CloudSigma
semidynamics
silicon design and verification services
BSC
Riser
RISER: RISC-V for cloud services
EXA P SYS
EXTOLL
latency matters.
FORTH
INSTITUTE OF COMPUTER SCIENCE
SIPEARL

NOMMU Linux on RISC-V for platform bring-up and evaluation

Nick Kossyfidis, Manolis Marazakis (FORTH)

Disclaimer:

"Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Health and Digital Executive Agency (HaDEA). Neither the European Union nor the granting authority can be held responsible for them."

European Commission
Horizon Europe
2021-2027

Call: Open source for cloud-based services, GA Nr: 101092993 (HaDEA)

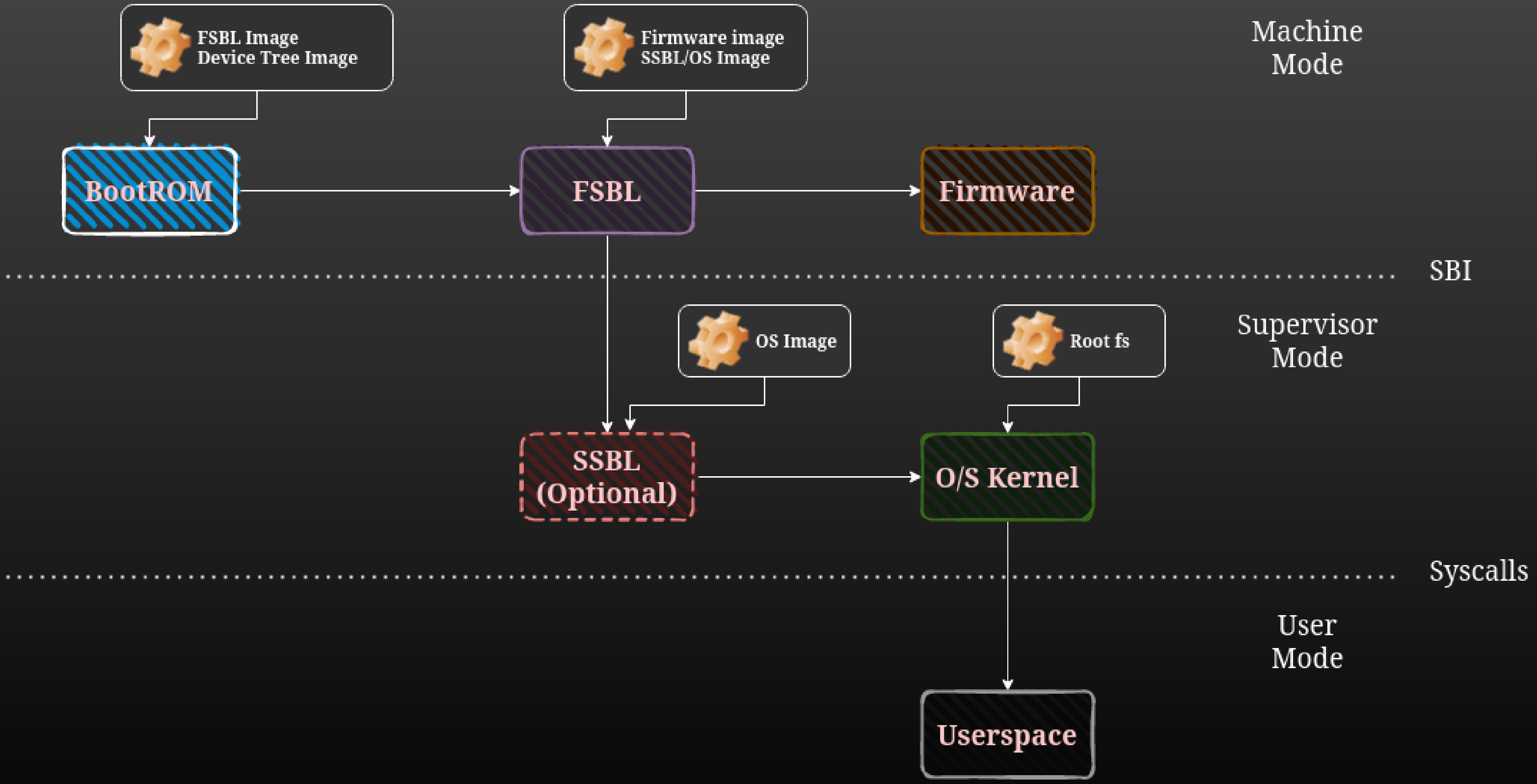
SoC: Design, Integration, Verification

- Design of individual CPU functional units
  - ALU, FPU, VPU, MMU, …
- Verification of individual functional units
  - e.g. directed/random tests, in a simulator
- Verification of the whole core
  - e.g. RISC-V ACT suite, checks against the SAIL model/reference simulator
- Post-synthesis co-simulation tests
- Integration with other IPs
  - Each IP with its own set of pre/post-synthesis tests

RISC-V SoC bring-up

- More advanced bare-metal tests for verification of the core
  - e.g. parts of the RISC-V spec not covered by ACT, custom extensions
- Progressively more complex bare-metal platform-level tests
  - e.g. interrupt delivery/delegation, communication between peripherals, peripheral operation
- Memory subsystem tests
  - e.g. litmus, cache-coherency with peripherals, IOMMU
- Security-related tests
  - e.g. constant-time requirements, TRNG operation, MTT, xPMP
- Stress testing/profiling/benchmarking

The way to full Linux boot

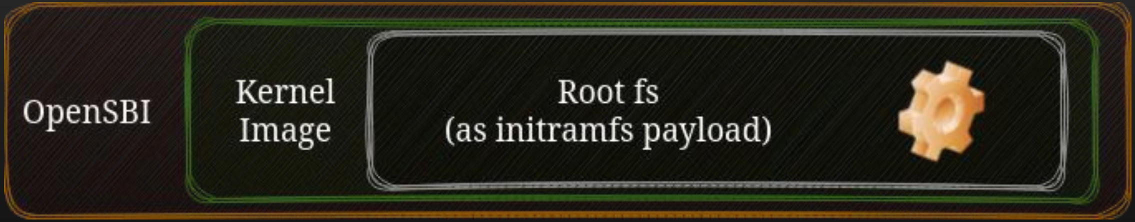


Step-by-step expansion of coverage

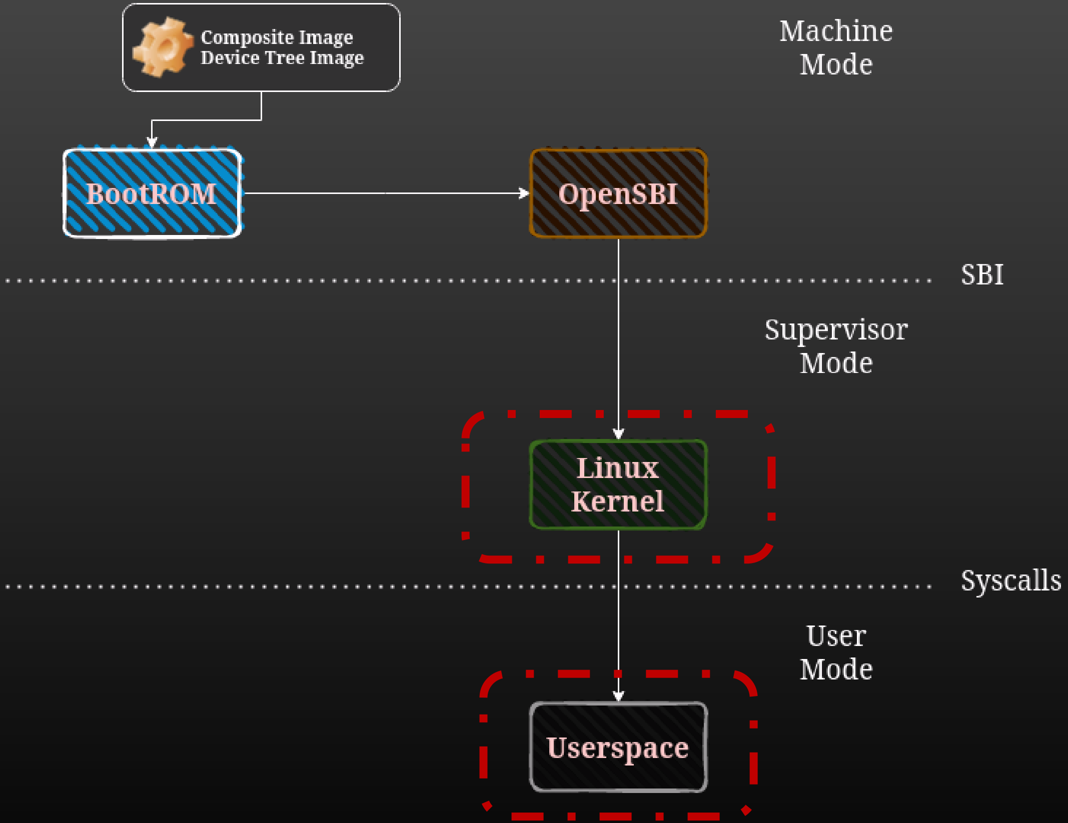
- Booting a full-blown Linux distro. greatly expands test coverage … and complexity
- Tracking down HW bugs in such a setup is a nightmare!
- We need a strategy to progressively expand test coverage

Simplifying the Linux boot process

- Use OpenSBI, a firmware implementation that also acts as FSBL
  - Get rid of SSBL and jump to Linux kernel directly
- Reduce number of external images
  - Kernel image as an OpenSBI payload
  - Root FS included as initramfs in the kernel image



Simplifying the Linux boot process



Simplifying the Linux kernel

- Start with a bare minimum kernel configuration
  - No networking, no storage, NOMMU
  - Limited functionality, single user
- Move on to more complex kernel configurations
  - With networking, storage, multiple users, …
- Finally, a full-blown kernel configuration
  - With systemd support and everything needed to boot a fully-featured Linux distro.

Simplifying userspace

- Start with a single process (busybox), statically linked
- Add more tools and networking support
  - e.g. iperf, ssh
- Use an off-the-shelf rootFS of a full-blown Linux distro.
  - e.g. Ubuntu

Why NOMMU Linux

- MMU is a common source of HW bugs in our experience
  - Microarchitectural bugs that are hard to reproduce in simple tests we previously did
  - Especially when we go multicore
- Why not go for a simple RTOS ? (e.g. FreeRTOS, Zephyr)
  - Using standard tools (e.g. busybox, iperf) would be harder (different syscall API)
  - Building the image would be more complicated (need to go through an SDK)
  - Usually support only M-mode/U-mode setups
  - Would be harder to compare behavior between MMU/NOMMU

NOMMU Linux basics

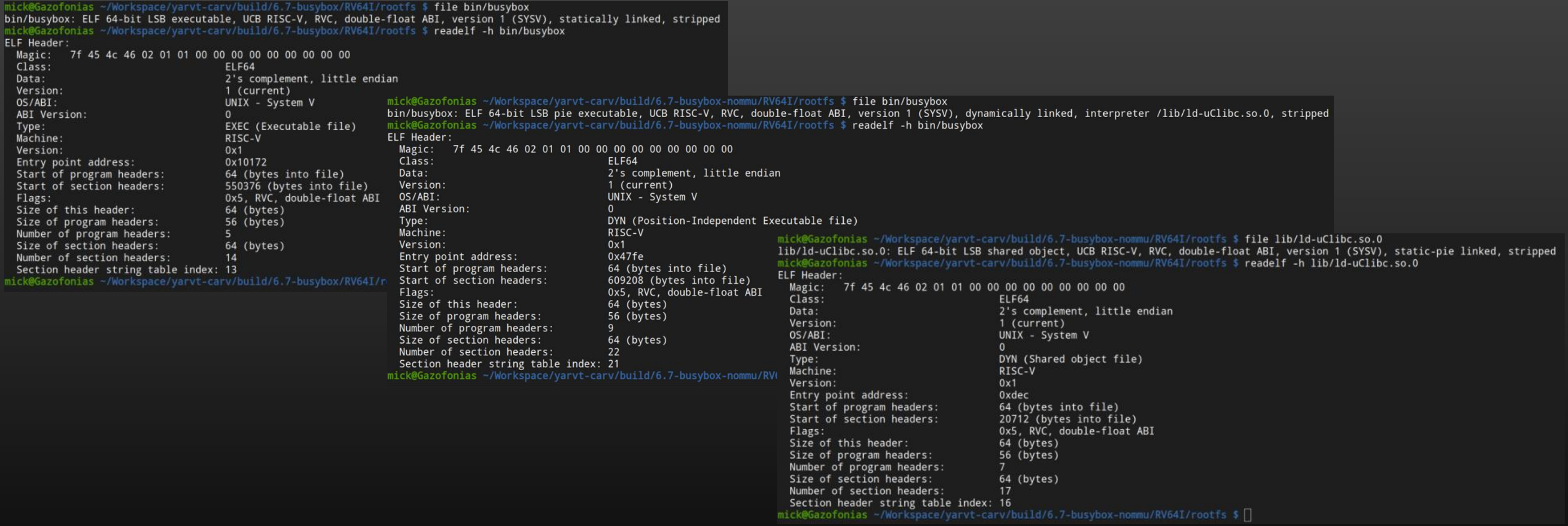
Part of mainline Linux kernel

- Different memory allocators: mm/nommu.c
- Limitations on mmap: Documentation/nommu-mmap.txt
  - No memory protection
  - No fork() support
    - fork() relies on COW, but vfork() is supported
  - No overcommit / lazy binding
  - No swap
  - No dynamic heap/stack
    - avoid using alloca(), brk(), sbrk(), use malloc()/free() instead
  - No MAP_SHARED on files
    - in general MAP_SHARED functionality is limited
  - No MAP_FIXED
  - Limitations on MAP_PRIVATE
    - no COW/paging
  - Excessive fragmentation, avoid large mappings

NOMMU Linux basics

- When MMU is available, the BINFMT_ELF loader is used to load executables / shared libraries.
- Without MMU, alternative loaders/binary formats are used
  - BINMFT_FLAT
    - Stripped down ELF (through elf2flt)
    - No dynamic loading (libld)
    - No shared libraries
    - Limitations on executable’s size
  - BINFMT_ELF_FDPIC
    - Position Independent (PIC/PIE) ELF, no ET_EXEC support
    - Support for shared libraries through function descriptors (FD)
    - Support for dynamic loading (libld)
    - May also be used when MMU is enabled
- Alternative toolchains also required
  - based on µClibc or musl

NOMMU Linux basics



```
mick@Gazofonias ~/Workspace/yarvt-carv/build/6.7-busybox/RV64I/rootfs $ file bin/busybox
bin/busybox: ELF 64-bit LSB executable, UCB RISC-V, RVC, double-float ABI, version 1 (SYSV), statically linked, stripped
mick@Gazofonias ~/Workspace/yarvt-carv/build/6.7-busybox/RV64I/rootfs $ readelf -h bin/busybox
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           RISC-V
  Version:                           0x1
  Entry point address:               0x10172
  Start of program headers:          64 (bytes into file)
  Start of section headers:          550376 (bytes into file)
  Flags:                             0x5, RVC, double-float ABI
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         5
  Size of section headers:           64 (bytes)
  Number of section headers:         14
  Section header string table index: 13
mick@Gazofonias ~/Workspace/yarvt-carv/build/6.7-busybox/RV64I/r
```

```
mick@Gazofonias ~/Workspace/yarvt-carv/build/6.7-busybox-nommu/RV64I/rootfs $ file bin/busybox
bin/busybox: ELF 64-bit LSB pie executable, UCB RISC-V, RVC, double-float ABI, version 1 (SYSV), dynamically linked, interpreter /lib/ld-uClibc.so.0, stripped
mick@Gazofonias ~/Workspace/yarvt-carv/build/6.7-busybox-nommu/RV64I/rootfs $ readelf -h bin/busybox
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Position-Independent Executable file)
  Machine:                           RISC-V
  Version:                           0x1
  Entry point address:               0x47fe
  Start of program headers:          64 (bytes into file)
  Start of section headers:          609208 (bytes into file)
  Flags:                             0x5, RVC, double-float ABI
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         9
  Size of section headers:           64 (bytes)
  Number of section headers:         22
  Section header string table index: 21
mick@Gazofonias ~/Workspace/yarvt-carv/build/6.7-busybox-nommu/RV
```

```
mick@Gazofonias ~/Workspace/yarvt-carv/build/6.7-busybox-nommu/RV64I/rootfs $ file lib/ld-uClibc.so.0
lib/ld-uClibc.so.0: ELF 64-bit LSB shared object, UCB RISC-V, RVC, double-float ABI, version 1 (SYSV), static-pie linked, stripped
mick@Gazofonias ~/Workspace/yarvt-carv/build/6.7-busybox-nommu/RV64I/rootfs $ readelf -h lib/ld-uClibc.so.0
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Shared object file)
  Machine:                           RISC-V
  Version:                           0x1
  Entry point address:               0xdec
  Start of program headers:          64 (bytes into file)
  Start of section headers:          20712 (bytes into file)
  Flags:                             0x5, RVC, double-float ABI
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         7
  Size of section headers:           64 (bytes)
  Number of section headers:         17
  Section header string table index: 16
mick@Gazofonias ~/Workspace/yarvt-carv/build/6.7-busybox-nommu/RV64I/rootfs $
```

Currently, RISC-V does not support static PIE.

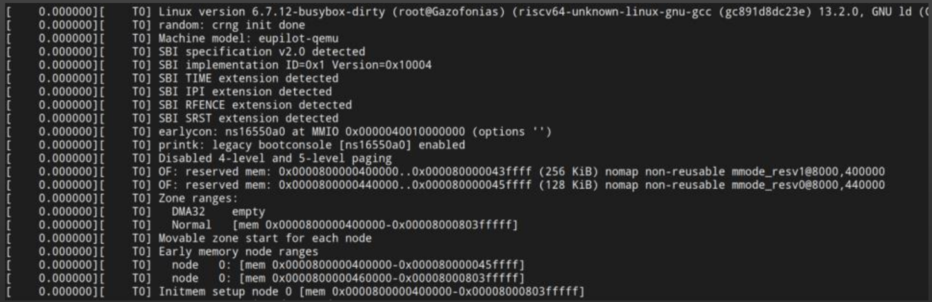
NOMMU Linux on RISC-V (kernel)

- Initial support added on Linux 5.5
  - Only M-mode/U-mode scenario
  - Mainly to support the Kendryte K210 that had a non-compliant MMU
- Almost declared deprecated on Feb. 2024
  - But after community feedback, it remains supported
    - New patches came up, and support keeps getting better
  - Support for running NOMMU Linux on S-mode has been added
    - Still needs further work though

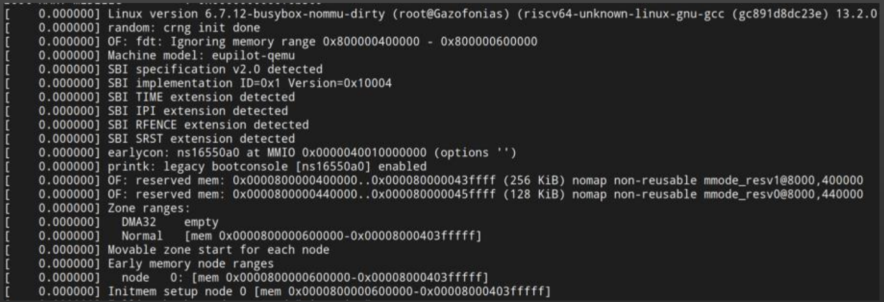
NOMMU Linux on RISC-V (userspace)

- FLAT binaries supported, but won’t work for us
  - Due to custom memory layout in our prototypes
- ELF psABI for FDPIC support is still WiP
  - But we can at least run busybox (64-bit)
- µClibc added support for RISC-V
  - Still no upstream toolchain, or support on crosstool-ng
  - We are working on it:
    - https://github.com/riscv-collab/riscv-gnu-toolchain/pull/1475
    - https://github.com/CARV-ICS-FORTH/riscv-gnu-toolchain/tree/uclibc
  - To replicate our setup with yarvt (Yet Another RISC-V Tool):
  - https://github.com/CARV-ICS-FORTH/yarvt/tree/riser

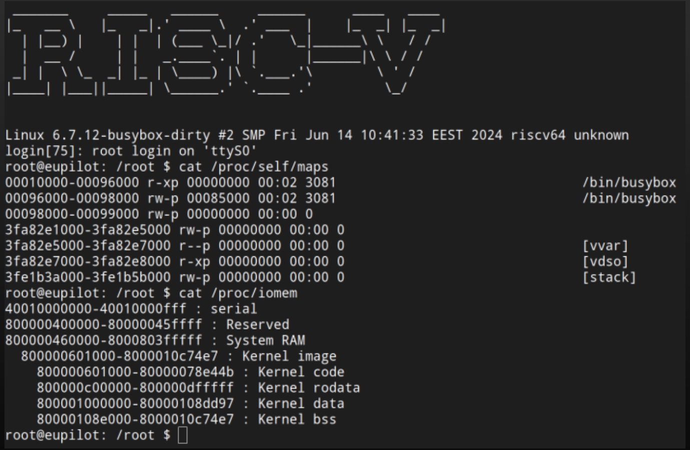
Testing MMU vs NOMMU



```
[    0.000000][    T0] Linux version 6.7.12-busybox-dirty (root@Gazofonias) (riscv64-unknown-linux-gnu-gcc (gc891d8dc23e) 13.2.0, GNU ld ((
[    0.000000][    T0] random: crng init done
[    0.000000][    T0] Machine model: eupilot-qemu
[    0.000000][    T0] SBI specification v2.0 detected
[    0.000000][    T0] SBI implementation ID=0x1 Version=0x10004
[    0.000000][    T0] SBI TIME extension detected
[    0.000000][    T0] SBI IPI extension detected
[    0.000000][    T0] SBI RFENCE extension detected
[    0.000000][    T0] SBI SRST extension detected
[    0.000000][    T0] earlycon: ns16550a0 at MMIO 0x0000040010000000 (options '')
[    0.000000][    T0] printk: legacy bootconsole [ns16550a0] enabled
[    0.000000][    T0] Disabled 4-level and 5-level paging
[    0.000000][    T0] OF: reserved mem: 0x0000800000400000..0x000080000043ffff (256 KiB) nomap non-reusable mmode_resv1@8000,400000
[    0.000000][    T0] OF: reserved mem: 0x0000800000440000..0x000080000045ffff (128 KiB) nomap non-reusable mmode_resv0@8000,440000
[    0.000000][    T0] Zone ranges:
[    0.000000][    T0]   DMA32    empty
[    0.000000][    T0]   Normal   [mem 0x0000800000400000-0x00008000803fffff]
[    0.000000][    T0] Movable zone start for each node
[    0.000000][    T0] Early memory node ranges
[    0.000000][    T0]   node   0: [mem 0x0000800000400000-0x000080000045ffff]
[    0.000000][    T0]   node   0: [mem 0x0000800000460000-0x00008000803fffff]
[    0.000000][    T0] Initmem setup node 0 [mem 0x0000800000400000-0x00008000803fffff]
```

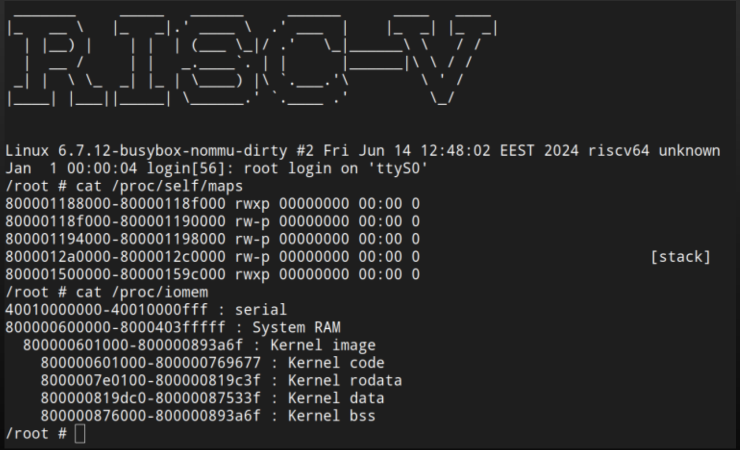


```
[    0.000000] Linux version 6.7.12-busybox-nommu-dirty (root@Gazofonias) (riscv64-unknown-linux-gnu-gcc (gc891d8dc23e) 13.2.0
[    0.000000] random: crng init done
[    0.000000] OF: fdt: Ignoring memory range 0x800000400000 - 0x800000600000
[    0.000000] Machine model: eupilot-qemu
[    0.000000] SBI specification v2.0 detected
[    0.000000] SBI implementation ID=0x1 Version=0x10004
[    0.000000] SBI TIME extension detected
[    0.000000] SBI IPI extension detected
[    0.000000] SBI RFENCE extension detected
[    0.000000] SBI SRST extension detected
[    0.000000] earlycon: ns16550a0 at MMIO 0x0000040010000000 (options '')
[    0.000000] printk: legacy bootconsole [ns16550a0] enabled
[    0.000000] OF: reserved mem: 0x0000800000400000..0x000080000043ffff (256 KiB) nomap non-reusable mmode_resv1@8000,400000
[    0.000000] OF: reserved mem: 0x0000800000440000..0x000080000045ffff (128 KiB) nomap non-reusable mmode_resv0@8000,440000
[    0.000000] Zone ranges:
[    0.000000]   DMA32    empty
[    0.000000]   Normal   [mem 0x0000800000600000-0x00008000403fffff]
[    0.000000] Movable zone start for each node
[    0.000000] Early memory node ranges
[    0.000000]   node   0: [mem 0x0000800000600000-0x00008000403fffff]
[    0.000000] Initmem setup node 0 [mem 0x0000800000600000-0x00008000403fffff]
```



```
 _______     _____   ______     ______       ____   ____
|_   __ \   |_   _|.' ____ \  .' ___  |     |_  _| |_  _|
  | |__) |    | |  | (___ \_|/ .'   \_|______ \ \   / /
  |  __ /     | |   _.____`. | |      |______| \ \ / /
 _| |  \ \_  _| |_ | \____) |\ `.___.'\         \ ' /
|____| |___||_____| \______.' `.____ .'          \_/


Linux 6.7.12-busybox-dirty #2 SMP Fri Jun 14 10:41:33 EEST 2024 riscv64 unknown
login[75]: root login on 'ttyS0'
root@eupilot: /root $ cat /proc/self/maps
00010000-00096000 r-xp 00000000 00:02 3081                               /bin/busybox
00096000-00098000 rw-p 00085000 00:02 3081                               /bin/busybox
00098000-00099000 rw-p 00000000 00:00 0
3fa82e1000-3fa82e5000 rw-p 00000000 00:00 0
3fa82e5000-3fa82e7000 r--p 00000000 00:00 0                              [vvar]
3fa82e7000-3fa82e8000 r-xp 00000000 00:00 0                              [vdso]
3fe1b3a000-3fe1b5b000 rw-p 00000000 00:00 0                              [stack]
root@eupilot: /root $ cat /proc/iomem
40010000000-40010000fff : serial
800000400000-80000045ffff : Reserved
800000460000-8000803fffff : System RAM
  800000601000-8000010c74e7 : Kernel image
    800000601000-80000078e44b : Kernel code
    800000c00000-800000dfffff : Kernel rodata
    800001000000-80000108dd97 : Kernel data
    80000108e000-8000010c74e7 : Kernel bss
root@eupilot: /root $ ▯
```



```
 _______     _____   ______     ______       ____   ____
|_   __ \   |_   _|.' ____ \  .' ___  |     |_  _| |_  _|
  | |__) |    | |  | (___ \_|/ .'   \_|______ \ \   / /
  |  __ /     | |   _.____`. | |      |______| \ \ / /
 _| |  \ \_  _| |_ | \____) |\ `.___.'\         \ ' /
|____| |___||_____| \______.' `.____ .'          \_/


Linux 6.7.12-busybox-nommu-dirty #2 Fri Jun 14 12:48:02 EEST 2024 riscv64 unknown
Jan  1 00:00:04 login[56]: root login on 'ttyS0'
/root # cat /proc/self/maps
800001188000-80000118f000 rwxp 00000000 00:00 0
80000118f000-800001190000 rw-p 00000000 00:00 0
800001194000-800001198000 rw-p 00000000 00:00 0
8000012a0000-8000012c0000 rw-p 00000000 00:00 0                              [stack]
800001500000-80000159c000 rwxp 00000000 00:00 0
/root # cat /proc/iomem
40010000000-40010000fff : serial
800000600000-8000403fffff : System RAM
  800000601000-800000893a6f : Kernel image
    800000601000-800000769677 : Kernel code
    8000007e0100-800000819c3f : Kernel rodata
    800000819dc0-80000087533f : Kernel data
    800000876000-800000893a6f : Kernel bss
/root # ▯
```

RISER: RISC-V for cloud services

https://riser-project.eu

Thank you for your attention. Questions and comments ?

Disclaimer:

“Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Health and Digital Executive Agency (HaDEA). Neither the European Union nor the granting authority can be held responsible for them.”

European Commission

Horizon Europe 2021-2027