Random IT Utensils

IT, operating systems, maths, and more.

Menu

Windows Research Kernel Part 3 — Syscall

AUGUST 4, 2018

> This is the third part of the WRK series. For your convenience you can find other parts in the table of contents in Part 1 — Compiling and debugging

Today we are going to write a very simple hello world in the kernel space. Let's go.

Kernel

First, we need to add new syscall to the table of services. Go to the file `base\ntos\ke\i386\systable.asm`. In line 392 add the following:

```
TABLE_ENTRY  HelloKernel, 1, 1
```

This specifies that at least one argument must be passed (first 1) and at most one as well (second 1).
Next, replace `TABLE_END 295` with `TABLE_END 295`.

Now we need to add a stub. Go to `base\ntos\ke\i386\sysstubs.asm` and add the following in line 2485:

```
SYSSTUBS_ENTRY1  296, HelloKernel, 1
SYSSTUBS_ENTRY2  296, HelloKernel, 1
SYSSTUBS_ENTRY3  296, HelloKernel, 1
SYSSTUBS_ENTRY4  296, HelloKernel, 1
SYSSTUBS_ENTRY5  296, HelloKernel, 1
SYSSTUBS_ENTRY6  296, HelloKernel, 1
SYSSTUBS_ENTRY7  296, HelloKernel, 1
SYSSTUBS_ENTRY8  296, HelloKernel, 1
```

Now the linker expects a method with one int parameter. We need to implement it, so let's go to `base\ntos\ps\psquery.c` and add the following in line 4220:

```
NTSTATUS
NtHelloKernel(
    int count
    )
{
    int i;
    PAGED_CODE();
    for (i=0; i < count; ++i) {
        DbgPrint("Hello world: [%d]",i);
    }

    return STATUS_SUCCESS;
}
```

Recompile the kernel and reboot the os.

User mode

Now we need to execute the method from the user space. In theory we should implement a wrapper for the syscall in a DLL. Next, we would just write an application calling the method via the DLL. But, we are going to do it a little differently:

```
void
CallNtHelloKernel(
    int count
    )
{
    void** stackFrame = (void*)(&count);

    __asm {
        mov eax, 0x0128
        mov edx, stackFrame
        int 0x2E
    }
}

int main(int argc, char* argv[]) {

    printf("calling HelloKernel\n");

    // use new system service call
    CallNtHelloKernel(5);
}
```

We first define a method with exactly the same signature as the method in the kernel. Next, we define a stack frame for the parameter. Finally, we switch to the kernel space with the interrupt. `eax` register holds the ordinal of the syscall (remember what number we put in `sysstubs.asm`?) and `edx` holds a stack frame.

Finally, we just call this method from the C code.

Compile the code to a binary. This binary is completely unrelated to the WRK, it is just an application like every other. Copy it to your VM and run in your modified kernel. As a bonus, try to run it with the unmodified WRK kernel and see what happens.

POSTED IN CODING, DEBUGGING

WRK

‹ PREVIOUS
Windows Research Kernel Part 2 — Monitoring the function invocation

NEXT ›
Windows Research Kernel Part 4 — New module

Search … Search

Recent Posts

State Machine Executor Part 6 — Forking

Non-atomic assignments in Python

State Machine Executor Part 5 — Streaming

State Machine Executor Part 4 — Timeouts, exceptions, suspending

State Machine Executor Part 3 — Actions and history

Categories

Administration

Coding

Computer Science

Databases

Debugging

Math

Philosophy

Archive

November 2025 (2)

October 2025 (5)

August 2025 (2)

July 2025 (1)

November 2024 (1)

September 2024 (2)

June 2024 (1)

May 2024 (4)

April 2024 (1)

March 2024 (3)

February 2024 (2)

December 2023 (2)

February 2023 (4)

January 2023 (4)

December 2022 (5)

November 2022 (4)

October 2022 (5)

September 2022 (4)

August 2022 (4)

July 2022 (5)

June 2022 (4)

May 2022 (4)

April 2022 (5)

March 2022 (4)

February 2022 (4)

January 2022 (5)

December 2021 (4)

November 2021 (4)

October 2021 (5)

September 2021 (4)

August 2021 (4)

July 2021 (5)

June 2021 (4)

May 2021 (5)

April 2021 (4)

March 2021 (4)

February 2021 (4)

January 2021 (5)

December 2020 (4)

November 2020 (4)

October 2020 (5)

September 2020 (4)

August 2020 (5)

July 2020 (4)

June 2020 (4)

May 2020 (5)

April 2020 (4)

March 2020 (4)

February 2020 (5)

January 2020 (4)

December 2019 (4)

November 2019 (5)

October 2019 (4)

September 2019 (4)

August 2019 (5)

July 2019 (4)

June 2019 (5)

May 2019 (4)

April 2019 (4)

March 2019 (5)

February 2019 (5)

January 2019 (4)

December 2018 (5)

November 2018 (4)

October 2018 (4)

September 2018 (5)

August 2018 (4)

July 2018 (4)

June 2018 (5)

May 2018 (4)

April 2018 (4)

March 2018 (5)

February 2018 (4)

January 2018 (4)

January 2016 (5)

December 2015 (4)

November 2015 (4)

October 2015 (5)

September 2015 (4)

August 2015 (3)

Posts