4GB by 4GB split
One 32bit address space for everyone
Refresher: monolithic kernels

- **User process**
  - Unprivileged abstract machine
  - Virtual resources: file handles, memory

- **Kernel, Supervisor**
  - Higher privilege level
  - Everything shared
  - Physical resources: CPU, memory, network, storage

- **State change, exception entry**
  - CPU control transfer to privileged kernel
  - Synchronous: system call, page fault, invalid instruction
  - Asynchronous: hardware request, inter-processor-interrupt
  - Exit restores user state, possibly to different process
Linux/Arm virtual memory map

- Similar to Linux on most other architectures
- Per-process page table
- Low virtual addresses accessible by user and kernel*
- High virtual addresses accessible in privileged mode
- Dynamic kernel mappings at top address
  - Modules, vmalloc, ioremap, kmap
- Physical memory mapped to inbetween

* Unless using CONFIG_CPU_SW_DOMAIN_PAN
Highmem

Virtual addresses

Physical addresses
Removing highmem?

“Highmem sucks no matter what and the only sane solution is to remove it completely”

Thomas Gleixner, while working on improving highmem

https://lore.kernel.org/lkml/20200919091751.011116649@linutronix.de/

- LWN: An end to high memory https://lwn.net/Articles/813201/
- LWN: Highmem getting slower https://lwn.net/Articles/831678/
Changing vmsplit

- **CONFIG_VMSPLIT_3G_OPT**
  - 2.75GB user
  - 1GB lowmem

- **CONFIG_VMSPLIT_1G**
  - 1GB user virtual
  - 2.75GB lowmem

- LWN: Linker limitations on 32-bit [https://lwn.net/Articles/797303/](https://lwn.net/Articles/797303/)
Introducing the 4G:4G split

Before

User

Physical lowmem

vmalloc

After

Physical lowmem

User

vmalloc
4G:4G split with highmem

Kernel virtual

User virtual

Physical address

vmalloc

Physical lowmem

User

vmalloc

Physical highmem

Physical lowmem

0xffffffff

0xf0000000

0x00000000

0x00000000

4G:4G split with highmem
Most common memory sizes (MB) on 32-bit Arm

Estimate on machines still getting Linux kernel updates in 2020

Meaningless Y axis

(LP)DDR4
(LP)DDR3
(LP)DDR2
(LP)DDR
SDRAM
SRAM

Highmem Systems
Future 32-bit systems running Linux

- Cortex-A7 class with 512 MB DDR3
- Larger systems increasingly 64-bit+LPDDR4
- Smaller systems getting obsolete
- New chips until ~2030
- New deployments until ~2040
- More information at: https://linuxplumbersconf.org/event/7/contributions/655/
64-bit hardware

- Under 2GB RAM, run 32-bit user space
- Over 512MB RAM, run a 64-bit kernel
LPAE vs classic VMSAv7 MMU

- LPAE on ARMv7VE+ cores:
  - Cortex-A7, Cortex-A15, Cortex-A17, PJ4B, Armv8 AArch32
  - 4GB uncommon, 8GB+ exceptional

- Older cores (Arm11, Cortex-A8, Cortex-A9, Cortex-A5)
  - More obscurities in platform code
  - Usually 1GB or less memory, 2GB rare

- Exceptions:
  - NXP i.MX6QuadPlus (Cortex-A9)
  - Calxeda Highbank (Cortex-A9, 4GB)
  - Armada XP (PJ4B, up to 8GB)
  - TI Keystone2 (Cortex-A15, up to 8GB)
<table>
<thead>
<tr>
<th>Possible timeline</th>
<th>LPAE kernel</th>
<th>non-LPAE kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Add CONFIG_VMPLIT_4G_4G</td>
<td>1. Add CONFIG_VMPLIT_2G_OPT</td>
<td></td>
</tr>
<tr>
<td>2. Default to VMSPLIT_4G_4G</td>
<td>2. Default to VMSPLIT_2G_OPT</td>
<td></td>
</tr>
<tr>
<td>4. Drop support for &gt; 4GB systems and highmem in 2025+</td>
<td>4. Drop support for &gt;2GB systems and highmem in 2025+</td>
<td></td>
</tr>
</tbody>
</table>
Historic CONFIG_VMSPLIT_4G_4G on x86

- Created by Ingo Molnar to support 4GB lowmem / 60GB highmem
- Trampoline on kernel entry/exit
- Full TLB flush on each context switch
- Software page walk on copy_to_user/copy_from_user
- Never got merged because of CPU overhead
- Abandoned because of 64-bit CPUs
Page table tricks with Arm LPAE

- Two page table base registers: TTBR0, TTBR1
- TTBR0 for per-process low addresses
- TTBR1 for shared privileged high addresses
- Configurable split today: 1G/3G, 2G/2G, 3G/1G
- Hardware support for 0.25G/3.75G, 0.5G/3.5G, 3.5G/0.5G, 3.75G/0.25G
- Address Space ID (ASID) avoids TLB flushes
- TTBR0 unmap in kernel mode for full isolation
Problems

1. Exceptions

2. User memory access
Exception entry

- Kernel runs near start of lowmem, address reused in user space
- x86 workaround: trampoline page at high address
- Better: relocate whole kernel to vmalloc space
Kernel in vmalloc space

- **Today**
  - 256MB virtual address range
  - Contains modules, MMIO mappings, and vmalloc allocations

- **Plan**
  - Kernel vmlinux needs ~20MB address space
  - Mapped using TTBR1 page table base
  - Possibly Virtually mapped kernel stack
  - TTBR0 switch on kernel entry/exit
  - Reserve one ASID for lowmem TTBR0
  - No need for trampoline
  - Prototype by Afzal Mohammed
User memory access

- `get_user/put_user, copy_from_user,copy_to_user`
- Rewrite needed
- Multiple ideas, all slow
User memory access, #1 (original x86 method)

1. Walk page table to find physical page, prevent unmap with `get_user_pages()`
2. If highmem, map using `kmap()`
3. Copy in-page data with `memcpy()`
4. Unmap with `kunmap()`
5. Repeat for 2. and 3. for each page

- Prototype implemented by Afzal Mohammed
- Possible race against `mmap/munmap`
- Runtime overhead:
  - Page table walk plus local TLB invalidate per 4K page
User memory access, #2 (get_user/put_user)

1. Change ttbr0 to user page table
2. read/write one register
3. Handle page fault if necessary
4. Change ttbr0 back to kernel page table

- Overhead:
  - Two ttbr0 writes plus barriers per data word
User memory access, #3, (use more registers)

1. Change ttbr0 to user page table
2. read/write up to 8 registers with ldm/stm, 32 bytes
3. Handle page fault if necessary
4. Change ttbr0 back to kernel page table
5. write/read those registers to kernel memory
6. Repeat 1-4 for next 32 bytes

● Overhead:
  ○ Two ttbr0 writes plus barriers per 32 bytes
User memory access, #4, (use NEON registers)

1. Save user NEON register if necessary
2. Change ttbr0 to user page table
3. read/write up to 32 NEON registers (256 bytes)
4. Handle page fault if necessary
5. Change ttbr0 back to kernel page table
6. write/read NEON registers to kernel memory
7. Repeat 2.-6. For each 256 bytes
8. Restore user NEON registers upon next use

● Overhead:
  ○ Two ttbr0 writes plus barriers per 256 bytes
  ○ Full NEON register save/restore
User memory access, #5, (remap kernel memory)

1. Remap kernel memory into vmalloc (ttbr1) area with vmap()
2. Change ttbr0 to user page table
3. Copy using memcpy()
4. Change ttbr0 back to kernel page table
5. Call vunmap() or to remove mapping

- Overhead:
  - Two ttbr0 writes plus barriers per copy
  - Page table update
  - Local TLB flush
User memory access, #6, (vmallocl stacks)

- In copy_to_user(), copy outbound data to on-stack buffer with memcpy()
- Change ttbr0 to user page table
- Copy using memcpy() between stack and user space
- Change ttbr0 back to kernel page table
- In copy_from_user(), copy inbound data from on-stack buffer
- Repeat as needed depending on temporary buffer size

- Overhead:
  - Two ttbr0 writes plus barriers per on-stack buffer size
  - Reduced vmallocl address space
  - Double memcpy()
User memory access, actual plan

- Start with page walk prototype
- Try other approaches, measure overhead
- Pick the fastest method based on size
Security implications: PAN

- **Bug: Unchecked kernel access to user pointer**

- **Existing Mitigations:**
  - Armv8.1-A “Privileged-access-never”
  - Armv8.0-A CONFIG_ARM64_SW_TTBR0_PAN
  - x86 SMAP
  - Armv4 through v7 (non-LPAE) CONFIG_CPU_SW_DOMAIN_PAN

- **Armv7-A (LPAE) protected by** CONFIG_VMSPLIT_4G_4G
Security implications: Meltdown

- Bug: hardware allows bypassing page read protection
- Existing mitigation:
  - Kernel page table isolation (arm64, ppc, x86)
- CONFIG_VMSPLIT_4G_4G partial mitigation:
  - Protects access to linear mapping
  - Does not prevent reading vmalloc space
Other 32-bit architectures

- **MIPS**: 512MB lowmem, HW restriction in MIPS32r2 and earlier
  - Baikal T1 MIPS32r5: up to 8GB
  - Ingenic JZ4770+ MIPS32r2: up to 1GB DDR2/3

- **x86**: 768MB to 3GB lowmem (configurable), lacks ASID
  - Intel Xeon: up to 64GB, all obsolete
  - Core Duo (Yonah), Atom N270: 3GB memory limit
  - Vortex86EX2: 2GB memory limit

- **PowerPC**: 768MB lowmem, could do VMSPLIT_4G_4G
  - PowerBook G4: 2GB memory limit
  - Freescale/NXP P4080: up to 8GB DDR3

- **ARC, CSky, microblaze, nds32, sparc, xtensa**: highmem supported
  - Could not find systems using highmem
Thank you
4GB/32 bits for everyone