Building Shell Commands `mem` and `date` in Novix OS

We now have a working shell and essential commands like `ls` and `ps`.
Let's take another step toward a **fully featured userland environment** by adding:

- `mem` — display total and free memory
- `date` — show the current system time and date

Both commands use **syscalls** to request information directly from the kernel, demonstrating how user applications can *safely query system state*.

Learning Goals

By the end of this chapter, you will understand:

- How to implement information-reporting syscalls (`SYS_SYSINFO` and `SYS_GET_TIME`)
- How to safely return kernel data to user space (`copy_to_user`)
- How to build user-level utilities to access system memory and time
- How to extend the Novix shell with more useful built-in tools

Overview: From Kernel to Userland

To make these features work, we'll add:

1. **Kernel syscalls** for memory and time
2. **User-space wrapper programs** (`mem.c` and `date.c`)
3. **Shell integration** to expose them as commands

Kernel Implementation

Let's start with the new syscalls in `kernel/syscall.c`.

```
case SYS_SYSINFO:
    f->regs.a0 = sys_sysinfo((uint64_t)f->regs.a0);
    break;

case SYS_GET_TIME:
    f->regs.a0 = sys_time((uint64_t)f->regs.a0);
    break;
```

Each case maps the syscall number to a handler function that performs the actual logic.

System Information (`sys_sysinfo`)

```
uint64_t sys_sysinfo(uint64_t arg0, ...) {
    struct sysinfo info = {
        .total_pages = page_allocator_total_pages(),
        .free_pages = page_allocator_free_pages(),
    };

    // copy_to_user ensures safe transfer to user space
    return copy_to_user(current_proc->page_table, (void *)arg0, &info, sizeof(info));
}
```

Key details:

- `sysinfo` contains memory statistics such as total and free pages.
- `copy_to_user()` safely copies data from kernel space to the user's address space, avoiding privilege or memory access violations.

Time Query (`sys_time`)

```
int64_t sys_time(uint64_t arg0, ...) {
    struct DateTime dt;

    uint64_t ticks = get_time();        // Retrieve current hardware time
    compute_datetime(ticks, &dt);       // Convert ticks to structured date/time

    return copy_to_user(current_proc->page_table, (void *)arg0, &dt, sizeof(dt));
}
```

Explanation:

- `get_time()` fetches the raw tick count from the system timer.
- `compute_datetime()` converts ticks into human-readable time (year, month, day, etc.).
- The final structure is then safely passed back to user space.

Userland Program: `mem`

Let's create a standalone program to display memory usage.

`user/mem.c`

```
#include "include/stdio.h"
#include "include/syscall.h"
#include "include/sysinfo.h"

int main() {
    struct sysinfo info;
    long ret = syscall(SYS_SYSINFO, (uintptr_t)&info, 0, 0);
    if (ret != 0) {
        printf("sys_sysinfo failed\n");
        return 1;
    }

    printf("=== Memory Status ===\n");
    printf("Total pages: %lu\n", info.total_pages);
    printf("Free  pages: %lu\n", info.free_pages);
    printf("Used  pages: %lu\n", info.total_pages - info.free_pages);
    return 0;
}
```

How it works:

- Calls the `SYS_SYSINFO` syscall directly.
- Receives a `struct sysinfo` from the kernel.
- Prints total, free, and used memory (measured in pages).

Output example:

```
=== Memory Status ===
Total pages: 1024
Free  pages: 800
Used  pages: 224
```

Userland Program: `date`

Now, let's add a command that displays the current system date and time.

user/date.c

```c
#include "include/stdio.h"
#include "include/syscall.h"
#include "include/time.h"

int main(void) {
    struct DateTime now;
    if (syscall(SYS_GET_TIME, (uint64_t)&now, 0, 0) < 0) {
        printf("date: syscall failed\n");
        return -1;
    }

    char ampm[] = "AM";
    int hour = now.hour;
    if (hour >= 12) {
        ampm[0] = 'P';
        if (hour > 12) hour -= 12;
    } else if (hour == 0) {
        hour = 12;
    }

    printf("%s %s %02d %02d:%02d:%02d %s CEST %d\n",
        WEEKDAYS[now.weekday],
        MONTHS[now.month - 1],
        now.day,
        hour, now.minute, now.second,
        ampm,
        now.year);

    return 0;
}
```

Highlights:

- Calls `SYS_GET_TIME` to fetch the current system date/time.
- Converts 24-hour time to 12-hour format with AM/PM.
- Displays output in human-readable format, including timezone and weekday.

Example output:

```
Tue Oct 08 03:45:21 PM CEST 2025
```

Shell Integration

To make these new tools accessible from the command line,
we update `user/shell.c` with help text and automatic discovery:

```
void print_help(void) {
    puts("Available commands:\n");
    puts("  help         Show this overview\n");
    puts("  hello        Show the hello message\n");
    puts("  ps           Show list of active processes\n");
    puts("  ls           Show files in the current directory\n");
    puts("  mem          Show memory usage / available memory\n");
    puts("  date         Show the current date and time\n");
    puts("  clear        Clear the screen\n");
}
```

Because `mem.elf` and `date.elf` are standalone binaries,
no further modification is needed — the shell will automatically recognize them.

Example Session

```
Novix RISC-V 64 OS, (c) NovixManiac, Shell version : 0.0.1
$ ls
Files in tarfs:
  shell.elf
  hello.elf
  ps.elf
  ls.elf
  mem.elf
  date.elf

$ mem
=== Memory Status ===
Total pages: 1024
Free  pages: 780
Used  pages: 244

$ date
Wed Oct 08 10:32:18 AM CEST 2025
```

- Both commands run independently as user processes
- They use syscalls to query real kernel data
- The shell dynamically launches them without modification

Design Reflection

Concept	Purpose
`SYS_SYSINFO`	Reports memory usage (pages)
`SYS_GET_TIME`	Returns system date/time
`copy_to_user()`	Safely transfers kernel data to user space
`mem.elf` / `date.elf`	User utilities that use syscalls
Shell	Launches both as independent user programs

This pattern opens the door to future commands like `uptime`, `df`, or even `top`.

Summary

In this chapter, we've achieved:

- Added `SYS_SYSINFO` and `SYS_GET_TIME` kernel syscalls
- Built `mem.elf` and `date.elf` user programs
- Extended the shell's help menu and command set
- Demonstrated safe kernel-to-user data transfer

Novix OS now offers **system-level introspection** —
users can monitor memory and check the system clock,
just like in any Unix-style operating system.

Next up:

We'll introduce **command arguments**, allowing programs like `echo` to accept input directly from the command line.

```
*A smile on my face - because Novix OS shell has now date and time!*
```