

novus□□□□

- [□□□□ M12□ devmgmt□](#)
- [NKM □□□□□□□□ V1.0](#)
- [Novus OS Debug □□□□ V1.0](#)
- [Novus OS □□□□□□□□□□](#)



M12 devmgmt



Novus OS + Windows 11 + Win11 +
/ " " / / / /
Win11

```
“ M12 kernel/devtab.{h,c} kernel/sysinfo.{h,c}  
kernel/usb.{h,c} wm/novus-wm/nwm.c apps/src/wm.rs  
apps/src/devmgr/{mod,tree,props,fields}.rs apps/src/bin/devmgmt.rs
```

1.

apps_launch() devmgmt

1. / / / /
- 2.
3. " " "
4. nsh devmgmt

kernel/session.c desktop_app() apps.c apps_launch()
" " " " / /
"

2.

```
kernel/devtab.c — PCI + blkdev/net/display/CPU/ACPI/USB/IOAPIC/PS2  
| devtab_count / devtab_at / devtab_field_count / devtab_field_at  
kernel/sysinfo.c — REQ_DEVLIST(0x031B) / REQ_DEVFIELD(0x031D) 128 B  
| IPC  
apps/src/sinfo.rs — devices() / dev_fields()  
apps/src/devmgr/* — + +
```

```
| wm::Win::create_ex(..., NWM_F_CREATE_MODAL) / next_event_any()
wm/novus-wm/nwm.c — 120/160 ██████████10 ██████████
```

--	--	--	--	--

- `devtab_init()` `ide_init()` `net_init()`
DMA `net_init()` `devtab_refresh()` `devtab` `DEV_MAX 48`
`PCI_MAX_DEVICES 32` 2.3 KB + 4 KB BAR
- ID 128 B (dev, tab, index)
2 / 56
- + `NWM_F_CREATE_MODAL` WM

3.

--	--	--	--

sysinfo_dev_t 40 B 3 / = 4 + 120

項目	型別	説明
parent	u16	デバイスIDの上位16ビット。0xFFFFは不明。
class_id	u8	DEV_CAT_*を参照。 /CPU /GPU /IO / /MEM /NET /USB /VIRT / /UNKNOWN
status	u8	DEV_STATUS_OK / DEV_STATUS_WARN デバイスの状態
kind	u8	DEV_KIND_*を参照 CPU/PKI /GPU /IO / /USB /VIRT /USB /VIRT / ...
idx	u8	デバイスのインデックス。BAR / I/Oポート
flags	u8	DEV_FLAG_REMOVABLE / DEV_FLAG_NODRV フラグ
name	char[32]	UTF-8で記述された名前

```
sysinfo_devfield_t 60 B 2 / = 4 + 120
```

項目	データ型	初期値
key	u8	DEV_F_* / / /MAC/ / ID/ / / /I/O /IRQ/...
group	u8	BAR [8 bits]

value	char[56]	UTF-8

```
sysinfo_list_t total / tab index n
index += n
```

4.

820x540 " 10 px +

- 22 px (F) / (A) / (V) / (H)
- 36 px / / /
- 20 px 14 px/ + v/> / 12x12 USB ! + 3 px
- 24 px " Enter " scroll ensure_visible

↑/↓ Enter F5 Esc /

5.

520x600 " "

- 44 px / / /
- 24x24 + / " "
- 220x30 / / /
- (P) / +
- / + 3 + " (V)..."
- / x2 I/O IRQ + " " + "(C)..." + "
- " / " "

/ / /

6. 窗口管理

- `nwm_proto.h` 定义了 `NWM_MAX_SHAPES 120` 和 `NWM_MAX_TEXTS 160` 窗口数量，约 35/30 个窗口，约 60/70 个文本。
`NWM_MAX_WINDOWS 10` 窗口大小。
- `NWM_F_CREATE_MODAL` 模态窗口，`raise` 提升窗口。
- `apps/src/wm.rs` 实现 `create_ex(..., flags)` 和 `next_event_any() -> (win, Ev)` 窗口事件，`id` 窗口 ID。

7. 用户界面

窗口管理 WM 和 UI 是分离的，UI 负责窗口管理，WM 负责窗口管理。
UI 初始化 `ui::init` / `ui::poll` 窗口事件，`ui::l(zh, en)` 窗口语言。
lang 窗口语言。

8. 应用管理

- `make` 编译 `out/NOVUS.EFI` 和 `out/apps/devmgt.bin`。
- 配置 `[DEVTAB] n nodes, m pci devices` 窗口管理，`[APP] display list over shape/text budget` 窗口管理。
- 编译 `Makefile APPS_NAMES` 窗口管理，`kernel/apps.c` 窗口管理，`kernel/session.c` 窗口管理，`graphics/nd_demo.c` 窗口管理，`NG_ICON_DEVICES` 窗口管理，`apps/src/shell/cmds.rs` 窗口管理，`KNOWN_APPS` 窗口管理，`apps/Cargo.toml` 窗口管理，`[[bin]]` 窗口管理。

NKM

V1.0

NKM

“

V1.0 Novus OS V1.6-FinalSpec NKM
 x86_64 UEFI
 /

1.

NKM Novus Kernel Module Novus OS .nkm
 Win32 GPU DRM
 NUC

	NUC	NKM
	Ring3	Ring0
		Ed25519
	entry_point	nkm_init() + nkm_exit()
		nkm_unload

2.

0x0000	NKM Header (nkm_hdr_t)	208
0x00D0	Module Dependencies	dep_count × nkm_dep_t (16B/)
...	Segment Table	seg_count × nkm_segment_t (32B/)
...	Symbol Table	sym_count × nkm_sym_t (24B/)
...	Relocation Table	rel_count × nkm_rel_t (16B/)
...	String Table	
...	.text/.data/.rodata	4KiB

3. NKM Header

208 0x00D0 0x0000 packed

“ 128 3.2 — reserved[32] 0xB0 32 0xD0 6.3 0x50..0xB0 208 3.3

3.1

```
#define NKM_MAGIC 0x4E4B4D21 // "NKM!"

typedef struct __attribute__((packed)) {
    uint8_t  magic[4];           // 0x00 "NKM!" = 4E 4B 4D 21
    uint16_t abi_version;        // 0x04 bit[0..7]=minor, bit[8..15]=major
    uint16_t header_size;        // 0x06 = 208 (0xD0)
    uint64_t min_kernel_ver;     // 0x08 
    uint32_t module_type;        // 0x10 /FS/...
```

```

uint16_t seg_count;           // 0x14  0000
uint16_t dep_count;          // 0x16  00000000
uint32_t symtab_count;       // 0x18  00000000
uint32_t rel_count;          // 0x1C  00000000
uint32_t strtab_offset;      // 0x20  00000000
uint32_t rel_offset;         // 0x24  00000000
uint32_t init_offset;        // 0x28  nkm_init() 0 .text 0000
uint32_t exit_offset;        // 0x2C  nkm_exit() 0 .text 0000
uint8_t  module_name[32];    // 0x30  0000 '\0' 0000
uint8_t  module_hash[32];    // 0x50  0000 SHA256 0000000000
uint8_t  module_sig[64];     // 0x70  Ed25519 00
uint8_t  reserved[0x20];     // 0xB0  00 32 00
} nkm_hdr_t;                 // 0000 208 00 (0xD0)

```

3.2 00000000

00	00	00	0000
0x00	4	magic	00 NKM! 00000000
0x04	2	abi_version	0000 =major0000 =minor0000 0x0100
0x06	2	header_size	00 208 0xD0 00000000
0x08	8	min_kernel_ver	00000000 0000 >= min_kernel_ver
0x10	4	module_type	0 \$3.3
0x14	2	seg_count	0000 1~16
0x16	2	dep_count	00000000 0=0000
0x18	4	symtab_count	0000
0x1C	4	rel_count	0000
0x20	4	strtab_offset	00000000
0x24	4	rel_offset	00000000
0x28	4	init_offset	nkm_init() 0 .text 00000000
0x2C	4	exit_offset	nkm_exit() 0 .text 00000000
0x30	32	module_name	0000 \0 0000 31 00

Offset	Size	Field	Value
0x50	32	module_hash	SHA256(\$6)
0x70	64	module_sig	Ed25519(\$6)
0xB0	32	reserved	0

3.3 module_type

Index	Module Type	Description
0	NKM_TYPE_DRIVER	VGA/DP/PCI
1	NKM_TYPE_FS	
2	NKM_TYPE_COMPAT	Win32
3	NKM_TYPE_CRYPT0	
4	NKM_TYPE_OTHER	

4.

Header dep_count 16

```
typedef struct __attribute__((packed)) {
    uint32_t dep_name_off; // 0x00 String Table
    uint64_t min_dep_ver; // 0x04
    uint32_t reserved; // 0x0C 0
} nkm_dep_t; // 16
```

- 1.
2. →
3. min_dep_ver →
4. → NOVUS_EDEP

5. / /

	NKM
vaddr	0000 00000000 000000000000
0000	00 Ring00000 VM_USER TEXT 0000 DATA 00
0000	0000 00000000 00 nksym 00000000 SDK
0000	0 R_X86_64_64 64 0000000000000000 PC32

00000000 nksym 00000000 vmlinux 0000

```
mm_alloc, mm_free, vm_map, vm_unmap
sched_spawn, sched_exit
vfs_register_fs, vfs_read, vfs_write
hal_display_probe, hal_display_flush
cap_check, log_print, err_report
pci_enumerate, blk_read, blk_write
```

6. Ed25519



6.1 0000

- 0000 SHA256
- 0000 Ed25519 RFC 8032
- 0000 32 00
- 0000 64 00

6.2 000000

```
// 00000000000000000000000000000000
static const ed25519_pubkey_t kernel_trusted_keys[4] = {
    /* 001 */, /* 002 */, /* 003 */, /* 004 */
};
static const uint8_t trusted_key_count = 1; // 0001000000000000
```

- 4 个 16 字节
- 16 字节 = 1 个 16 字节
- 16 字节
- 16 字节

6.3 16 字节

“ 16 字节 SHA256 16 字节 = 16 字节 16 字节 module_hash[32] 16 字节 module_sig[64] 16 字节 ”

16 字节

1. 16 字节 temp_buf
2. 16 字节 temp_buf 0x50 16 字节 96 16 字节 32+64 16 字节 0
— 16 字节 module_hash 16 字节 module_sig 16 字节
3. computed_hash = SHA256(temp_buf, 16 字节)
4. 16 字节 computed_hash 16 字节 module_hash
16 字节 → 16 字节

6.4 16 字节

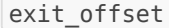
1. 16 字节 computed_hash 16 字节 \$6.3
2. 16 字节 computed_hash == nkm_hdr.module_hash
3. 16 字节 module_sig[64] 16 字节 Ed25519 16 字节
4. 16 字节 nkm_hdr.module_hash[32] 16 字节
5. 16 字节 kernel_trusted_keys[]
if ed25519_verify(module_hash, module_sig, pubkey[i]):
→ 16 字节
6. 16 字节 → 16 字节

6.5 16 字节 hash+sig 16 字节

16 字节 "16 字节" 16 字节 "16 字节" 16 字节 "16 字节" 16 字节

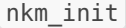
7.

 NKM 

 init_offset /  exit_offset 

```
//   
//  0   
int nkm_init(void);  
  
//   
//  0   
int nkm_exit(void);
```



-  ABI  x86_64 SysV
-  nkm_init  0 → 
-  nkm_exit  IRQ

8.

1. nkm_load(path) 
2.  cap_check(mod.load) → NOVUS_EPERM
3. VFS  208  Header
4.  magic == "NKM!"
5.  abi_version 
6.  min_kernel_ver <= 
7. 
8. 
 - a.  0x50..0xB0  0
 - b.  SHA256
 - c.  module_hash  → NOVUS_ECORRUPT
9.  Ed25519  → NOVUS_EMODULE
 9 
10. 
11. 

12. `nkysym` `nkysym`

13. `nk_init()`

14. `init` `0` → `nkysym`

15. `init` `0` → `nkysym`
- ## 9. `nkysym`
1. `nk_unload(name)` `nkysym`

2. `cap_check(mod.unload)`

3. `nkysym` `dep_count` `nkysym`
`nkysym` → `nkysym` `NOVUS_EBUSY`

4. `nk_exit()`

5. `nkysym`

6. `nkysym`

7. `nkysym`

10. `nkysym`

- `nkysym` `nkysym` `nkysym`
 - `nkysym` `mod.load` `CAP` `nkysym`
 - `nkysym` `NKM_FLAG_ALLOW_UNSIGNED` `nkysym`
 - `nkysym` `WARNING` `nkysym` `PID` `nkysym`
 - `nkysym` `nkysym`
 - `nkysym`

11. `nkysym`

<code>nkysym</code>	<code>nkysym</code>
<code>NOVUS_OK (0)</code>	<code>nkysym</code>
<code>NOVUS_EPERM (-4)</code>	<code>nkysym</code> <code>mod.load</code> / <code>mod.unload</code> <code>CAP</code>
<code>NOVUS_EINVAL (-1)</code>	<code>nkysym</code>
<code>NOVUS_ECORRUPT (-7)</code>	SHA256 <code>nkysym</code>

項目	値
NOVUS_EMODULE (-8)	Ed25519 000000
NOVUS_EBUSY (-5)	00000000
NOVUS_ENOTFOUND (-3)	000000
NOVUS_EDEP (-12)	0000
NOVUS_EIO (-6)	000000
NOVUS_ENOMEM (-2)	000000

12. 0000000000000000

12.1 Header 0000

```
magic:      4E 4B 4D 21  ("NKM!")
abi_version: 00 01      (major=1, minor=0)
header_size: D0 00      (208)
min_kernel_ver: 00 00 00 00 00 00 00 01  (0000 >= 1.0)
module_type: 00 00 00 00  (DRIVER)
seg_count:   02 00      (2)
dep_count:   00 00      (0)
symtab_count: 01 00 00 00
rel_count:   02 00 00 00
strtab_offset: 00 02 00 00  (0x200)
rel_offset:  C0 01 00 00  (0x1C0)
init_offset: 10 00 00 00  (.text+0x10)
exit_offset: 30 00 00 00  (.text+0x30)
module_name: "nvme.ko" 00 00...
module_hash: <3200SHA256>
module_sig:  <6400Ed2551900>
```

12.2 00000000 C

```
#include <novus/nkm.h>
#include <novus/mm.h>
#include <novus/log.h>
```

```

static void* buf = NULL;

int nkm_init(void)
{
    log_print(LOG_INFO, "nvme driver init\n");
    buf = mm_alloc(1);
    if (!buf) return NOVUS_ENOMEM;
    // 00 NVMe 000
    return NOVUS_OK;
}

int nkm_exit(void)
{
    log_print(LOG_INFO, "nvme driver exit\n");
    if (buf) mm_free(buf, 1);
    return NOVUS_OK;
}

```

NUC 0 NKM 00000000 cppcc 000000 .nuc 0000 NKM 0000000000 0000
 tools/nkmggen 0000 NKM 0000000000 Ed25519/SHA
 0000000000000000 C/C++ NKM 0000000000 NKM 00000000
 cppcc0

13. 000000

0000 NKM 000000000000

- Header 208 0000 0x00D000 reserved[32]0
- 0000 0x50..0xB00 hash+sig 000000 SHA2560
- Ed25519 0000 = module_hash 0 32 0000
- 000000 4 0000000
- 0000 nkm_init / nkm_exit 0
- 0000000000000000

000000 ABI 00 RFC000000

0000 000 **M3**0 NKM 000000 **M10**0000000000

Novus OS Debug 0.

--	--	--	--

1. `int C;`
2. `int;`
3. `int / int;`
4. `int [10];` CAP `int [10];`
5. `int [100];`
6. `int [10];`
7. `int [10];` `int [10];`
8. `int [10];` `int [10];`
9. `int [10];` `int [10];` `/int [10];`
10. `int [10];` `int [10];`
11. `int Bug [10];`

[illegible]

			
---	---	---	---

0	NOVUS_OK	成功	成功
-1	NOVUS_EINVAL	無効なパラメータ	NULLポインタ 0 指定
-2	NOVUS_ENOMEM	メモリ不足	メモリ不足
-3	NOVUS_ENOTFOUND	ファイルが見つからない	ファイルが見つからない エラー
-4	NOVUS_EPERM	権限不足	cap_check() が false
-5	NOVUS_EBUSY	デバイスが忙し	デバイスが忙し エラー
-6	NOVUS_EIO	IOエラー	PCIバスのエラー
-7	NOVUS_ECORRUPT	データが破損	SHA256/CRC32 エラー
-8	NOVUS_EMODULE	モジュールが見つからない	ABIバージョン
-9	NOVUS_EFULL	IPCが満杯	IPCが満杯 WAL
-10	NOVUS_EAGAIN	一時失敗	一時失敗
-11	NOVUS_EOVERFLOW	スタックオーバーフロー	スタックオーバーフロー エラー
-12	NOVUS_EDEP	依存関係エラー	依存関係エラー

2. SMP / AB 環境

2.1 smp_shmem_write

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

- SMP 環境
- A (BSP)
- smp_shmem_write
 - A (BSP)
 - smp_shmem_lock() を呼び出す
 - smp_shmem_write() を呼び出す
- 成功

パラメータ	型	範囲	説明
offset	uint32_t	0 ~ 4095	8 バイト オフセット

参数	返回	说明	备注
val	uint64_t	返回	返回

- 返回 void
- 返回 CPU 的 sfence
- 返回
 - assert(smp_shmem_read(offset) == val);
 - log_print(DEBUG, "shmem write off=%u val=%lx\n", offset, val);
- Bug offset

2.2 smp_shmem_read

```
uint64_t smp_shmem_read(uint32_t offset);
```

- 返回 B (AP)
- 返回 AP
- 返回 offset
- 返回
- 返回 bsp_state 200ms

2.3 smp_shmem_lock / smp_shmem_unlock

```
int smp_shmem_lock(void);
void smp_shmem_unlock(void);
```

- 返回
- 返回 1,000,000
- 返回 NOVUS_EBUSY (-5)
- 返回
 -
 -
- 返回

2.4 acore_enter_sleep

```
void acore_enter_sleep(void);
```

- 返回 A
- 返回
 -

- IPI 00000000 AP
- bsp_state 00 2
- 00 00 cli; hlt; 0000
- 0000 00 BSP 00000000
- 0000 000000 "Acore entering sleep"

2.5 smp_wake_ap

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

- 0000 00
 - bsp_state 00 2
 - 000 sfence
 - AP 0000000000
- 00 00 ap_id = AP 0000
- 00 00 IPI 00 0x80
- 000 00 0 000 -ENOMEM 00
- 0000 000000 ap_ready 0000 200ms → 00

2.6 smp_ap_signal_ready

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

- 000 00 AP 00 GDT/IDT/000000
- 0000 000000 ap_ready = 1 00 BSP 000000

3. 00000000

3.1 mm_alloc

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

- 00 000000
- 0000 00
 - 0000000000 M1 000
 - 0000000000 GFP_ATOMIC 000
- 00 00

- 0 0
- - -EINVAL va/pa/len
 - -ENOMEM
- CR3 TLB
- **W^X** flags WRITE EXEC → -EINVAL

3.4 vm_unmap

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

-
-

3.5 oom_scan

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

- < 10%
-
-

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

- init novus-wm novus-settings novus-taskmgr
- 0 out -ENOMEM
- - score
 - victim
- **Bug** →

3.6 oom_reclaim

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

- level
 - 0 =
 - 1 = FB NovusFS

- `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`

args	return
path	<code>.nuc</code> <code>"/system/bin/novus-settings"</code>
caps	<code>sys.proc.spawn</code> CAP 64
out_pid	<code>sys.proc.spawn</code> PID

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

4.2 `sys.proc.spawn`

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

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

4.3 sched_exception_report

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

- 0x00000000 /0x0000 /00
- 00 00
 1. 0x00000000 exc_info_t
 2. 0 panic0x0000
 3. 00 IPC 0x0000 novus-wm
 4. WM 0x00000000
- **exc_info_t** 00 00

```
typedef struct {  
    uint64_t pid;  
    uint64_t rip;        // 0x00000000  
    uint64_t cr2;        // 0x00000000  
    uint32_t err_code;    // 0x00  
    char proc_name[64];  
    uint8_t cause;        // 0=0x00 1=0x0000 2=00 3=00  
} exc_info_t;
```

5. 00 /CAP 00

5.1 cap_check

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

- 00 00
 - p0x0000 CAP 0x0000
 - bit0x00000000 0~310
- 000 00
 - 10x00000000
 - 00x00000000
- 00 00

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

- 0x0000 0x0000000000000000 bit 00

- **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` `fs.full`
- `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` `buf` `len`
 - `fd` `vfs_open`
 - `buf`
 - `len` /
- `fd` `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，尝试分配新块
- 如果块不可用，返回 0 并设置 *out_block 为 NULL
- 如果块不可用，返回 -ENOSPC 或 -ECORRUPT
- 如果块可用，将数据写入块
- 如果块可用，返回 1
- 如果块可用，返回 inode 指针

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=commit
- 如果块不可用，返回 -ENOSPC
- 如果块不可用，返回 -EFULL

wal_commit

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

- 如果块不可用，返回 -ENOSPC
- 如果块不可用，返回 -EFULL

wal_rollback

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

- 如果块不可用，返回 -ENOSPC
- 如果块不可用，返回 -EFULL

WAL 操作

```
// wal_write_ptr == tail_ptr
if (wal_write_ptr == wal_tail_ptr) {
    // 检查 tail 是否可用
    wal_advance_tail();
}
```

8. Merkle 树

8.1 merkle_verify_root

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

- 函数 A 的返回值
- 函数 1. merkle_root_primary
2. primary 备份 backup
3. expect A 的哈希值
4. 函数 → 返回 -ECORRUPT
- 函数
 - -ECORRUPT 返回值
 - -EIO 返回值
- 函数 expect 与 stored 的哈希值 8 字节

8.2 树结构

```
// idx: 索引=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;
}
```

- 函数
 - leaf_count == 0 → 返回
 - leaf_count == 1 → 返回 =2 返回 =1 返回 =2

9. 树结构

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` 数组中矩形个数
- `rects` 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` 是 `novus_ipc_hdr_t` 结构体
 - `magic` 魔法值
 - `sender_pid` 发送者 PID
 - `msg_type` 消息类型
 - `payload_len` 有效载荷长度
- `payload` 是有效载荷
- 返回 0 表示成功, 非 0 表示失败

- 返回 0
 - -EFULL 缓冲区满
 - -EINVAL 无效的 magic 值
 - -EPERM 无权限
- 返回 -1 表示失败，并设置 errno 为 ENOMEM。msg_type 为消息类型，target_pid 为目标进程的 PID，payload_len 为消息长度。

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
rsi 2	rsi
rdx 3	rdx
rcx 4	rcx
r8 5	r8
r9 6	r9
rax	rax
rbp, rbx, r12, r13, r14, r15	

11.2 系统调用

1. 系统调用号存放在 rax 寄存器中。
2. 系统调用参数存放在 rdi, rsi, rdx, rcx, r8, r9 寄存器中。
3. 系统调用返回结果存放在 rax 寄存器中。
4. 系统调用失败时，errno 寄存器中会设置错误码。

12. 用户空间 ABI 和 debug 接口

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

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

13.3 Bug

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



`main` Debug `main`
`main`

`main` `main` **M0-M10** `main`

Novus OS



1. Novus OS



Novus OS 




1.1 Novus OS



- **.hotpth** 


- **.npth**   NKM HAL

- **.apth**   NUC

- **.novu** 

 NUP

1.2 Novus OS



1.  **level_urgent** 
2.  **level_recommend** 
3.  **level_preview** 
4.  **level_optional** 
5.  **level_dev** 

2. Novus OS

[illegible]

2.1

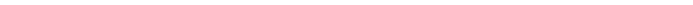
[illegible]

1

Novus OS ██████████ ██████████

[illegible]

--	--	--	--	--	--	--

- 
- 
- 
- 

2.2



.hotpth [5 boxes] > .npth [5 boxes] > .apth [5 boxes] > [5 boxes] > [5 boxes] >
Dev[5 boxes]

2.3



- [30 boxes]
- [30 boxes]
- [15 boxes]
- [25 boxes]

2.4 [15 boxes]

- [30 boxes]
- [15 boxes]
- [25 boxes]

7. Novus OS





uint32_t .novu / .npth / .apth / .hotpth uint32_t

packed 4



uint8_t uint8_t

7.1



```
// 更新头结构体
#define NOVUS_UPDATE_HDR_VER_MAJOR 1
#define NOVUS_UPDATE_HDR_VER_MINOR 0

// 更新类型枚举
#define NOVU_TYPE_FULL          0   // .novu 更新
#define NOVU_TYPE_SYS_PATCH     1   // .npth 系统补丁
#define NOVU_TYPE_APP_PATCH     2   // .apth 应用补丁
#define NOVU_TYPE_HOT_PATCH     3   // .hotpth 热补丁

// 更新级别枚举
#define UPDATE_LVL_OPTIONAL     0
#define UPDATE_LVL_RECOMMEND   1
#define UPDATE_LVL_PREVIEW     2
#define UPDATE_LVL_URGENT      3
#define UPDATE_LVL_DEV         4
```

7.2



```
// 魔数定义
#define NOVU_FULL_MAGIC    0x554E5653  // NVUS → .novu
#define NOVU_NPTH_MAGIC    0x50544853  // PTHS → .npth
#define NOVU_APTH_MAGIC    0x41505448  // APTH → .apth
#define NOVU_HOTPTH_MAGIC  0x484F5450  // HOTP → .hotpth
```

7.3

128Byte



128

magic  file_type 

update_level



```
typedef struct __attribute__((packed)) {
    // 0x00 魔数
    uint32_t magic;                // 魔数
    uint16_t ver_major;
    uint16_t ver_minor;

    // 0x08 文件类型 + 更新级别
    uint8_t file_type;            // 0=novu 1=npth 2=apth 3=hotpth
    uint8_t update_level;         // 0~4 更新级别
    uint8_t remote_support;       // 1=支持 0=不支持
    uint8_t reserved_0;

    // 0x0C 保留字节
}
```

```
uint32_t flag_bits;

// 0x10
uint64_t target_os_ver;      //
uint64_t min_compat_ver;    //

// 0x20
uint32_t entry_count;        // /
uint32_t delta_block_count;  // 0

// 0x28
uint32_t meta_offset;        // /Manifest
uint32_t meta_len;
uint32_t table_offset;       // /
uint32_t table_len;
uint32_t data_offset;        // /
uint32_t data_len;
uint32_t snapshot_offset;    //
uint32_t snapshot_len;

// 0x60
uint8_t file_sha256[32];     //

// 0x80
uint32_t sig_offset;
uint32_t sig_len;

// 0x88 ID16ID
uint8_t patch_uuid_bin[8];

// 0x90 128
uint8_t reserved_fill[24];

} novus_update_header t;
```

7.4 Flag

bit0 = 1 00000000
bit1 = 1 00000000 / 0 00000000
bit2 = 1 00000000
bit3 = 1 0000000000000000
bit4 = 1 000000000000
bit5 = 1 000000000000/00
bit6 = 1 0000000000
bit7 = 1 000000000000
bit8~31 00

7.5



0000 + 000000000000000000000000

0000	Magic	file_type	00 update_level	000000
.novu 0000	NVUS	0	100000	00000000 00000000
.npth 0000	PTHS	1	100000	00000000 00000000
.apth 0000	APTH	2	000000	00000000 000000
.hotpth 0000	HOTP	3	300000	00000000 00000000

7.6 160000 UUID 00000000

000000 160000000000

000000 80000000 16Hex = 64bit

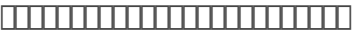


3. Novus OS



3.1



- .hotpth 
- 
- 

3.2



-  3
- 
- 

3.3



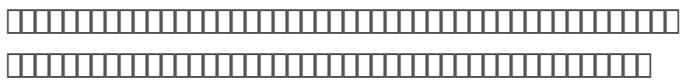
-  ID
- 
- 

BUG

3.4



Novus OS UI



3.4.1

-
-  
-  
-  
-  

CPU

3.4.2



- [Diagram showing two rows of boxes representing data structures]
- [Diagram showing two rows of boxes representing data structures]
- [Diagram showing three rows of boxes representing data structures, with labels "UUID", "SHA256", and "Ed25519" pointing to specific fields]
- [Diagram showing two rows of boxes representing data structures]
- [Diagram showing two rows of boxes representing data structures]

- 

- 

- 


3.4.3

-   IO

-  /  
-  / 

-  

3.4.4

- 

-  ID

- 

3.4.5

- **level_urgent**  
- **level_recommend**  
- **level_preview**  
- **level_optional**  
- **level_dev**  

3.5



3.5

[illegible]-

--	--

 -

--	--

 -

3.5.1

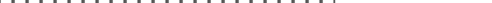


- **.apth**  
- **.npth**  
- **.hotpth**   

- **.novu**  

3.5.2

100% ☐ ☐ ☐ ☐

-  16  UUID
-  30 
- 
- 

60□

3.5.3



- 16 16 UUID
-
-
-
-

3.5.4

1. 16
2. 16
3. 16
4. 16
5. 16
6. 16
7. 16

3.5.5

- .hotpth
-
-

3.5.6

-
-
-

3.5.7

5. Novus OS



Novus OS **100%**

1. +
2. +
3.

6.

-
-
-
-