Kernel Report

v5.19~6.4v

김동현(Austin Kim)
austindh.kim@gmail.com

Who am I?

- Profile (Linux kernel BSP engineer at LGE)
  - https://www.linkedin.com/in/austin-kim-김동현-638214147/
  - 유튜브: https://www.youtube.com/@schezokim
  - 블로그: https://austindhkim.tistory.com/

- Author
  - "시스템 소프트웨어 개발을 위한 Arm 아키텍처의 구조와 원리" – (2023년)
  - "디버깅을 통해 배우는 리눅스 커널의 구조와 원리" – (2021년)

New architecture: LoongArch (v5.19)

New architecture: LoongArch

- Commits

```
LoongArch: Add build infrastructure
fa96b57c149061f71a70bd6582d995f6424fbbf4

LoongArch: Add CPU definition headers
f2ac457a61389b7769aad8295027cbe0f91c5b80

LoongArch: Add atomic/locking headers
5b0b14e550a006b4d093619e7517923872bcc218

LoongArch: Add memory management
09cfefb7fa70c3af011b0db0a513fd80b2f18abc

LoongArch: Add process management
803b0fc5c3f2baa6e54978cd576407896f789b08

LoongArch: Add exception/interrupt handling
0603839b18f4fb3bffa82515efcf5b02084505ef
```

more features of RISC-V

more features of RISC-V

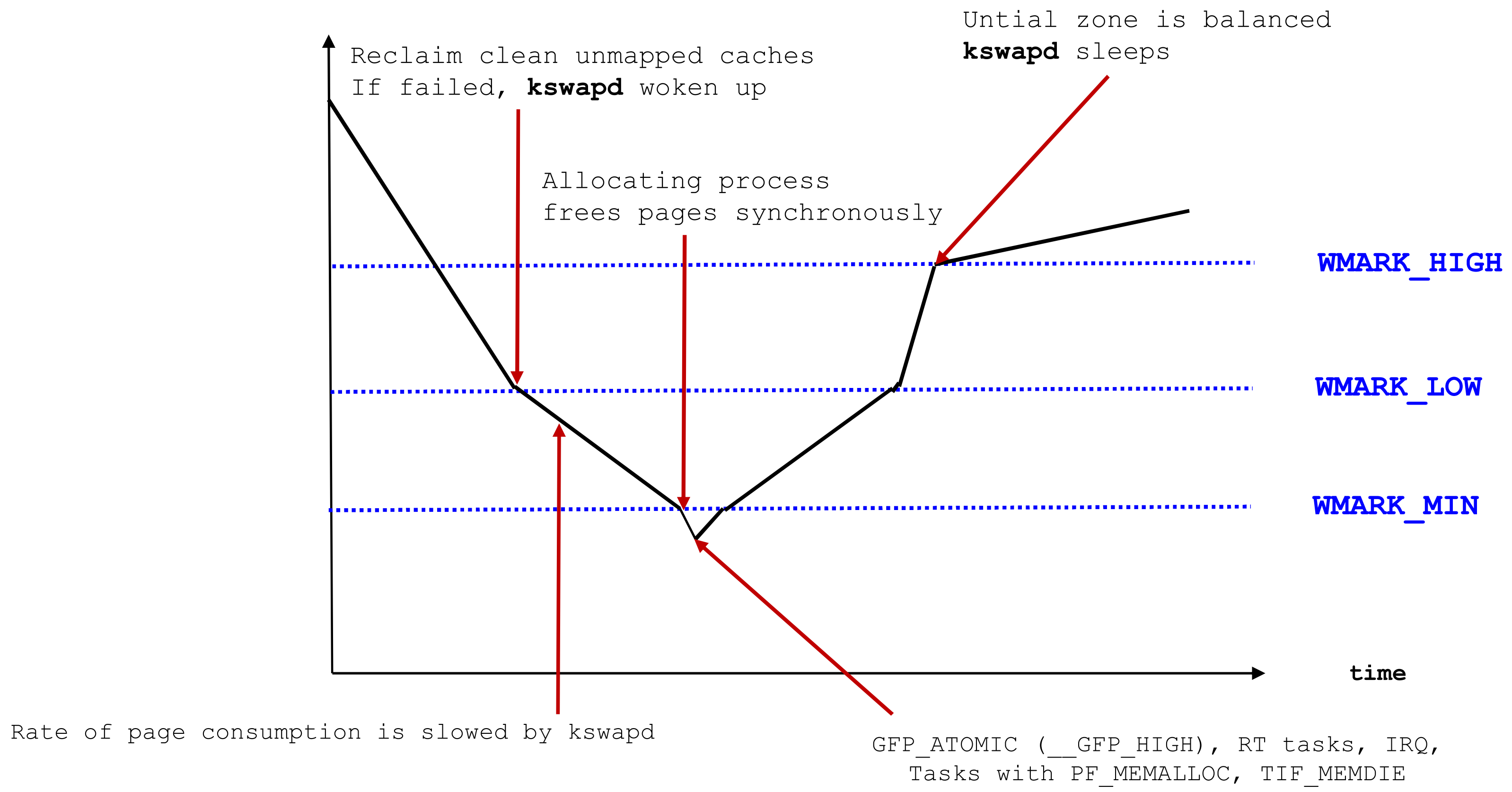
- Commits

```
RISC-V: Support for kexec_file on panic
8acea455fafaf2620b247de6c00774828b618a82

RISC-V: Add arch_crash_save_vmcoreinfo support
649d6b1019a2f243bc3a98cb85902a8ebf74289a
```

Proactive reclaim in memory control groups

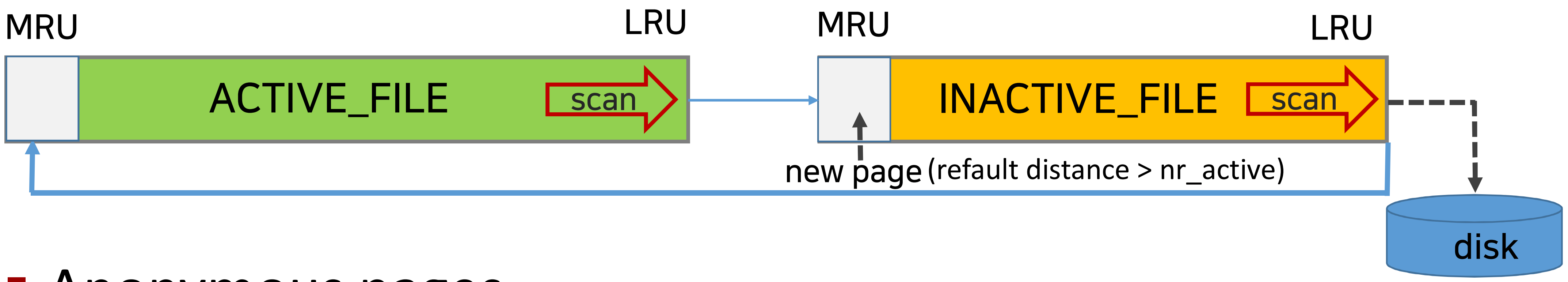
Background: page reclaim

- workflow of page reclaim

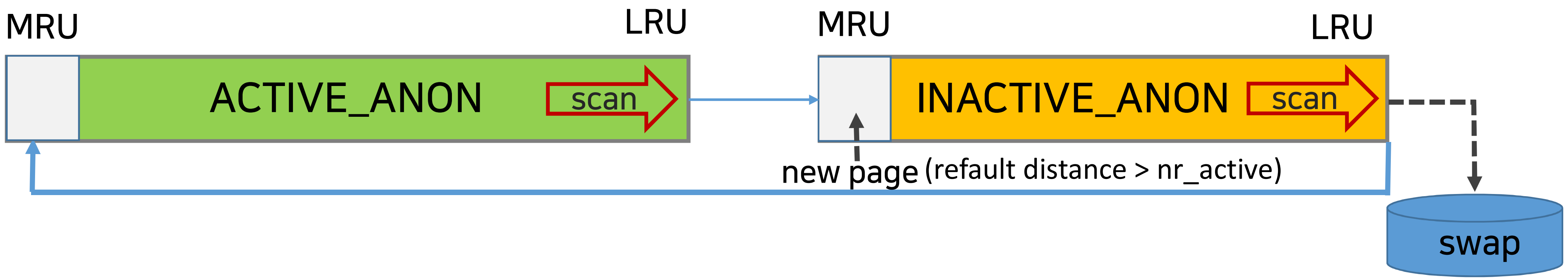

Reclaim clean unmapped caches
If failed, **kswapd** woken up
Untial zone is balanced
kswapd sleeps
Allocating process
frees pages synchronously
WMARK_HIGH
WMARK_LOW
WMARK_MIN
time
Rate of page consumption is slowed by kswapd
GFP_ATOMIC (__GFP_HIGH), RT tasks, IRQ,
Tasks with PF_MEMALLOC, TIF_MEMDIE

Background: Page reclaim (2nd chance algorithm)

- page cache



- Anonymous pages



Proactive reclaim in memory control groups

- Benefits of a user space reclaimer:
  - More flexible on who should be charged for the cpu of the memory reclaim. For proactive reclaim, it makes more sense to be centralized.
  - More flexible on dedicating the resources (like cpu). The memory overcommit controller can balance the cost between the cpu usage and the memory reclaimed.
  - Provides a way to the applications to keep their LRUs sorted, so, under memory pressure better reclaim candidates are selected. This also gives more accurate and uptodate notion of working set for an   application.

https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=94968384dde15d48263bfc59d280cd71b1259d8c

Proactive reclaim in memory control groups

- Provide interface for memory control groups to trigger memory reclaim on a memory cgroup

```
+   memory.reclaim
+          A write-only nested-keyed file which exists for all cgroups.
+
+          This is a simple interface to trigger memory reclaim in the
+          target cgroup.
+
+          This file accepts a single key, the number of bytes to reclaim.
+          No nested keys are currently supported.
+
+          Example::
+
+            echo "1G" > memory.reclaim
+
+          The interface can be later extended with nested keys to
+          configure the reclaim behavior. For example, specify the
+          type of memory to reclaim from (anon, file, ..).
```

https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=94968384dde15d48263bfc59d280cd71b1259d8c

Proactive reclaim in memory control groups

- Patch snippet

```
+static ssize_t memory_reclaim(struct kernfs_open_file *of, char *buf,
+                                    size_t nbytes, loff_t off)
+{
...
+
+         buf = strstrip(buf);
+         err = page_counter_memparse(buf, "", &nr_to_reclaim);
+         if (err)
+                   return err;
+
+         while (nr_reclaimed < nr_to_reclaim) {
...
+                   if (!nr_retries)
+                             lru_add_drain_all();
+
+                   reclaimed = try_to_free_mem_cgroup_pages(memcg,
+                                                       nr_to_reclaim - nr_reclaimed,
+                                                       GFP_KERNEL, true);
+
+                   if (!reclaimed && !nr_retries--)
+                             return -EAGAIN;
+
+                   nr_reclaimed += reclaimed;
+         }
```

Runtime verification system (v6.0)

Background: Runtime verification

- About Linux from 'functional safety' point of view
    - Linux kernel cannot meet specification of ISO26262(functional safety)
- * ELISA (Enabling Linux in Safety Application) is announced.
- Runtime verification is introduced as a way to launch functional safety of Linux kernel in automotive area.

https://elisa.tech/

Introduction to Runtime verification system

- What is Runtime Verification (RV)?
  - A lightweight (yet rigorous) method that complements classical exhaustive verification techniques (such as *model checking* and *theorem proving*) with a more practical approach for complex systems.
- Motivation of Runtime Verification (RV)?
  - Instead of relying on a fine-grained model of a system (e.g., a re-implementation a instruction level), RV works by analyzing the trace of the system's actual execution, comparing it against a formal specification ofthe system behavior.
- The main advantage
  - RV can give precise information on the runtime behavior of the monitored system, without the pitfalls of developing models that require a re-implementation of the entire system in a modeling language.

https://lwn.net/Articles/857862/

Big picture

- Diagram of RV

```
Linux   +---- RV Monitor ----------------------------------+ Formal
 Realm  |                                                  |  Realm
 +-------------------+     +----------------+     +-----------------+
 |   Linux kernel    |     |     Monitor    |     |    Reference    |
 |     Tracing       |  -> |   Instance(s)  | <-  |       Model     |
 | (instrumentation) |     | (verification) |     | (specification) |
 +-------------------+     +----------------+     +-----------------+
        |                          |                        |
        |                          V                        |
        |                     +----------+                  |
        |                     | Reaction |                  |
        |                     +--+--+--+-+                  |
        |                        |  |  |                    |
        |                        |  |  +-> trace output ?   |
        +------------------------|--|----------------------+
                                 |  +----> panic ?
                                 +-------> <user-specified>
```

How RV works? - 1

- 1) Model;
  - Define the feature of specific subsystem(e.g: preemption, task wakingup) using 'state machine'
  - Model is described in *.dot which can be generated as C code.
  - The generated code can be dynamically loaded as module via 'dot2c' utility.

How RV works? - 2

- 2) RV runtime;
  - When kernel module is initialized, it will register ftrace events which is being hooked.

```
https://elixir.bootlin.com/linux/v6.1-rc4/source/kernel/trace/rv/monitors/wip/wip.c
static int enable_wip(void)
{
...
   rv_attach_trace_probe("wip", preempt_enable, handle_preempt_enable);
   rv_attach_trace_probe("wip", sched_waking, handle_sched_waking);
   rv_attach_trace_probe("wip", preempt_disable, handle_preempt_disable);
...
```

  - 2nd parameter: name of ftrace event
  - 3rd parameter: The function to be hooked and executed

How RV works? - 3

- 3) Monitor
  - wip(wakeup preemptivie): Check weather wakeup event may be traced with preemption disabled
  - wwnr(Per task wakeup while not running): Check weather wakeup event may be traced if the state of process is not TASK_RUNNING

(False injection code)

```
set_current_state(TASK_UNINTERRUPTIBLE);
        // Perform task wakeup after IRQ
schedule();
```

[Appendix] Runtime verification

- The runtime verification subsystem
  - https://lwn.net/Articles/857862/

- Formal Verification Made Easy (and fast!) - Daniel Bristot de Oliveira, Red Hat
  - https://www.youtube.com/watch?v=BfTuEHafNgg

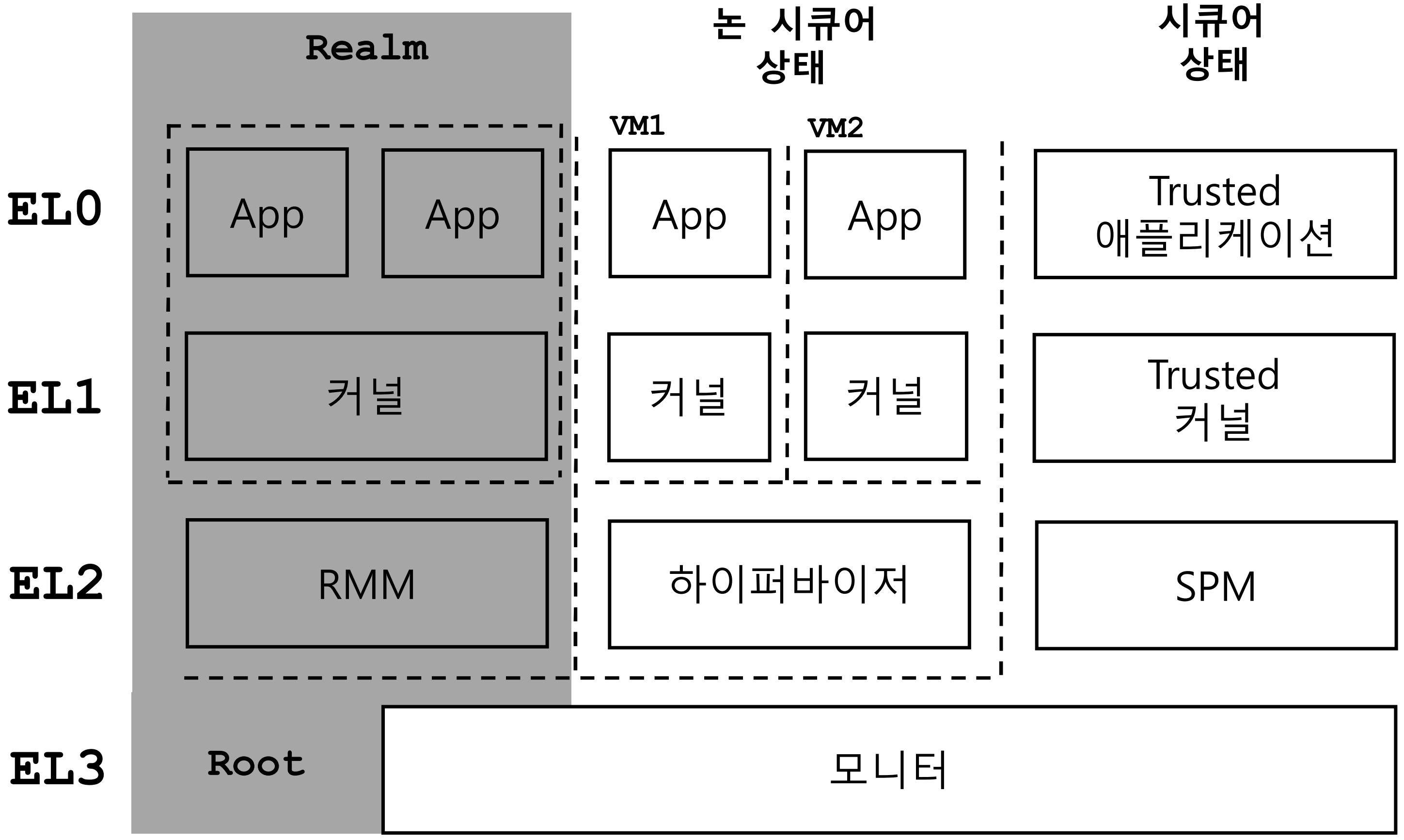
AMD SEV-SNP
(v5.19)

Motivation to research AMD SEV-SNP

- Mainstream security activity over CPU architecture
  - Prevent Hypervisor based threats
  - Linux kernel is major guest OS embedded all hypervisors
  - Necessary to know what is discussed and upstreamed in x86, AMD, Arm

Similar Security Enhancement in Arm Architecture

- Armv9 introduces the CCA for confidential computing environment



Pull request for AMD Secure Nested Paging in v5.19

- https://lore.kernel.org/lkml/YotXilVhT2BZHZ5R@zn.tnic/

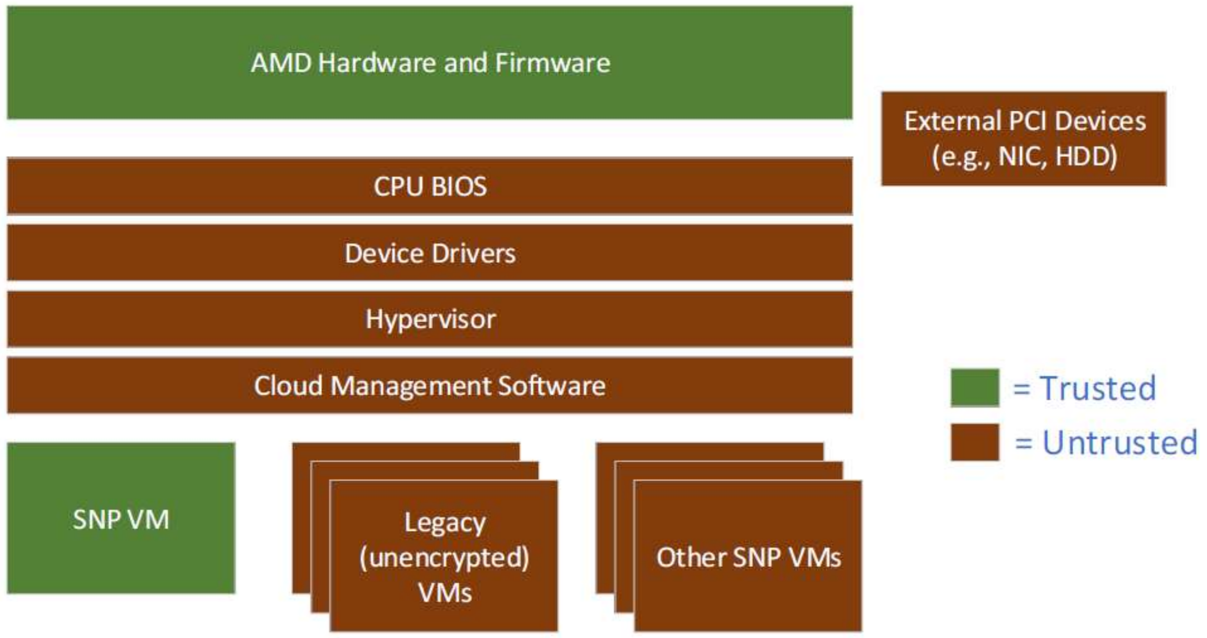
```
AMD SEV-SNP support

Add to confidential guests the necessary memory integrity protection
against malicious hypervisor-based attacks like data replay, memory
remapping and others, thus achieving a stronger isolation from the
hypervisor.

At the core of the functionality is a new structure called a reverse
map table (RMP) with which the guest has a say in which pages get
assigned to it and gets notified when a page which it owns, gets
accessed/modified under the covers so that the guest can take an
appropriate action.

In addition, add support for the whole machinery needed to launch a SNP
guest, details of which is properly explained in each patch.

And last but not least, the series refactors and improves parts of the
previous SEV support so that the new code is accomodated properly and
not just bolted on.
```

AMD Secure Encrypted Virtualization: Feature Overview

- Secure Encrypted Virtualization (SEV)
  - Provides Encryption of Guest Memory
- Encrypted State (SEV-ES)
  - Encrypts the register state of the VCPUs and make it inaccessible to the hypervisor.
- Secure Nested Paging (SEV-SNP)
  - Introduces a page ownership model between Hypervisor and Virtual Machine
  - Hypervisor cannot access or remap guest memory without the guest noticing it.

Background for AMD-SEV (Secure Encrypted Virtualization)

- Threat model



  - All other software components and PCI devices are treated as fully untrusted
  - Typical hypervisor is assumed to be malicious potentially conspiring
  - The hypervisor is not believed to be 100% secure
  - Designed to protect against additional threat

Rust for Linux (v6.1)

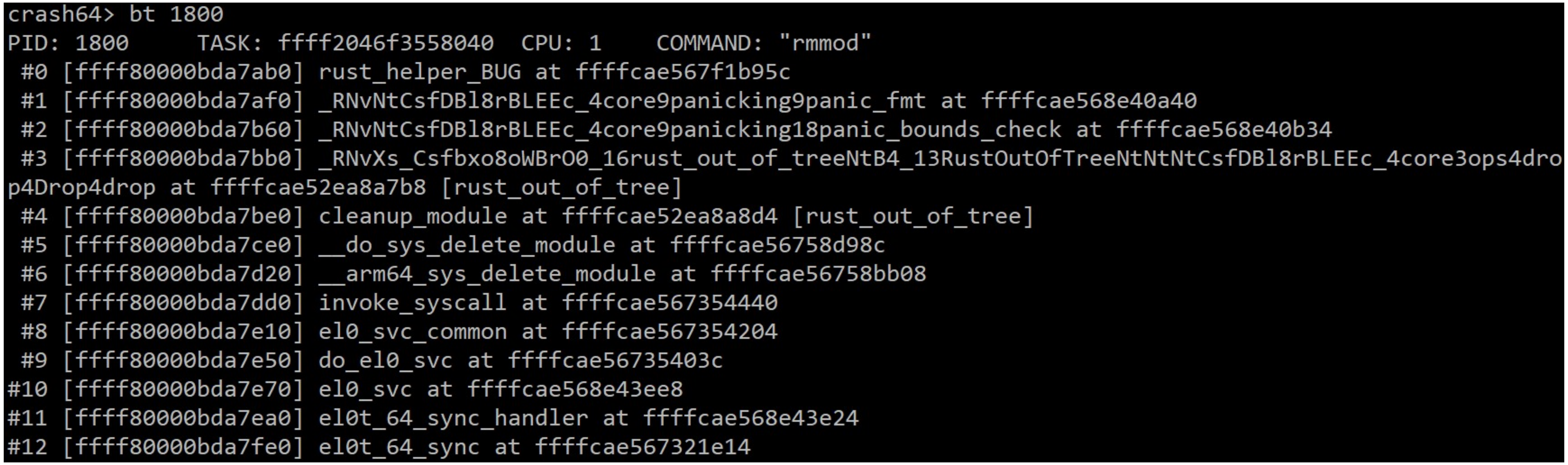
Kernel panic signature in Rust module

- Kernel log (Armv8)

```
[  203.720509] ------------[ cut here ]------------
[  203.720516] kernel BUG at rust/helpers.c:45!
[  203.721348] irq event stamp: 1613175
...
[  203.742154] CPU: 1 PID: 1800 Comm: rmmod Kdump: loaded Tainted: G           OE      6.2.0+ #12
[  203.744616] Hardware name: QEMU QEMU Virtual Machine, BIOS 0.0.0 02/06/2015
[  203.745908] pstate: 61400005 (nZCv daif +PAN -UAO -TCO +DIT -SSBS BTYPE=--)
[  203.747202] pc : rust_helper_BUG+0x10/0x14
[  203.748184] lr : rust_begin_unwind+0x5c/0x60
[  203.748875] sp : ffff80000bda7a60
...
[  203.760944] Call trace:
[  203.761414]  rust_helper_BUG+0x10/0x14
[  203.762014]  _RNvNtCsfDBl8rBLEEc_4core9panicking9panic_fmt+0x38/0x3c
[  203.762980]  _RNvNtCsfDBl8rBLEEc_4core9panicking18panic_bounds_check+0x54/0x58
[  203.763891]
_RNvXs_Csfbxo8oWBrO0_16rust_out_of_treeNtB4_13RustOutOfTreeNtNtNtCsfDBl8rBLEEc_4core3ops4drop4Drop4drop+0x248/0
x27c [rust_out_of_tree 9aec53abe8750236e4958f6759285c63adae465f]
[  203.766212]  cleanup_module+0x2c/0x7c [rust_out_of_tree 9aec53abe8750236e4958f6759285c63adae465f]
[  203.767274]  __do_sys_delete_module+0x254/0x378
[  203.767956]  __arm64_sys_delete_module+0x30/0x44
...
[  203.770934]  el0t_64_sync_handler+0x88/0xf8
[  203.771823]  el0t_64_sync+0x1a8/0x1ac
```

Kernel panic signature in Rust module

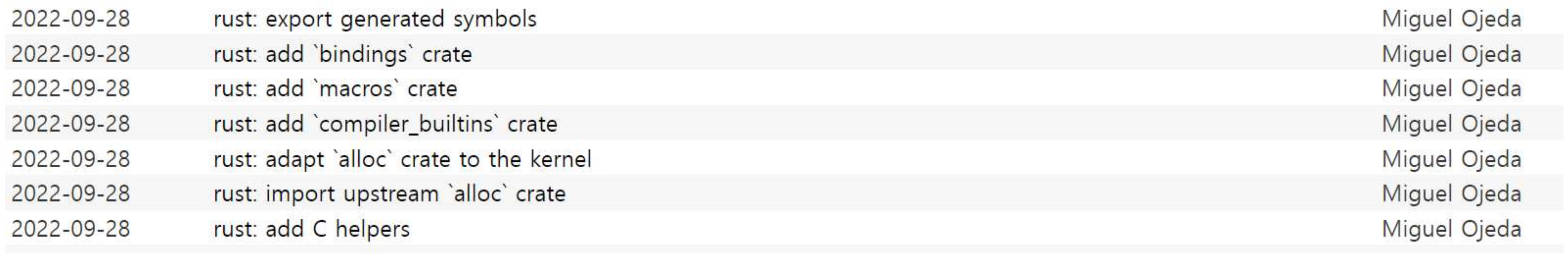
- crash utility signature



```
crash64> bt 1800
PID: 1800     TASK: ffff2046f3558040  CPU: 1    COMMAND: "rmmod"
 #0 [ffff80000bda7ab0] rust_helper_BUG at ffffcae567f1b95c
 #1 [ffff80000bda7af0] _RNvNtCsfDBl8rBLEEc_4core9panicking9panic_fmt at ffffcae568e40a40
 #2 [ffff80000bda7b60] _RNvNtCsfDBl8rBLEEc_4core9panicking18panic_bounds_check at ffffcae568e40b34
 #3 [ffff80000bda7bb0] _RNvXs_Csfbxo8oWBrO0_16rust_out_of_treeNtB4_13RustOutOfTreeNtNtNtCsfDBl8rBLEEc_4core3ops4dro
p4Drop4drop at ffffcae52ea8a7b8 [rust_out_of_tree]
 #4 [ffff80000bda7be0] cleanup_module at ffffcae52ea8a8d4 [rust_out_of_tree]
 #5 [ffff80000bda7ce0] __do_sys_delete_module at ffffcae56758d98c
 #6 [ffff80000bda7d20] __arm64_sys_delete_module at ffffcae56758bb08
 #7 [ffff80000bda7dd0] invoke_syscall at ffffcae567354440
 #8 [ffff80000bda7e10] el0_svc_common at ffffcae567354204
 #9 [ffff80000bda7e50] do_el0_svc at ffffcae56735403c
#10 [ffff80000bda7e70] el0_svc at ffffcae568e43ee8
#11 [ffff80000bda7ea0] el0t_64_sync_handler at ffffcae568e43e24
#12 [ffff80000bda7fe0] el0t_64_sync at ffffcae567321e14
```

Commit ‘linux for Rust’

- major commit



2022-09-28	rust: export generated symbols	Miguel Ojeda
2022-09-28	rust: add `bindings` crate	Miguel Ojeda
2022-09-28	rust: add `macros` crate	Miguel Ojeda
2022-09-28	rust: add `compiler_builtins` crate	Miguel Ojeda
2022-09-28	rust: adapt `alloc` crate to the kernel	Miguel Ojeda
2022-09-28	rust: import upstream `alloc` crate	Miguel Ojeda
2022-09-28	rust: add C helpers	Miguel Ojeda

https://git.kernel.org/pub/scm/linux/kernel/git/next/linux-next.git/log/?qt=author&q=Miguel+Ojeda

Programming Language for Linux Kernel

- Linux kernel is written in C since from the scratch
- Around 1997, C++ was only considered
- “Why GIT is implemented in C?”
  - Torvalds said “C++ is a horrible language”
  - Conservative approaches for new language

Rust Language history

- 2006: Under development Mozilla Research

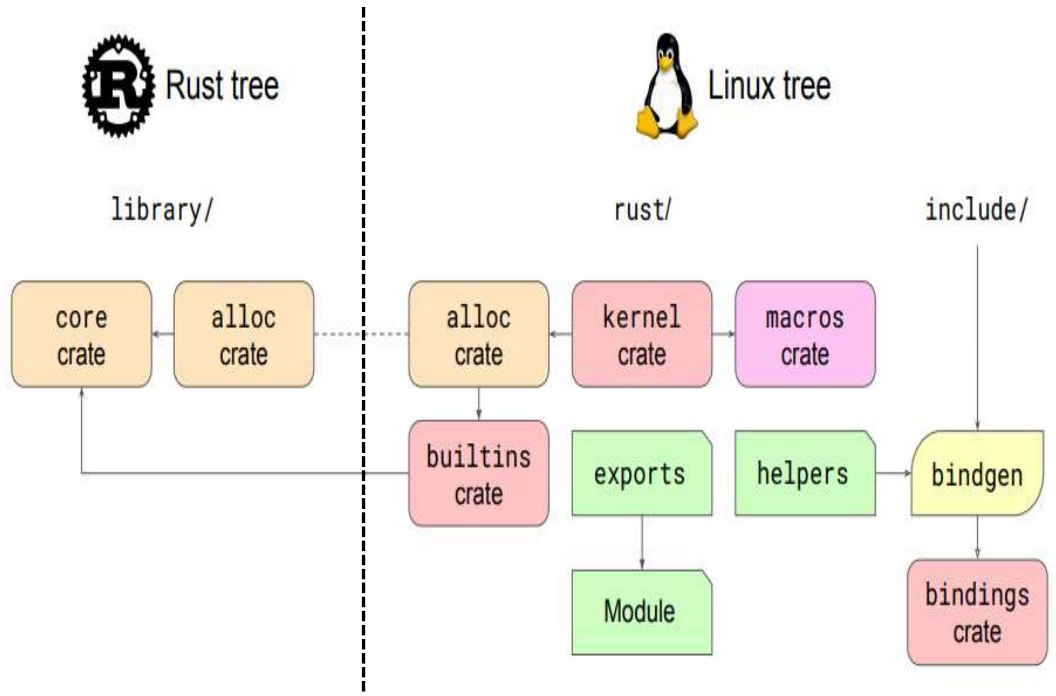
- 2020: Rust Foundation was announced
  - Mozilla, AWS, Huawei, Google, Microsoft

- 2022(October): Implementation for Rust for Linux was approved by Torvalds
  - Linux 6.1 will have direct support for Rust code
  - The option of using Rust for new code is added

Background to understand Rust for Linux

- Rust Crate
- Kernel APIs
- Crate kernel

Implementation of Rust for Linux, Component View



- alloc crate
  - copy of Rust tree, because some modifications are required
  - Eventually it will be removed
- bindgen
  - Tool that generates interfaces for existing C code

Rust Crate

- The smallest unit of code that the Rust compiler considers at a time
- Binary Crate vs Library Crate
  - Binary Crate : programs can be complied to an executable, e.g. command line program or server
  - Library Crate : No main function, provides functionality intended to be shared, generally referred as “crate”
- Core/Alloc Library Create

Core Crate

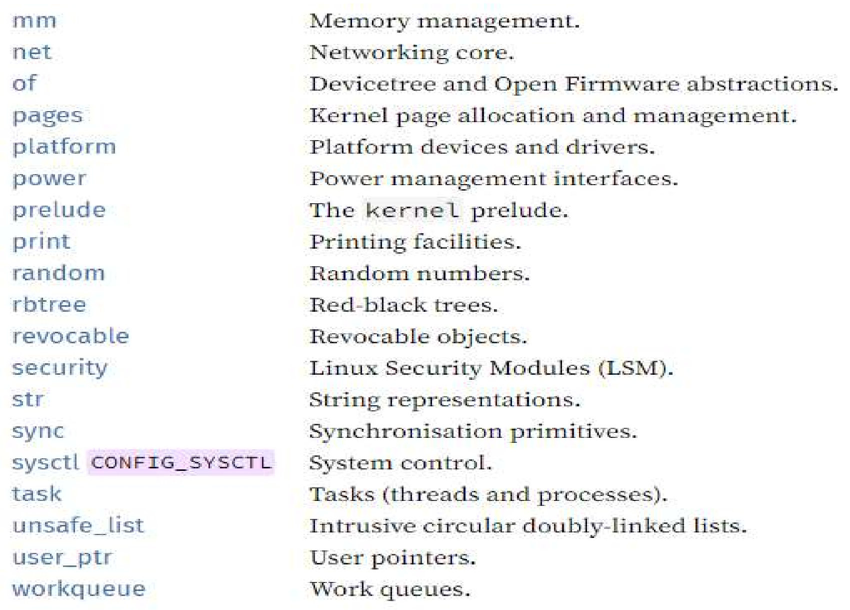
- Intrinsic and primitive building blocks of all Rust code
- Dependency-free foundation and Minimal
  - It links to no upstream libs, no system libs, no libc
  - It is not aware heap, concurrency or I/O
- Modules in core crate
  - f32, f64, i8, i16, i32, i64, u8, u16, u32, u64 : type and operations
  - char, str, slice : char and string manipulation
- Macros in core crate
  - assert, panic, unimplemented, unreachable

Alloc Crate

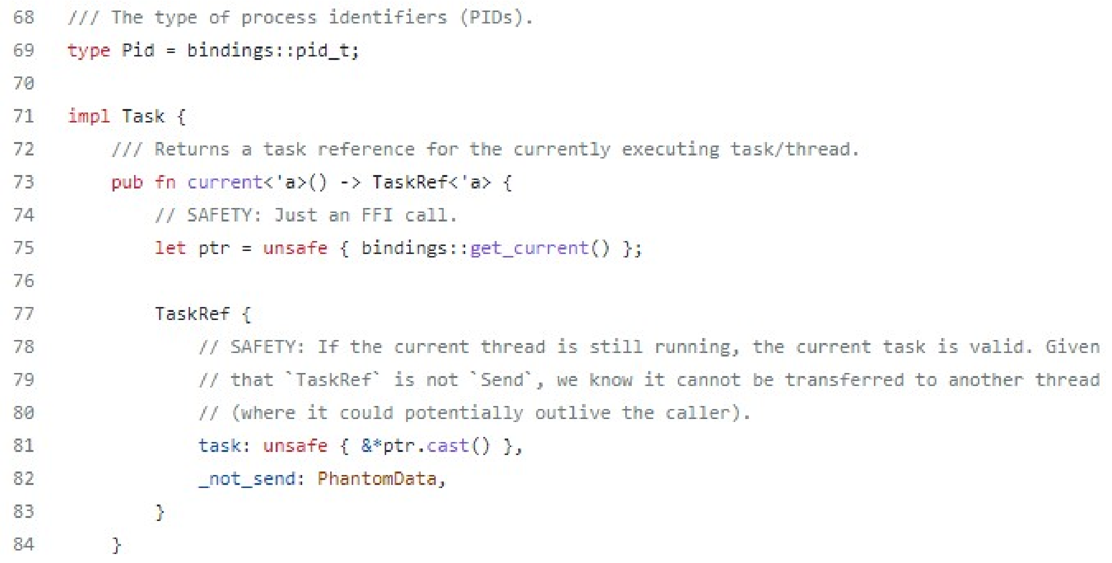
- Smart pointers and collections for managing heap-allocated values
- Heap interfaces modules
  - alloc module defines the low-level interfaces to the default global allocator
- Three types of pointer modules
  - Boxed : smart pointer type, there can only be own owner of a Box
  - Rc : reference-counted pointer, non-threadsafe, for sharing memory
  - Arc : reference-counted pointer, threadsafe

The Kernel Crate

- Kernel APIs that have been ported/wrapped for usage by Rust code in the kernel
  - i.e. all of the Rust code in the kernel depends on core, alloc and kernel crate
  - Modules in kernel crate



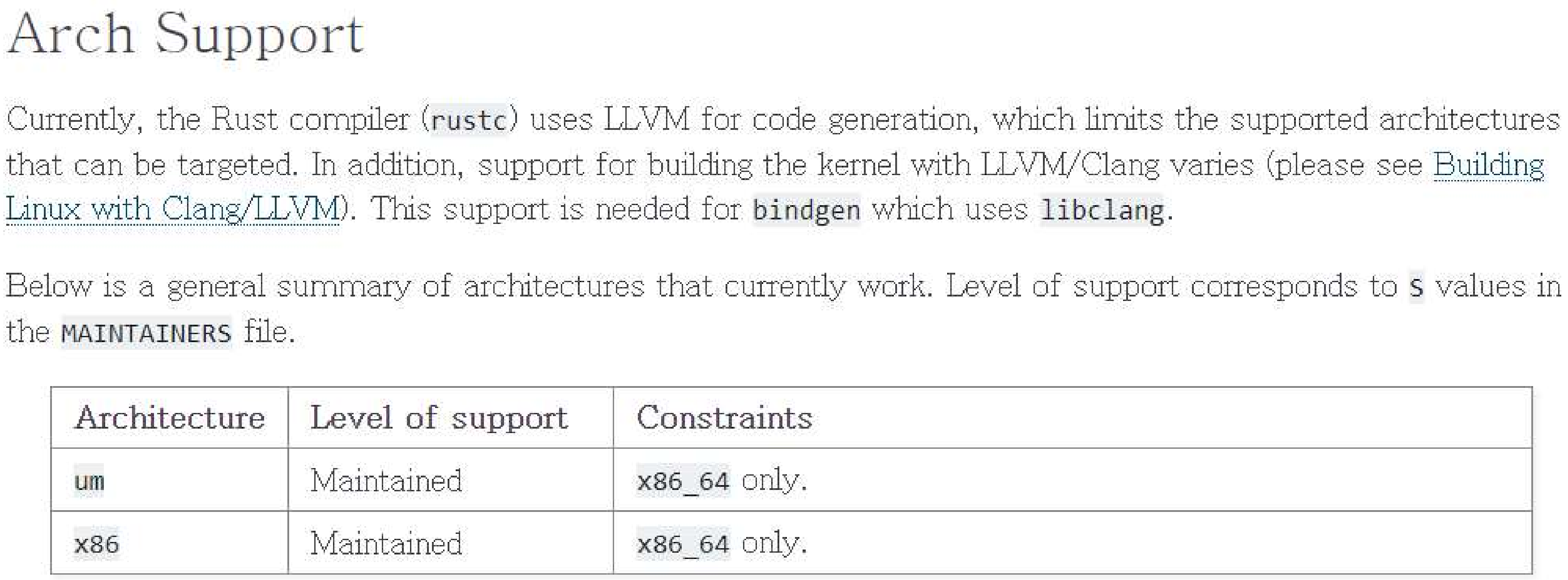
mm	Memory management.
net	Networking core.
of	Devicetree and Open Firmware abstractions.
pages	Kernel page allocation and management.
platform	Platform devices and drivers.
power	Power management interfaces.
prelude	The `kernel` prelude.
print	Printing facilities.
random	Random numbers.
rbtree	Red-black trees.
revocable	Revocable objects.
security	Linux Security Modules (LSM).
str	String representations.
sync	Synchronisation primitives.
sysctl CONFIG_SYSCTL	System control.
task	Tasks (threads and processes).
unsafe_list	Intrusive circular doubly-linked lists.
user_ptr	User pointers.
workqueue	Work queues.



```rust
/// The type of process identifiers (PIDs).
type Pid = bindings::pid_t;

impl Task {
    /// Returns a task reference for the currently executing task/thread.
    pub fn current<'a>() -> TaskRef<'a> {
        // SAFETY: Just an FFI call.
        let ptr = unsafe { bindings::get_current() };

        TaskRef {
            // SAFETY: If the current thread is still running, the current task is valid. Given
            // that `TaskRef` is not `Send`, we know it cannot be transferred to another thread
            // (where it could potentially outlive the caller).
            task: unsafe { &*ptr.cast() },
            _not_send: PhantomData,
        }
    }
```

Rust for Linux, Arch support

- <Documents/Rust/Arch Support from 6.1.0-rc3>



Arch Support

Currently, the Rust compiler (`rustc`) uses LLVM for code generation, which limits the supported architectures that can be targeted. In addition, support for building the kernel with LLVM/Clang varies (please see Building Linux with Clang/LLVM). This support is needed for `bindgen` which uses `libclang`.

Below is a general summary of architectures that currently work. Level of support corresponds to `S` values in the `MAINTAINERS` file.

Architecture	Level of support	Constraints
`um`	Maintained	`x86_64` only.
`x86`	Maintained	`x86_64` only.