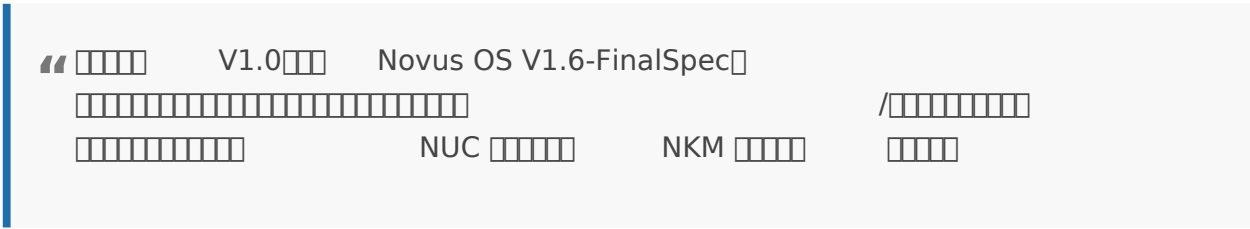


Novus OS Debug 00000

V1.0

Novus OS Debug 00000



0. 00000

00000000

- 00 0 C 0000
- 00000
- 000 / 0000
- 0000 000000000000 CAP00000000
- 000 000000000000000000000000
- 000 000000
- 00000 000000000000
- 0000 000000000
- 00000 00000000 /00000
- 0000 000000
- 00 Bug00000

1. 0000000000000000

值	宏名	类型	说明
0	NOVUS_OK	int	成功
-1	NOVUS_EINVAL	int	参数 NULL 或 0 无效
-2	NOVUS_ENOMEM	int	内存不足
-3	NOVUS_ENOTFOUND	int	找不到文件或设备
-4	NOVUS_EPERM	int	权限不足，cap_check() 返回 false
-5	NOVUS_EBUSY	int	设备忙
-6	NOVUS_EIO	IO 错误	IO 错误，PCI 设备
-7	NOVUS_ECORRUPT	int	SHA256 校验失败，CRC32 校验失败
-8	NOVUS_EMODULE	int	模块不存在，ABI 不兼容
-9	NOVUS_EFULL	int	IPC 队列满，WAL 写满
-10	NOVUS_EAGAIN	int	资源暂时不可用
-11	NOVUS_EOVERFLOW	int	溢出错误
-12	NOVUS_EDEP	int	依赖项未满足

2. SMP / AB 子系统

2.1 smp_shmem_write

```
void smp_shmem_write(uint32_t offset, uint64_t val);
```

- 调用者必须在 SMP 模式下
- 调用者必须是 A 进程 (BSP)
- 调用者必须持有锁
 - 调用者必须持有 A 进程的锁
 - 调用者必须调用 smp_shmem_lock() 获取锁
 - 调用者必须持有锁
- 调用者必须在

参数	返回值	错误码	备注
----	-----	-----	----

- `uint64_t`
- IPI `uint64_t` AP
- `bsp_state` `uint8_t` 2
- `cli; hlt;` `uint64_t`
- `uint64_t` BSP `uint64_t`
- `uint64_t` `uint64_t` "Acore entering sleep"

2.5 `smp_wake_ap`

```
int smp_wake_ap(uint64_t ap_id);
```

- `uint64_t` `uint64_t`
 - `bsp_state` `uint8_t` 2
 - `uint64_t` `sfence`
 - AP `uint64_t`
- `uint64_t` `uint64_t` `ap_id = AP` `uint64_t`
- `uint64_t` `uint64_t` IPI `uint64_t` 0x80
- `uint64_t` `uint64_t` 0 `uint64_t` -ENOMEM `uint64_t`
- `uint64_t` `uint64_t` `ap_ready` `uint64_t` 200ms → `uint64_t`

2.6 `smp_ap_signal_ready`

```
void smp_ap_signal_ready(void);
```

- `uint64_t` `uint64_t` AP `uint64_t` GDT/IDT/`uint64_t`
- `uint64_t` `uint64_t` `ap_ready = 1` `uint64_t` BSP `uint64_t`

3. `uint64_t`

3.1 `mm_alloc`

```
void* mm_alloc(size_t pages);
```

- `uint64_t` `uint64_t`
- `uint64_t` `uint64_t`
 - `uint64_t` `uint64_t` M1 `uint64_t`
 - `uint64_t` `uint64_t` GFP_ATOMIC `uint64_t`
- `uint64_t` `uint64_t`

- 返回 0 表示成功
- 返回非0表示失败
 - -EINVAL 无效的 va/pa/len
 - -ENOMEM 内存不足
- 成功时返回 CR3 和 TLB 信息
- **W^X** 标志位 flags 中的 WRITE 和 EXEC 不能同时为1，否则返回 -EINVAL

3.4 vm_unmap

```
int vm_unmap(uint64_t va, uint64_t len);
```

- 成功时返回 0
- 失败时返回非0

3.5 oom_scan

```
int oom_scan(proc_info_t* out);
```

- 返回 0 表示成功
- 返回非0表示失败
- 成功时返回 proc_info_t 结构体

```
score = rss_bytes * front_weight * age_weight * type_weight
front_weight: 0=1.8, 1=1.0
age_weight:   <30s=1.0, 30s~5min=0.7, >5min=0.4
type_weight:  0=1.0, 1=0.2
```

- 成功时返回 proc_info_t 结构体
- 失败时返回非0
- 成功时返回 proc_info_t 结构体
 - 返回 0 表示成功
 - 返回非0表示失败
 - 成功时返回 proc_info_t 结构体
 - 失败时返回非0
- 成功时返回 proc_info_t 结构体

3.6 oom_reclaim

```
int oom_reclaim(uint32_t level);
```

- 返回 0 表示成功
- 返回非0表示失败
- 成功时返回 0
- 失败时返回非0

- `sys.proc.spawn` → `sys.proc.spawn` → `NovusFS`
- `sys.proc.spawn` → `NovusFS`

4. `sys.proc.spawn`

4.1 `sys.proc.spawn`

```
int sys_proc_spawn(const char* path, uint64_t caps, pid_t* out_pid);
```

- `sys.proc.spawn`
 - `sys.proc.spawn` `CAP`
 - `path` `.nuc`
- `sys.proc.spawn`

Parameter	Default Value
<code>path</code>	<code>.nuc</code> <code>"/system/bin/novus-settings"</code>
<code>caps</code>	<code>CAP</code> <code>64</code>
<code>out_pid</code>	<code>PID</code>

- `sys.proc.spawn` `0`
- `sys.proc.spawn`
 - `-ENOENT`
 - `-EINVAL` `.nuc`
 - `-ENOMEM`
- `sys.proc.spawn` `PID`
- `sys.proc.spawn`

4.2 `sys.proc_ctl`

```
int sys_proc_ctl(pid_t pid, uint32_t op, void* arg);
```

- `sys.proc_ctl` `op`
 - `0` = `SIGSTOP`
 - `1` = `SIGCONT`
 - `2` = `SIGKILL`
 - `3` = `attach`
- `CAP` `sys.proc_ctl`
- `sys.proc_ctl` `pid` `op`

4.3 sched_exception_report

```
int sched_exception_report(pid_t pid, exc_info_t* exc);
```

- `pid` `exc_info_t` `panic` `IPC` `WM`
- `exc_info_t` `panic`
- 1. `exc_info_t`
- 2. `panic`
- 3. `IPC` `novus-wm`
- 4. `WM`
- `exc_info_t` `panic`

```
typedef struct {
    uint64_t pid;
    uint64_t rip;      // panic
    uint64_t cr2;      // IPC
    uint32_t err_code;  // WM
    char proc_name[64];
    uint8_t cause;      // 0=panic 1=IPC 2=WM 3=WM
} exc_info_t;
```

5. `cap` /CAP

5.1 cap_check

```
int cap_check(proc_cap_t* p, uint32_t bit);
```

- `cap` `cap`
 - `p` CAP
 - `bit` 0~31
- `cap` `cap`
 - 1
 - 0
- `cap` `cap`

```
return (p->cap_mask_low >> bit) & 1;
```

- `cap` `cap` `bit`

- **Bug** bit off-by-one

5.2 cap_drop

```
int cap_drop(proc_cap_t* p, uint32_t bit);
```

- 0
- 0
- 0 -EINVAL bit

5.3 nkm_load

```
int nkm_load(const char* path);
```

- mod.load CAP
- **debug**
 1. 128 Header
 2. magic == "NKM!" → -ENOENT
 3. abi_version == 1 → -EINVAL
 4. min_kernel_ver ≤ → -EMODULE
 - 5.
 6.
 - 0x50 96 0
 - SHA256
 - module_hash → -ECORRUPT
 7.
 - kernel_trusted_keys[0..key_count-1]
 - module_hash Ed25519 verify
 - → -EMODULE
 - NKM_FLAG_ALLOW_UNSIGNED
 - 8.
 - 9.
 - 10.
 11. nkm_init() → 0
-
- **Bug**
 - hash+sig →
 - → oops

6. VFS

6.1 `vfs_open`

```
int vfs_open(const char* path, uint32_t flags, uint32_t mode);
```

- `flags`
 - bit0 = `O_RDONLY`
 - bit1 = `O_WRONLY`
 - bit2 = `O_RDWR`
 - bit3 = `O_CREAT`
 - bit4 = `O_TRUNC`
- `path`
 - `/system` `fs.full` CAP
 - `fs.full` → `-ENOENT`
- `mode` `0777`
- `path` `flags` `mode` `fd`

6.2 `vfs_read` / `vfs_write`

```
int vfs_read(int fd, void* buf, uint64_t len);  
int vfs_write(int fd, const void* buf, uint64_t len);
```

- `fd`
 - `fd` `vfs_open`
 - `buf`
 - `len` `/`
- `len` `/`
- `fd` `len`
- **Bug** `buf` `NULL` `fd`

7. NovusFS

7.1 `novusfs_alloc_block`

```
int novusfs_alloc_block(uint64_t* out_block);
```

- 检查块是否可用
- 如果块不可用，则分配新块
- 将数据写入块，并更新 *out_block 指针
- 返回块编号
 - -ENOSPC 如果无法分配新块
 - -ECORRUPT 如果块已损坏
- 将块编号写入日志文件
- 将块编号写入 inode 表

7.2 WAL 操作

wal_begin_tx

```
int wal_begin_tx(uint32_t type, uint64_t block_no, wal_tx_t* out);
```

- type 0=PREP 1=inode 2=数据
- 如果块不可用，则分配新块
- 如果块已满，则返回 -EFULL

wal_commit

```
int wal_commit(wal_tx_t* tx, const void* data, uint32_t len);
```

- 将数据写入块 + COMMIT
- 返回块编号

wal_rollback

```
void wal_rollback(wal_tx_t* tx);
```

- 将 PREP 块标记为 COMMIT

WAL 实现

```
// 检查 write_ptr 是否等于 tail_ptr
if (wal_write_ptr == wal_tail_ptr) {
    // 检查 tail 是否等于 wal_write_ptr
    wal_advance_tail();
}
```

8. Merkle

8.1 merkle_verify_root

```
int merkle_verify_root(dev_t dev, const uint8_t expect[32]);
```

- 检查 A 是否正确
- 检查 1. merkle_root_primary
2. primary 是否正确 backup
3. 检查 expect 是否正确 A 是否正确
4. 检查 → 检查 -ECORRUPT
- 检查
 - -ECORRUPT 是否正确
 - -EIO 是否正确
- 检查 检查 expect 是否正确 stored 是否正确 8 是否正确

8.2

```
// idx: 检查=1
uint64_t node_to_block(uint64_t idx) {
    return merkle_start_block + (idx / 64);
}
uint64_t node_offset_in_block(uint64_t idx) {
    return (idx % 64) * 64;
}
```

- 检查
 - leaf_count == 0 → 检查
 - leaf_count == 1 → 检查 =2 检查 =1 检查 =2

9.

9.1 sys_gfx_create_surface

```
int sys_gfx_create_surface(uint32_t w, uint32_t h, uint32_t fmt, fb_handle_t* out);
```

- `out` 指向新创建的表面句柄
 - `w/h` 宽/高
 - `fmt` 0=BGRA8888, 1=BGR24
 - `out` 指向新创建的表面句柄
- **LowMem** 模式 4 字节 FB
- **Full** 模式 16 字节
- 返回 0 表示成功, 非 0 表示失败
- 失败原因: `-ENOMEM` 内存不足

9.2 sys_gfx_submit_dirty

```
int sys_gfx_submit_dirty(fb_handle_t h, const rect_t* rects, uint32_t n);
```

- `rect_t`

```
typedef struct {
    uint32_t x, y, w, h;
} rect_t;
```

- `n` 是 `rects` 数组中 dirty 区域的数量
- `rects` 指向 dirty 区域的数组
- 返回 0 表示成功, 非 0 表示失败

10. IPC

10.1 ipc_send

```
int ipc_send(novus_ipc_hdr_t* hdr, const void* payload);
```

- `hdr` 指向 `novus_ipc_hdr_t` 结构体
 - `magic` 魔法值, 用于验证消息格式
 - `sender_pid` 发送者的 PID
 - `msg_type` 消息类型
 - `payload_len` 消息体长度
- `payload` 指向消息体的指针
- 返回 0 表示成功, 非 0 表示失败

- 返回 0
 - -EFULL 缓冲区满
 - -EINVAL magic 不匹配
 - -EPERM 权限不足
- 返回 -1 表示失败，errno 为错误码，msg_type 为消息类型，target_pid 为目标进程的 PID，payload_len 为有效载荷长度

11. 用户空间 ABI 和 debug 接口

11.1 寄存器

寄存器	名称
RAX	rax
RDI 1	rdi
RDI 2	rsi
RDI 3	rdx
RDI 4	rcx
RDI 5	r8
RDI 6	r9
RAX	rax
堆栈指针	rbp, rbx, r12, r13, r14, r15

11.2 系统调用

1. 系统调用号 存放在 rax 寄存器
2. 系统调用参数 存放在 rdi, rsi, rdx 寄存器
3. 系统调用返回 存放在 rax 寄存器，rsp % 16 == 0
4. 系统调用失败 存放在 rax 寄存器，errno 为错误码，bit 0 为标志位

12. 用户空间

12.1 log_print

```
void log_print(uint32_t level, const char* fmt, ...);
```

- **level**
 - 0 = DEBUG
 - 1 = INFO
 - 2 = WARN
 - 3 = ERROR
- +
- **LowMem** DEBUG
- INFO

12.2 panic_handler

```
void panic_handler(cpu_state_t* state);
```

-
- - 1.
 2. VGA
 3. HLT
- `sched_exception_report` panic

13.

13.1

- init INFO
- ERROR +
- DEBUG LowMem
- VSync

13.2

- `assert(ptr != NULL);`
- `assert((uint64_t)addr % 16 == 0);`
-
- `int ret = foo(); assert(ret == NOVUS_OK);`

13.3 Bug

- 1. →
- 2. → → →inode→
- 3. → corruption
- 4. → x86_64
- 5. **CAP** →
- 6. **WAL** **PREP** **COMMIT** →
- 7. **Merkle** **hash** →
- 8. **NKM** →



 Debug

M0-M10

Revision #1
Created 2026-10-31 11:25:14 UTC by Admin
Updated 2026-10-31 11:25:25 UTC by Admin