uClinux --
Micro-Controller Linux

Presented by

Greg Ungerer
<gerg@uclinux.org>
<gerg@snapgear.com>

SnapGear - A CyberGuard Company
825 Stanley St., Woolloongabba
QLD. 4102. Australia
PH: +61 7 3435 2888
www.snapgear.com

Outline

1. uClinux
2. Kernel
3. Libraries
4. Applications
5. Tools
6. Developing
7. Future Work
8. References

uClinux

Pronounced "you-see-linux", the name uClinux comes from combining the greek letter "mu" and the english capital "C". "Mu" stands for "micro", and the "C" is for "controller".

- Linux for processors that have no memory management
- patches against standard Linux kernel sources
- targets classic embedded 32bit micro-controllers

Main Features

- open source project
- stable versions based on Linux kernels 2.0.39, 2.4.27 and 2.6.10
- most features of Linux kernel available:
  - process control
  - filesystems
  - networking
  - device drivers
- modified kernel memory subsystem
- conventional kernel/application separation

CPU Architectures

Supported architectures:

- Motorola 68k (68X302, 68306, 68X328, 68332, 68360)
- Motorola ColdFire (5206x, 5249, 527x, 5307, 5407)
- ARM (Atmel, NetSilicon, Aplio, TI, Samsung, ...)
- Sparc (LEON)
- MIPS (Brecis, ...)
- Xilinx Microblaze (FPGA)
- Altera NIOS (FPGA)
- NEC v850
- Hitachi H8/300, SH2

CPU Architectures

Under Development:

- Motorola MCORE
- Opencores OpenRISC 1000
- Analog Devices Blackfin
- Intel i960

Kernel Features

PROCESS:	full multi-tasking process system, XIP supported
API:	same system call set as standard Linux
IPC:	software signals, shared memory!
FILESYS:	ROMfs, ext2, NFS, SMB, JFFS(2), proc, ISO9660, CRAMfs, ...
NETWORK:	TCP/IP, PPP (PAP, CHAP), masquerading, routing, filtering, forwarding, IPsec
DRIVERS:	serial, network, timer, IDE, MTD, audio, LCD, watchdog, PCI bus, PCMCIA
MODULES:	loadable modules supported

Limitations

- no virtual memory
- no memory protection
  - between kernel, processes or hardware devices!
- no real *fork()*, only *vfork()*
- cannot dynamically grow stacks
- no conventional *sbrk()*
- memory fragmentation more of a problem

MM Changes

- basic kernel allocation can be left 'as is'
  - power of 2 allocator not ideal
  - larger sized regions used for larger allocations
- obviously no page tables to setup/maintain
- no *sbrk()* system call, use *mmap()/mfree()*
  - library *malloc()* needs to use *mmap()*
- swapping not supported

Stacks and Modes

- kernel and user stack setup same as on VM systems
- may need to emulate user and kernel stack pointers
  - keep *usp* and *ksp* global variables
  - swap at trap entry/exit
- kernel and user 'modes' also maintained
- may need to emulate user mode
  - maintain mode word in kernel variable

fork() Problem

- copying existing data/stack regions is the real issue
- any copy will be at different location in address space
- absolute memory references after copying not valid
- no way to easily 'fixup' the moved data
- vfork() is fast and efficient

Application Issues

- no MMU is largely transparent
- *fork()* needs to be dealt with on a case by case basis
- runtime limits on size of *malloc()*s
  - kernel has to find a free chunk of memory that size
  - memory fragmentation can be a problem
- need to think about stack usage and preset accordingly

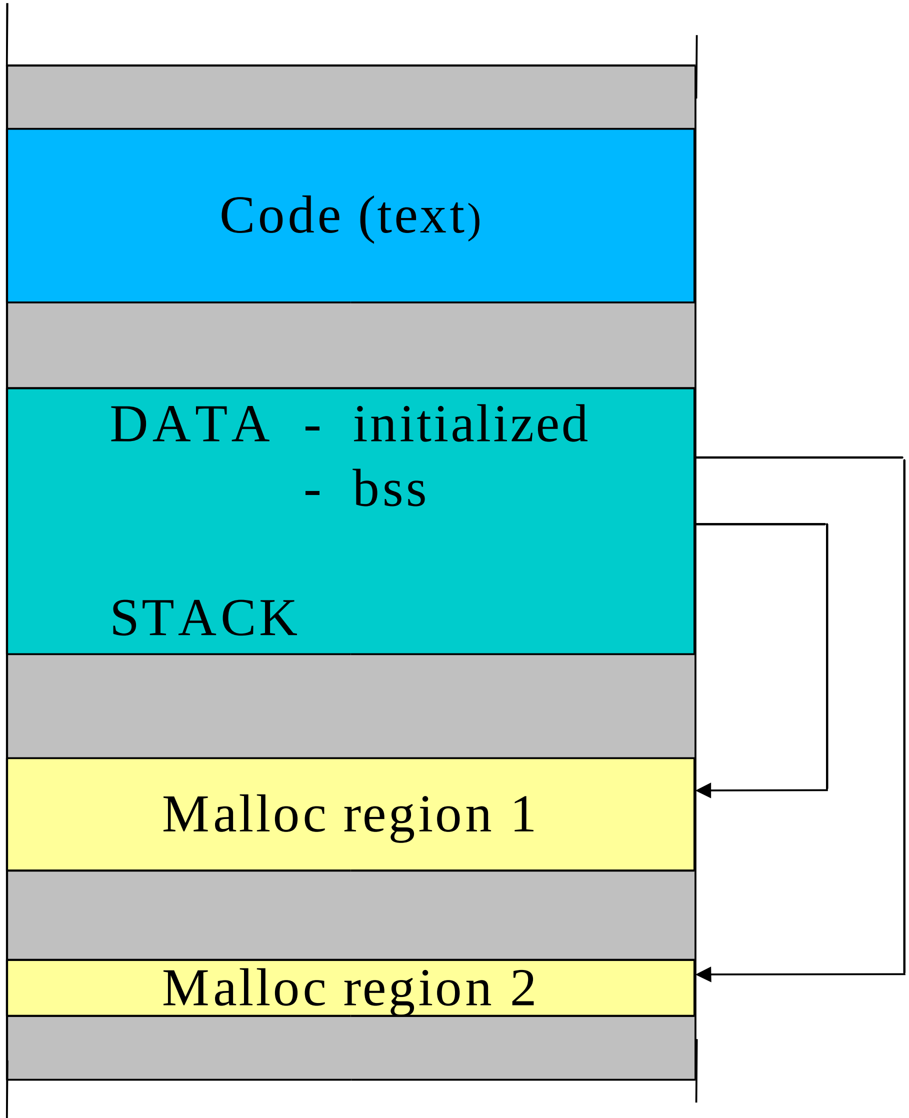
Application Methods

- new space saving file format, FLAT binary
- simple conversion from ELF format
- 2 models of object loading/executing:

1. Relocated applications
   - program code, data and stack allocated and loaded in RAM
   - relocation entries are patched at *exec()* time

2. PIC applications
   - code is position independent (thus can be shared)
   - each process gets allocated data and stack in RAM
   - execute in place (XIP) possible

Application Memory



- code may be shared if PIC
- data+stack region allocated in 1 chunk
- code+data+stack allocated in 1 chunk if relocating
- *malloc()/mmap()* regions freed on program exit by kernel

Libraries

- shared libraries supported on some platforms
- uClibc is preferred libc
  - small and light weight
  - mostly glibc compatible
- port of glibc exists on some platforms too
  - very large
- other supported libraries include:
  openssl, libpcap, libm, libdes, zlib, libpng, libjpeg

Tools

- standard GNU cross compile tool chain
  - ELF format (older versions used COFF)
- binutils (2.14) for as, ld, ar, objcopy
- gcc (2.95.3) (includes c++ support)
- gdb (5.0)
- elf2flt - FLAT format conversion from ELF
- x86 Linux PC most often used as development host

Developing

- get tool chain setup first
  - you need gas and gcc before you can port anything
- vendor development boards are an excellent start
- JTAG/BDM debugging hardware is _very_ useful
- supporting new boards with supported CPU is easy
  - existing board/platform support is pretty good
- supporting new CPU s can be a lot of work
- uClinux 2.0.39 and 2.4.27 are very mature and very stable

Future Work

- more CPU and platform support
- more hardware device support
- more applications ported
- shared library support on more architectures
- real time support (RTAI)
- maintaining source in mainline kernels

References

- uClinux
  http://www.uclinux.org

- uClinux CVS
  http://cvs.uclinux.org

- uClinux (and embedded linux) news
  http://www.ucdot.org

- GNU tools and source
  http://www.gnu.org