub-system — current
  - Infrastructure — current
  - eCos kernel — current
  - Dynamic memory allocation — current
  - ISO C and POSIX infrastructure — current
  - ISO C library — current
  - Math library — current
  - Wallclock device — current
  - Common error code support — current
  - MTFS Filesystem — current
    - Use a RAM emulated flash device
    - Use IO_FLASH system for interfacing
    - Use M25PXX flash device driver
  - LUA programming language — current
  - POSIX File IO compatibility layer — current

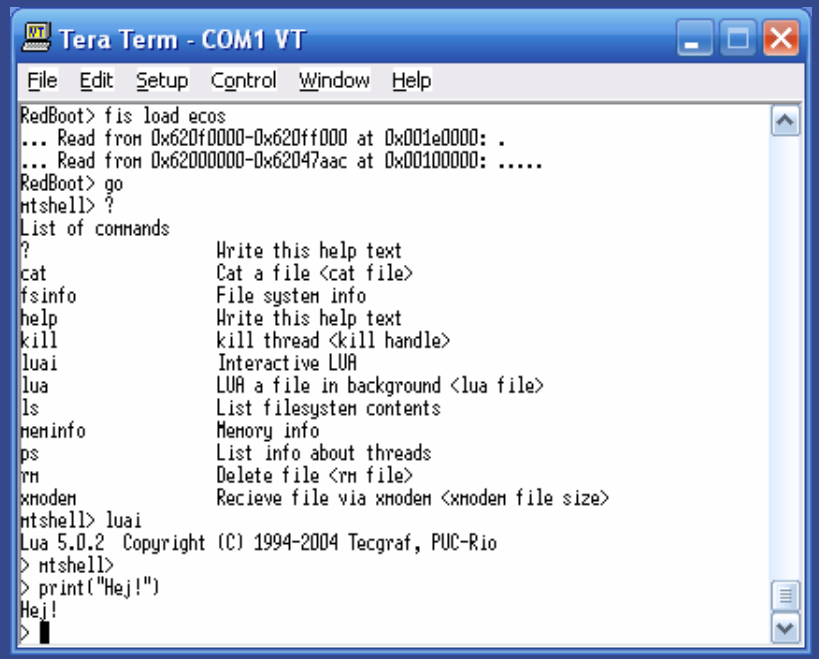
Property	Value
URL	ref/flash.html
Macro	CYGPKG_DEVS_FLASH_M25PXX
File	/home/aphex/armscomm1/software/ecos/build_tre...
Value	current
Default	current
Parent	CYGPKG_IO_FLASH
ActiveIf	CYGPKG_IO_FLASH
Requires	CYGSEM_IO_FLASH_READ_INDIRECT == 1

FLASH memory device support for m25pxx

Ready | No conflicts

Atmosfire eCos port

- There is a closed port for str7xx from eCoscentric
- Yours truly had to write his own (and publish are open source)
  - Good exercise into the inner workings of the HAL/kernel.



eCos Atmosfire setup

- str7xx HAL
- str71xx serial device
- m25Pxx spi flash device
  - (str71xx flash optional)
- MTFFS filesystem
  - Yes, it stands for Martin Trojer Flash File System
- Lua

Redboot

- Redboot is the eCos standard bootloader
- It's basically an eCos application, so it can be configured to do much of what eCos does
- It can boot more than eCos (Linux for instance)
- There are many boot options including tftp etc.
- It's typically about 100KiB big

eCos Sythetic Target

- It's possbile to run a eCos system within a host OS (i.e. Linux)
- This will give a fully functioning eCos system, that is faster and more easy to debug apps in
- You simply build a synth ecos in the build tree, it's then very easy to change your apps Makefile to use the syth libs

Atmosfire boot setup

- Redboot resides in the str71xx flash
- The m25pxx spi flash is devided in half by redboot and MTFFS (in eCos)
- Redboot reads the eCos image from flash and boots it
- eCos boots instantly after the “go” command

eCos size

- The Atmosfire eCos setup weighs in at about 110KiB (32bit arm)
- With Lua this becomes 296KiB
- That it truly “linux lite”, and great value for your flash bits
- 100KiB for redboot is kinda hefty in this setup and can be replaced by a simpler “home-cooked” bootloader

Just in case it's not clear...

eCos kicks a**!!!

eCos vs Linux

- Many Linux apps can be easily ported
  - Linux compat mode
- Linux code size typically lives in the MiBs, eCos lives in 100 KiB ~10 times smaller
- eCos boots much faster than Linux
- Ecos core functionality much more scalable than in Linux
- eCos has it's own core codebase

eCos vs Linux

- eCos could be used in many embedded applications where people use Linux today
- Not as widely known, but still very much worth a glance if you need this kind of functionality
- eCos is very usable on 60Mhz ARM7tdmi!
- eCos + Lua = true romance

Many cool labs this afternoon!