# Novix Filesystem in RISC-V

In this chapter, we introduce the **Novix Filesystem**, a simple in-memory filesystem built on top of a **TAR archive (initramfs)**.
This allows the kernel to **store and access user programs** like `idle.elf` and `shell.elf` directly from memory during boot.

## Concept Overview

The Novix filesystem is based on a **static TAR archive** that is embedded into the kernel image during build time.
The kernel accesses this archive through linker-defined symbols:

```
_binary_initramfs_tar_start
_binary_initramfs_tar_end
```

Each file entry in the TAR archive contains a **POSIX USTAR header** followed by the actual file contents.
By reading these headers sequentially, the kernel can locate and load files by name.

## Function: tarfs_lookup

```c
void *tarfs_lookup(const char *filename, size_t *filesize, int output_flag);
```

### Purpose

Searches for a given file inside the in-memory TAR filesystem (initramfs) and returns a pointer to its data.

### Parameters

| Parameter | Type | Description |
|---|---|---|
| `filename` | `const char *` | Name of the file to search for. |
| `filesize` | `size_t *` | Optional pointer to store the file size in bytes. |
| `output_flag` | `int` | If nonzero, prints each discovered filename through UART for debugging. |

### Return Value

- Returns a pointer to the start of the file data in memory.
- Returns `NULL` if the file is not found.

### Detailed Description

1. Start scanning from `_binary_initramfs_tar_start`.
2. Read one `struct tar_header` (512 bytes) at a time.
3. Stop if the header name is empty (`hdr->name[0] == '\0'`).
4. Compare the header name with the requested filename:
   ```c
   if (strcmp(hdr->name, filename) == 0)
   ```
5. If matched, convert the octal size field into an integer and return a pointer to the file content:

```
size_t size = (parsed from hdr->size);
if (filesize) *filesize = size;
return ptr + TAR_BLOCK_SIZE;
```

6. If not matched, skip to the next file by computing the number of blocks and moving the pointer accordingly.

#### Usage Example

```
size_t fs;
void *idle_elf = tarfs_lookup("idle.elf", &fs, 1);
if (!idle_elf) PANIC("idle.elf not found!");
```

## User Programs in Initramfs

During the build process (see `run.sh`), several **userland ELF programs** are packed into a TAR archive and included in the kernel image.

| File | Purpose |
|---|---|
| `idle.elf` | Minimal user process that keeps the CPU in a wait-for-interrupt loop. |
| `shell.elf` | Interactive shell (introduced in later chapters). |

#### Example: user/idle.c

```
void main() {
    while (1) {
        __asm__ __volatile__("wfi");
    }
}
```

The **WFI (Wait-For-Interrupt)** instruction puts the CPU in a low-power idle state until the next interrupt occurs.
This is ideal for an `idle` process.

## User Startup Code (init_crt0.S)

```
.section .text.start
.global start
start:
    la sp, __user_stack_top
    call main
    call exit
```

#### Explanation

- Loads the stack pointer from the symbol `__user_stack_top`.
- Calls the user program's `main()` function.
- Calls `exit()` when `main()` returns (which never happens in most cases).

## Function: exit

```
__attribute__((noreturn)) void exit(void) {
    for (;;);
}
```

### Purpose

A simple placeholder for the `exit()` function in user space.
Later, this will become a system call that notifies the kernel when a process terminates.

## Linker Script: user.ld

The **user.ld** file defines the memory layout of every user-space ELF binary.

### Important Sections

| Section | Description |
|---|---|
| `.text` | Executable code and entry point (`start`) |
| `.data`, `.bss` | Global variables and buffers |
| `.stack` | 64 KB user stack |
| `.heap` | 32 MB heap space for dynamic memory (`sbrk()`) |

The linker script also exports symbols used by `init_crt0.S` such as `__user_stack_top` and `__user_heap_start`.

## Function: kernel_main (Filesystem Integration)

```c
void kernel_main(void);
```

### Purpose

Initializes all kernel subsystems and demonstrates filesystem integration.

### Detailed Steps

1. **Initialize the kernel and subsystems**

   ```c
   bss_init();
   trap_init();
   stack_init();
   ram_init();
   heap_init();
   paging_init();
   ```

2. **Search for the idle process**

   ```c
   size_t fs;
   void *idle_elf_file = tarfs_lookup("idle.elf", &fs, 1);
   if (!idle_elf_file) {
       PANIC("[kernel_main] idle.elf not found!");
   }
   ```

3. The kernel now has access to user programs stored in memory and can later load and execute them.

## Summary

At this stage, **Novix OS** has a fully functional **in-memory filesystem (initramfs)** capable of:

- Storing user programs in a TAR archive.
- Accessing files directly from RAM without storage hardware.
- Locating binaries like `idle.elf` for boot initialization.

This marks the beginning of **userland support** in Novix OS.
In the next chapters, the kernel will use this filesystem to **load ELF executables** and create **user processes**.