Lab 6, Compile something

By ADMIN | *Published:* DECEMBER 11, 2012

Now it is lab 6 and it is time to compile “something” – kernel of inferno, but we are going to compile without worrying that it would not work (even would not link). We just need inferno kernel which can be compiled okay for R-Pi using a lot of stabs.
Files/Folders structure:

```
inferno-os/
  |-os/
  |     |-rpi/
  |     |     |-rpi
  |     |     |-mkfile
  |     |     |-load.s
  |     |     |-main.c
```

That is how port for R-Pi is organized – “os” folder contains all supported native versions of inferno os. We create there “rpi” folder and move inside our “hello world” example containing load.s and main.c(test.c) files. Now we need a mkfile and a definition of this native version by “rpi” file (having same name as folder name). Let’s check native arm-based port in “../ipaq1110/” for example.

mkfile:

```
<../../mkconfig

CONF=rpi
CONFLIST=rpi
loadaddr=0x00008000

SYSTARG=$OSTARG
OBJTYPE=arm
INSTALLDIR=$ROOT/Inferno/$OBJTYPE/bin

<$ROOT/mkfiles/mkfile-$SYSTARG-$OBJTYPE

<| $SHELLNAME ../port/mkdevlist $CONF


OBJ=\
    load.$O\
    main.$O\
    $IP\
    $DEVS\
    $ETHERS\
    $LINKS\
    $PORT\
    $MISC\
    $OTHERS\
    $CONF.root.$O\

LIBNAMES=${LIBS:%=lib%.a}
LIBDIRS=$LIBS

CFLAGS=-wFV -I$ROOT/Inferno/$OBJTYPE/include -I$ROOT/include -
I$ROOT/libinterp
KERNDATE=`{$NDATE}

default:V: i$CONF

i$CONF: $OBJ $CONF.c $CONF.root.h $LIBNAMES
```


Categories

- Blog
- Boost
- C++
- Cryptography
- Embedding
- Hybrids
- Inferno OS
- MacAppStore
- Misc
- Models
- Projects
- PyQt
- PySide
- Qt
- QtSpeech
- Raspberry Pi
- Research
- Ru
- TogMeg
- Trac
- TTS
- Tutorial
- Undo
- Web

```
    $CC $CFLAGS -DKERNDATE=$KERNDATE $CONF.c
    $LD -l -o $target -R4 -T$loadaddr $OBJ $CONF.$O $LIBFILES

<../port/portmkfile

main.$O:    $ROOT/Inferno/$OBJTYPE/include/ureg.h
```

rpi:

```
dev
    root
    cons
    env
    mnt
    pipe
    prog
    srv
    dup

lib
    interp
    math
    kern
    sec

mod
    math
    sys

port
    alarm
    alloc
    allocb
    chan
    dev
    dial
    dis
    discall
    exception
    exportfs
    inferno
    latin1
    nocache
    nodynld
    parse
    pgrp
    print
    proc
    qio
    qlock
    random
    sysfile
    taslock
    xalloc

init
    bootinit

root
    /chan   /
    /dev    /
    /dis    /
    /env    /
    /fd     /
    /net    /
    /prog   /
    /dis/lib
    /dis/disk
```

Now if we try to compile with "mk" we will find fast that some header files are required:

```
# touch dat.h fns.h io.h mem.h
```

* **fns.h** - define signatures of required methods, also should include **"../port/portfns.h"**
* **dat.h** - kernel rpi specific data structures, also should include **"../port/portdat.h"**
* **io.h** - input/output defines and enums
* **mem.h** - defines related to our memory model

Also add the section for header files to "mkfile":

```
HFILES=\
```

```
    mem.h\
    dat.h\
    fns.h\
    io.h\
```

After mutiple times (well, not bazillion times) of change+compile+compare-with-ipaq1110/sa1110 we will reveal that minimal content of these header files is:

io.h:

-- EMPTY --

fns.h:

```
#define KADDR(p)    ((void *)p)
#define PADDR(p)    ((ulong)p)

int     waserror();
void    (*screenputs)(char*, int);

#include "../port/portfns.h"
```

mem.h:

```
#define KiB     1024u       /*! Kibi 0x0000000000000400 */
#define MiB     1048576u    /*! Mebi 0x0000000000100000 */
#define GiB     1073741824u /*! Gibi 0000000000040000000 */

#define KZERO       0                       /*! kernel address space */
#define BY2PG       (4*KiB)                 /*! bytes per page */
#define BY2V        8                       /*! only used in xalloc.c */
#define MACHADDR    (KZERO+0x2000)          /*! Mach structure */
#define ROUND(s,sz) (((s)+(sz-1))&~(sz-1))

#define KSTKSIZE    (8*KiB)
#define KSTACK      KSTKSIZE
```

dat.h:

```
#define HZ          (100)       /*! clock frequency */
#define MS2HZ       (1000/HZ)   /*! millisec per clock tick */
#define TK2SEC(t)   ((t)/HZ)    /*! ticks to seconds */
#define MS2TK(t)    ((t)/MS2HZ) /*! milliseconds to ticks */

#define MACHP(n)    (n == 0 ? (Mach*)(MACHADDR) : (Mach*)0)

typedef struct Lock Lock;
typedef struct Ureg Ureg;
typedef struct Label Label;
typedef struct FPenv FPenv;
typedef struct Mach Mach;
typedef struct FPU FPU;
typedef ulong Instr;
typedef struct Conf Conf;

struct Lock
{
    ulong   key;
    ulong   sr;
    ulong   pc;
    int pri;
};

struct Label
{
    int x;
};

enum /* FPenv.status */
{
    FPINIT,
    FPACTIVE,
    FPINACTIVE
};

struct FPenv
{
    int x;
};

struct  FPU
```

```
{
    FPenv env;
};

struct Conf
{
    ulong   nmach;      /* processors */
    ulong   nproc;      /* processes */
    ulong   npage0;     /* total physical pages of memory */
    ulong   npage1;     /* total physical pages of memory */
    ulong   base0;      /* base of bank 0 */
    ulong   base1;      /* base of bank 1 */
    ulong   ialloc;     /* max interrupt time allocation in bytes */
};

#include "../port/portdat.h"

struct Mach
{
    int     machno;     /* physical id of processor */
    ulong   ticks;      /* of the clock since boot time */
    Proc*   proc;       /* current process on this processor */
    Label   sched;      /* scheduler wakeup */
};

extern Mach *m;
extern Proc *up;
```

So, with this minimal "set" we already can compile all files of our inferno kernel, but of course we failed on the linking stage because we definitely need an implementation of referenced functions:

```
(2358)  BL  ,waserror+0(SB)
iprint: undefined: splhi
(2495)  BL  ,splhi+0(SB)
iprint: undefined: splx
(2502)  BL  ,splx+0(SB)
panic: undefined: setpanic
(2514)  BL  ,setpanic+0(SB)
panic: undefined: spllo
(2522)  BL  ,spllo+0(SB)
panic: undefined: dumpstack
(2523)  BL  ,dumpstack+0(SB)
panic: undefined: exit
(2525)  BL  ,exit+0(SB)
conswrite: undefined: reboot
(3391)  BL  ,reboot+0(SB)
conswrite: undefined: halt
(3394)  BL  ,halt+0(SB)
uartreset: undefined: addclock0link
(2106)  BL.NE   ,addclock0link+0(SB)
poolimmutable: undefined: getcallerpc

(2207)  BL  ,getcallerpc+0(SB)
too many errors
```

Anyway this is very good start to move to actual implementation of required routines to have our kernel linked and then we can retry our hello world example and move further to have some codes for the initialization stage.

ps: again this minimum set can be used as starting point for another arm boards with tweaking of mem.h for specifying addresses where if kernel is placed, kernel stack, etc.

Download files:

os/rpi/dat.h
os/rpi/fns.h
os/rpi/load.s
os/rpi/main.c
os/rpi/mem.h
os/rpi/mkfile
os/rpi/rpi

« *Lab 5, Hello World*

Lab 7, linking, planning next »