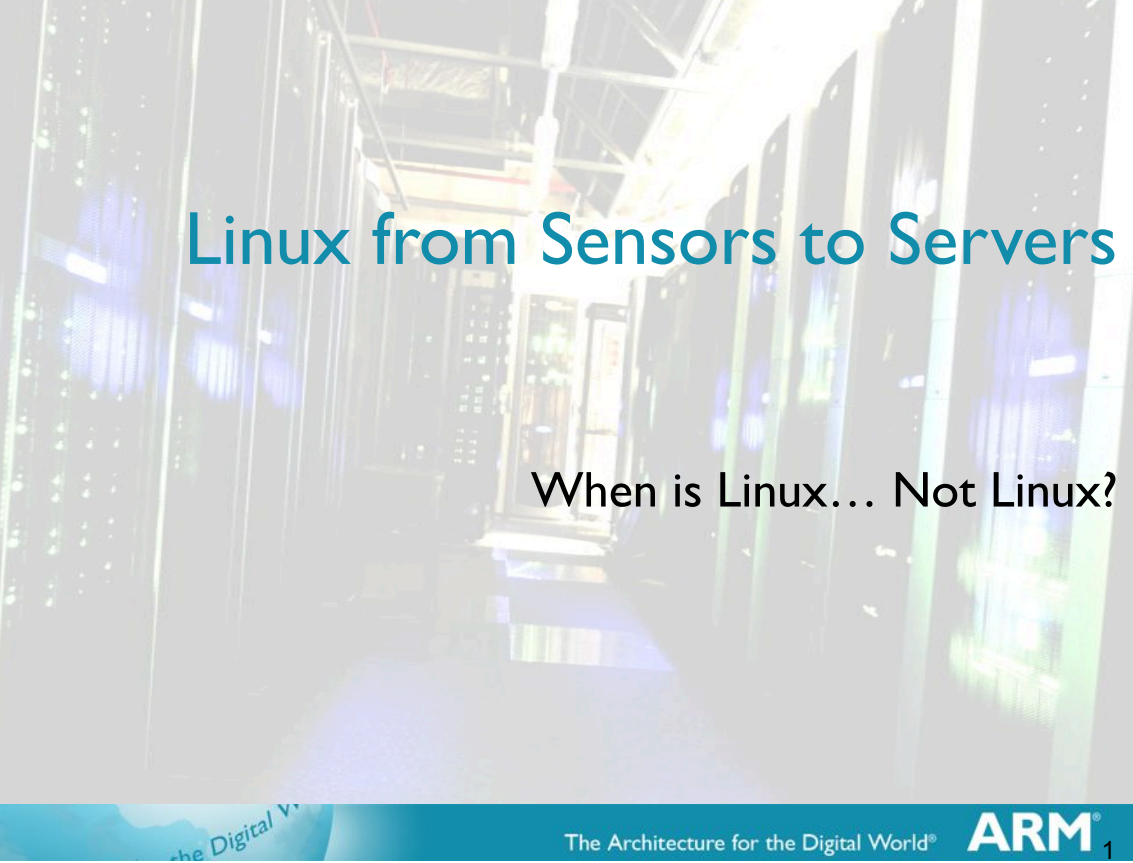


Linux from Sensors to Servers

When is Linux… Not Linux?

Linux runs across a huge range of systems

What's the difference between Linux on a big thing and Linux on a little thing?

What's the difference between Linux kernels, userspace and toolchains on a big thing and a little thing?

What's the difference between Linux kernels, userspace and toolchains on a system with an MMU and one without?

What's the difference between Linux kernels, userspace and toolchains on an ARM system with an MMU and one without?

Overview

- A, R and M class cores
- Anatomy of a (uC)Linux system
  - uClibc: with or without an MMU
  - Multitasking without an MMU
- What's different in uClinux/!MMU?
  - For the kernel
  - For userspace
  - In toolchain-land
- SMP uClinux and how Linux (doesn't) use the MPU (memory protection unit)

<marketing>

Comparison via ARM's product range

A Application

R Real-time

M Microcontroller (eMbedded?)

Typical systems

A

- MMU
- SMP
- System coherency
- GBs of memory
- Displays
- GPUs

R

- MPU
- 100s of MBs of memory
- MP-core/SMP
- 100s of MHz

M

- No MMU
- SRAM (KBs-MBs)
- Off-chip RAM
- Tiny displays
- 10s-100s of MHz
- Cheap!

Typical Applications

A

- Smartphones
- Laptops (Chromebook)
- TV/STB
- ‘Embedded’ systems
- Servers!

R

- High density storage
- Baseband-processors
- Automotive
- Medical/industrial
- More ‘Embedded’ systems

M

- Little things
  - Sensors
  - Data loggers
- Smart watches
- White goods
- Task-specific applications alongside A-class

</marketing>

	A	R	M
Exception Model	V7A/R	V7A/R	V7M
Memory model	VMSAv7	PMSAv7	V7M (i.e. M3/M4)
Memory protection	MMU	Limited protection from MPU – none between userspace tasks and kernel.	MPU not used in Linux yet, so none at all.
Pre-emptible kernel threads	Yes		No – kernel runs in handler mode and isn't pre-empted by SWI
Binary Formats	ELF, FLAT, a.out	FLAT (BFLT)	
Shared libraries	Via virtual memory. The way it should be	Only 4 per application, require unique numbers managed by custom building and configuration. No ABI for this!	
Support for Real-Time	No. BUT there's a low-latency patch called preempt-rt that does 'soft realtime' (not in the mainline kernel)		Not with current Linux implementation

	A	R	M
Exception Model	V7A/R	V7A/R	V7M
Memory model	VMSAv7	PMSAv7	V7M (i.e. M3/M4)
Memory protection	MMU	Limited protection from MPU – none between userspace tasks and kernel.	MPU not used in Linux yet, so none at all.
Pre-emptible kernel threads	Yes		No – kernel runs in handler mode and isn't pre-empted by SWI
Binary Formats	ELF, FLAT, a.out	FLAT (BFLT)	
Shared libraries	Via virtual memory. The way it should be	Only 4 per application, require unique numbers managed by custom building and configuration. No ABI for this!	
Support for Real-Time	No. BUT there's a low-latency patch called preempt-rt that does 'soft realtime' (not in the mainline kernel)		Not with current Linux implementation

V7M Exception Model

Just three modes:

- Thread mode
  - Privileged
  - Unprivileged (can use svc to escalate permissions)
- Handler mode
  - Always privileged

Thumb2 only (no ARM!)

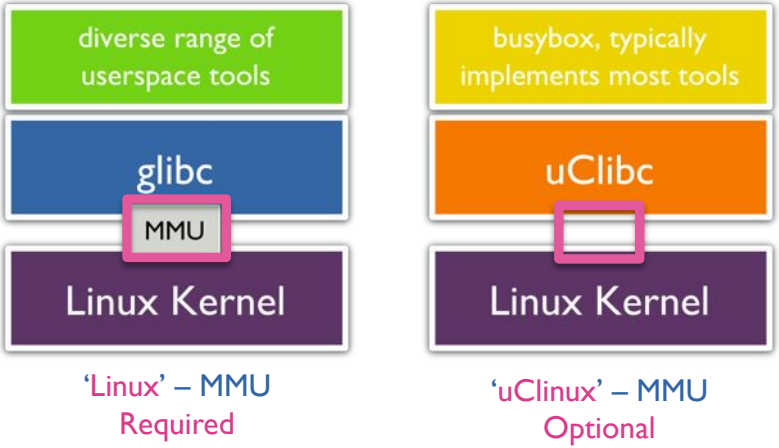
Thread mode can use the 'main' or the 'process' stack, Handler mode always uses the main stack

V7M is very different to V7A/R!

Anatomy of a (uC)Linux system

Overview

- A, R and M class cores
- Anatomy of a (uC)Linux system
  - uClibc: with or without an MMU
  - Multitasking without an MMU
- What's different in uClinux/!MMU?
  - For the kernel
  - For userspace
  - In toolchain-land
- SMP uClinux and how Linux (doesn't) use the MPU (memory protection unit)



'Linux' – MMU Required

'uClinux' – MMU Optional

'uClinux' refers to any system using the Linux Kernel and uClibc. uClibc can be built with or without support for an MMU. Whereas a glibc/Linux system (what people think of as 'Linux') requires an MMU, uClinux can be built to support hardware without an MMU.

We talk about uClinux/NoMMU to refer to the kind of uClinux that we use on R and M-class.

Some terminology

- Linux – the kernel, built with either CONFIG_MMU or not
- uClibc – a stripped down C-library, can be configured !MMU
- uClinux – a system built with Linux and uClibc. May target hardware with an MMU
- uClinux/!MMU – a system built with the Linux kernel

Multitasking without an MMU

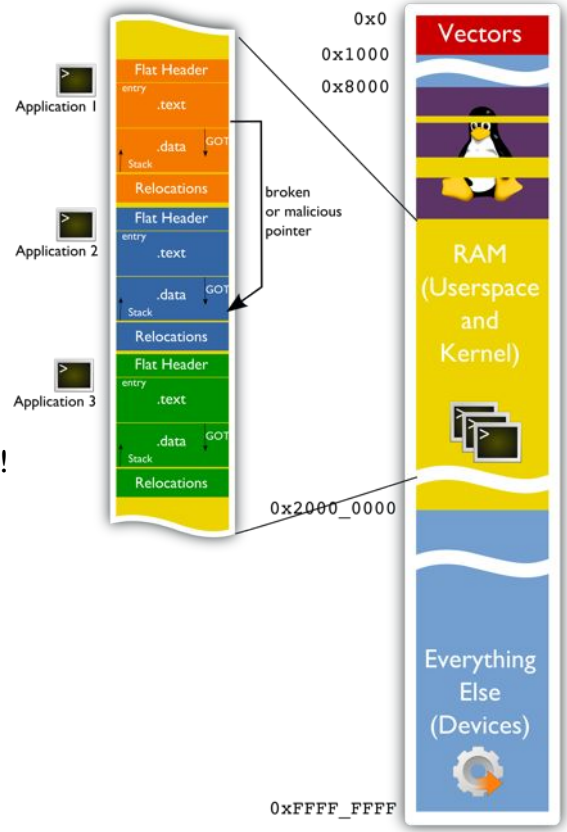
...must be done co-operatively and carefully! The exception models of V7M and V7A/R allow the kernel to pre-empt userspace, so it isn't quite like 'the bad old days'. However, there is very little to protect tasks from each other.

- uClinux without an MMU is **not** recommended for any scenario where input is coming from 'the outside world' and security is important!
- The MPU offers some extra security but the design of Linux and the MPU are not very compatible.

Sharing the address space

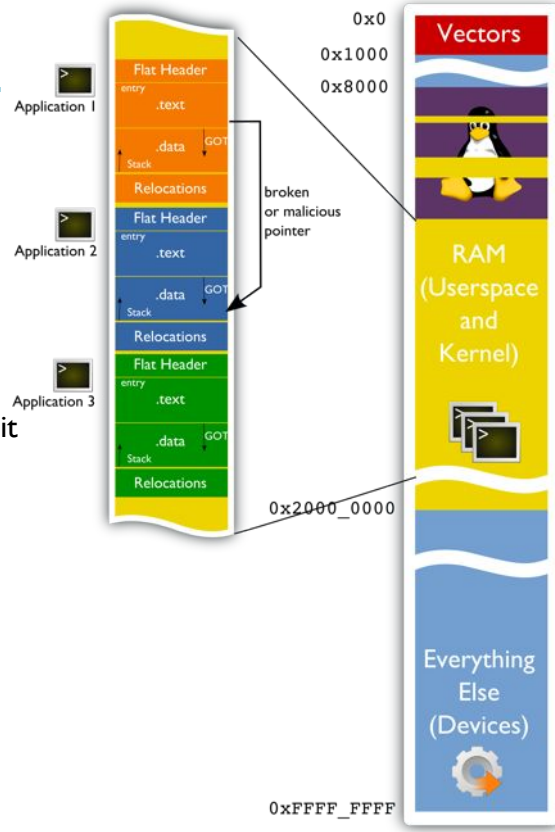
- Virtual memory IS physical memory
- Processes are loaded next to each other.
- Pointers are suddenly very dangerous!
- Security is…challenging…

- Data freed inside the kernel complicates the memory layout



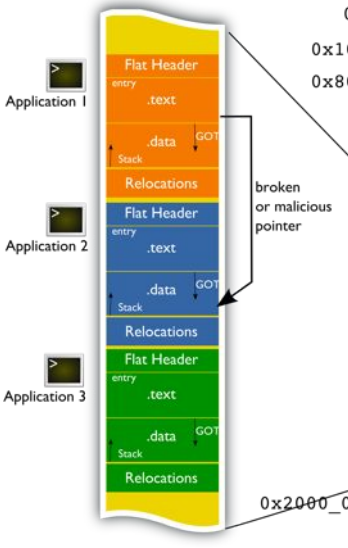
Position Independent Code Required

- Shared address space → apps cannot be linked at a fixed location as you do when there's an MMU.
- Position independent code (PIC) is used and every binary is relocated as it is loaded.
- R9 is used as the 'PIC offset base register' that points to the Global Offset Table.
- Code linked at fixed offset will break!



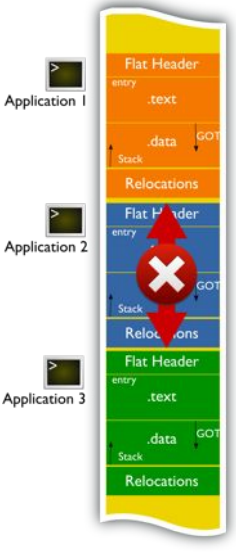
Pointers: Danger!

- Bad or malicious pointers might point outside your binary
  - One program can cause corruption in another one, including the kernel
  - Tough to debug, userspace bugs can show as kernel panics!
- Special case: jumping to a null pointer
  - Linux places a special 'SVC 0' at 0x0
    - if a programme jumps to a null pointer
- Protection on R-class via the MPU



malloc()

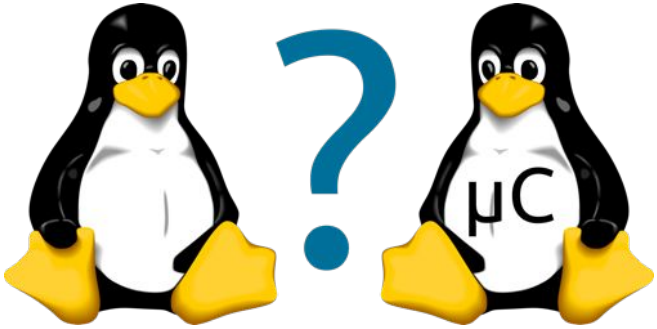
- Limited ability for ‘sbrk’ operations to increase memory allocated to a task
- malloc for !MMU uses a global, shared memory pool
- This approach suffers from fragmentation issues, there may be enough memory available but not in a contiguous chunk
  - Allocate smaller chunks, rather than big ones
  - Design restartable applications



Overview

- A, R and M class cores
- Anatomy of a (uC)Linux system
  - uClibc: with or without an MMU
  - Multitasking without an MMU
- What's different in uClinux/!MMU?
  - For the kernel
  - For userspace
  - In toolchain-land
- SMP uClinux and how Linux (doesn't) use the MPU (memory protection unit)

Linux Kernel without an MMU

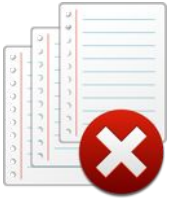


The kernel for uClinux is no longer a fork

- NoMMU support for Linux kernel merged in 2002
  - Greg Ungerer (et al!)
- ARM/NoMMU Support merged in 2007
  - Hyok S. Choi
- Cortex-M3 and Cortex-R7 support merged in 3.11
  - Uwe Kleine Konig, Jonny Austin, Catalin Marinas,

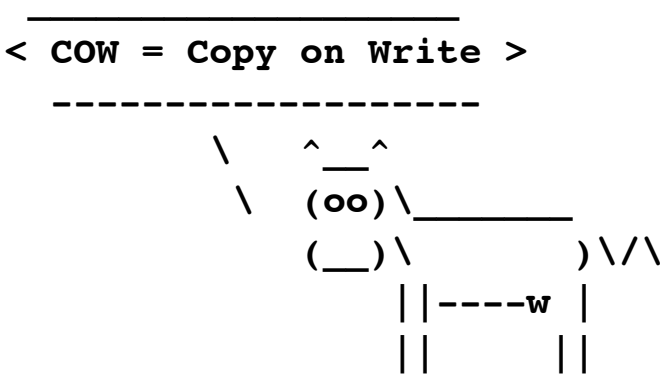
Major differences with !CONFIG_MMU

- No support for the fork() system call (vfork() instead)
- No elf support (elf is the standard binary format for Linux). BFLT instead
- ABI is different for certain operations (a modified C-library is required)
- No 'kuser helpers' (utility functions provided by the kernel at fixed addresses
- No paging, memory management which leads to fragmentation issues with mmap() and a need to load all code instead of relying on faulting it in.



This is an elf…

What?! No `fork()`? — Because of CoW



Fork and CoW

- The fork() syscall 'completely copies' the address space of the parent for the child. Linux uses CoW to provide the 'fork' system call efficiently
- The parent and child share the same pages until they're written to...
- This relies on the existence of an MMU!
- fork() is very commonly followed by exec(), which blows away the existing address space. Because of this, implementing fork() for uClinux would commonly have a huge, unnecessary overhead
- ➔We don't have fork() on uClinux /NoMMU (uClinux **with** an MMU can use fork(), which is a source of much confusion)

…vfork() instead

uClinux/NoMMU does have 'vfork()', which can be used instead:

- When a new process is created with vfork(), the parent process is temporarily suspended
- Child process executes 'in the parent's address space' until
  - child exits OR calls execve()
- ...At which point the parent process continues.

…vfork()

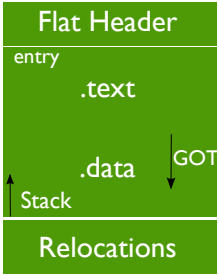
- Fork+exec can be simulated by vfork() followed by exec() of the same binary (modified to read the new arguments and jump to the right place)
  - If this is done, there is limited impact on multitasking
- If a task does more than a simple fork-then-exec, 'porting' the behaviour to uClinux can be non-trivial.
  - The child can clobber things the parent later relies on

Eg uClinux spawn for the 'dumb' case

```
int main(int argc, char *argv[])
{
        char *newargs[2];
        pid_t pid;

        if ((pid = vfork()) < 0) {
                fprintf(stderr, "failed to vfork\n");
                exit(1);
        } else if (pid != 0) {
                /* We're the parent, and we should just gracefully exit */
                printf("Parent is exiting...\n");
                exit(0);
        }

        /* Could be the same binary as is already running */
        newargs[0] = "/bin/newtask";
        newargs[1] = NULL;
        execv(newargs[0], newargs);

        /* Not reached */
        fprintf(stderr, "Couldn't exec\n");
        _exit(1);
}
```

BFLT instead of elf

- Simple binary format
  - Designed for !MMU systems, based on a.out
  - Smaller header than elf
  - Easier to load at arbitrary location then fixup relocations
  - PIC required (more later)
- uClinux specific toolchains required to make BFLAT from elf images with an elf2flt linker script
  - `-Wl,-elf2flt`
  - Keep the elf for debug

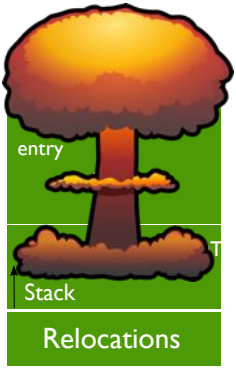



```
struct flat_hdr {
   char magic[4];
   unsigned long rev;          /* version */
   unsigned long entry;        /* Offset of first executable instruction
                                  with text segment from beginning of file */
   unsigned long data_start;   /* Offset of data segment from beginning of
                                  file */
   unsigned long data_end;     /* Offset of end of data segment
                                  from beginning of file */
   unsigned long bss_end;      /* Offset of end of bss segment from beginning
                                  of file */

   /* (It is assumed that data_end through bss_end forms the bss segment.) */

   unsigned long stack_size;   /* Size of stack, in bytes */
   unsigned long reloc_start;  /* Offset of relocation records from
                                  beginning of file */
   unsigned long reloc_count;  /* Number of relocation records */
   unsigned long flags;
   unsigned long filler[6];    /* Reserved, set to zero */
};
```

A good reference for BFLAT: http://retired.beyondlogic.org/uClinux/bflt.htm

Fixed-stack

- With an MMU, VM can be used to dynamically increase the stack
- !MMU → Fixed stack
  - We don't event get exceptions when a program overflows the stack!
  - 'Silent' corruption that will be witnessed later
- Specify stack size when building (`-s 16384`)



uClinux/uClibc Toolchains

	A	R	M
Toolchain availability	Trivial! Widely available	Nontrivial! Codesourcery toolchain form 2011 (if you can find it). Build your own/use Buildroot/ptxdist	Even less trivial than R-class as you need thumb2 binaries. Build your own/find someone who has…
C Library	Usually glibc	uClibc	
Threading	pthreads, etc. Lots of higher level languages too.	No fork()! Linux threading. Some pthreads if you're lucky.	No fork()! Linux threads. pthreads doesn't build for Thumb2
Binary Formats	ELF, FLAT, a.out	FLAT (BFLT)	
User Stack	Dynamically allocated (up to limit). Protected by MMU	Statically allocated when binary is loaded, overflowable!	
Shared libraries	Via virtual memory. The way it should be	Only 4 per application, require unique numbers managed by custom building and configuration. No ABI for this!	

uClinux tools

A uClinux toolchain is typically GCC that incorporates the uClibc C-library.

- uClibc was mostly written from scratch but incorporates bits of glibc.

Most people build their own toolchain for uClinux, but it isn't uncommon for !MMU builds to break. Configurations are not standardised in the same way as they are for other tools, either

- The 'triplet' does not describe uClinux tools as uniquely as it does for A-class/Glibc. For example, arm-none-uclibc-uclinuxeabi is the same as arm-uclinuxabi!

Elf2flt required as the linker, 'strip' doesn't run on BFLT so many unpatched tools *think* they've failed to build


Threading?

What about pthreads?

- Multiple different pthreads implementations: linuxthreads, nptl
- None of them have THUMB2-only implementations for atomic operations (yet)
  - They use ARM mode or the now-deprecated swp instruction
  - So no (upstream) pthreads on M-class yet
- Some out-of-tree modifications to work around this:
  - Either requiring custom syscall
  - Or patched using load/store-exclusives

Shared Libraries

- Caveat: Different on blackfin/frv/sh with BINFMT_FDPIC_ELF
- FLAT shared
  - Each library needs its own unique ID
  - Toolchain allows 256 IDs, Kernel allows only 4!
  - include/uapi/linux/flat.h

```
16 #ifdef CONFIG_BINFMT_SHARED_FLAT
17 #define MAX_SHARED_LIBS                 (4)
18 #else
19 #define MAX_SHARED_LIBS                 (1)
20 #endif
```

Shared FLAT

- 4 shared libraries
  - 0: the application itself
  - 1: The C library
  - 2: pthreads
  - 3: yours to re-use!
  - 4: yours to re-use!

Can be used in conjunction with XIP (execute in place) to save memory.

If you really want to do this, follow the docs at

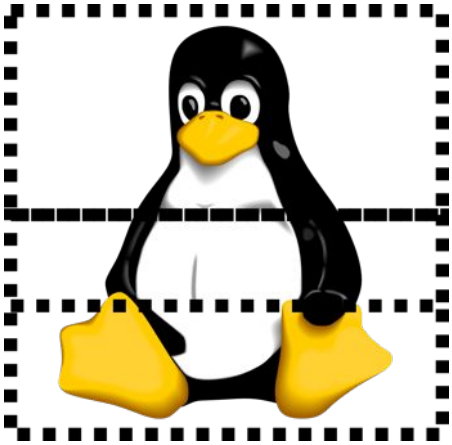
http://blackfin.uclinux.org/doku.php?id=toolchain:creating_libraries

Overview

- A, R and M class cores
- Anatomy of a (uC)Linux system
  - uClibc: with or without an MMU
  - Multitasking without an MMU
- What's different in uClinux/!MMU?
  - For the kernel
  - For userspace
  - In toolchain-land
- SMP uClinux and how Linux (doesn't) use the MPU (memory protection unit)

How does Linux (not) use the MPU?

MPU = Memory Protection Unit

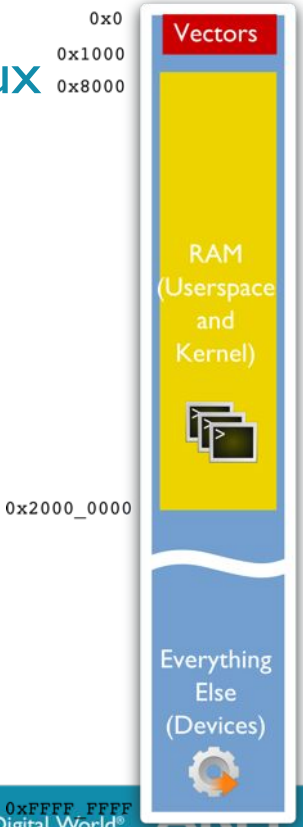


Protection regions, instead of VM

- The MPU lets you define up-to 16 (overlapping) regions
- Regions can have
  - Different access permissions
  - Different memory types
  - Can be marked 'XN'
- Higher numbered regions have higher 'priority' when regions overlap
- Regions can be divided in to subregions
- R-class: cp15, M-class: memory-mapped

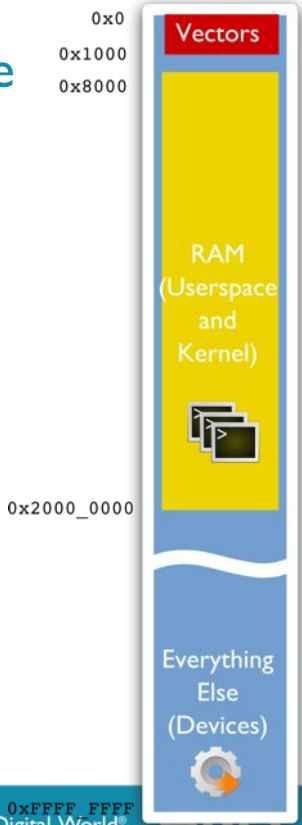
MPU is required for SMP uClinux

- In order for exclusives to work, the correct memory attributes must be set using the MPU.
- Secondary cores need their MPU configured before entering the C-world
  - Bring-up looks very similar to what we do for MMU/Page tables
- Currently a very minimal setup



Linux for R-class makes minimal use of the MPU

- Set up regions to ensure that DRAM is shared so SMP works.
- Mark all of the rest of the address space as 'device' memory, XN
- Protect the vectors from being accessed or executed from userspace (this also stops null pointer offsets from causing a kernel panic, instead just killing the task)



How can we make better use of the MPU?

- Dynamically moving windows?
  - Fault every time we read outside of the currently mapped windows, determine permissions of the faulting address, map region if necessary. This has a **lot** of overhead. A high percentage of memory accesses will fault! Breaks deterministic memory access!
- Disable Linux's funky memory management?
  - Don't free things inside the kernel, so we can at least protect the kernel from userspace – a reasonable compromise
- Make malloc always give MPU mappable region sizes
  - This wastes lots of memory, suboptimal on tiny systems!
  - Still needs us to remap regions on context switch
- Map 'guard region' at the end of the stack to detect overflow

References

- Free Electrons uClinux Introduction:
  - http://free-electrons.com/doc/uclinux_introduction.pdf
- Blackfin uClinux wiki
  - http://blackfin.uclinux.org/
- uClinux 'State of the Nation' ('07)
  - http://elinux.org/images/e/e4/Uclinux-sotn.pdf

Questions & Suggestions?