8. Systémová volání

Systémová volání a inline assembler

Cílem tohoto cvičení je detailně se seznámit s tím, jak aplikace volá služby operačního systému a jak lze systémová volání volat přímo, bez použití knihoven jako `libc`. Tato znalost se vám bude hodit v dalších cvičeních.

Domácí příprava

Seznamte se se základními instrukcemi architektury `x86` a způsobem, jakým vkládat instrukce assembleru přímo do zdrojového kódu v jazyce C/C++. Projděte si prezentaci o inline assembleru a připomeňte si první a druhou přednášku.

Dále se seznamte s nástroji strace a ltrace. Doporučujeme si oba nástroje vyzkoušet na následujícím programu:

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    printf("Hello world\n");
    return 0;
}
```

Pokud vás zajímají podrobnosti o kombinovaní assembleru a C, podívejte se do dokumentace překladače GCC. Mimo jiné tam najdete popisy obecných omezení a omezení závislých na architektuře CPU (hledejte sekci “x86 family”).

Zadání úlohy

Vytvořte program, který čte ze standardního vstupu celá nezáporná dekadická čísla oddělená mezerami nebo konci řádků (případně jinými nečíselnými znaky) a vypíše je na standardní výstup v hexadecimální podobě oddělená koncem řádku. Program nesmí používat žádnou standardní knihovnu jako např. `libc`. Předpokládejte, že maximální hodnota čísla bude `2^32-1`.

Jinými slovy vytvořte program fungující podobně jako program níže, ale tak, aby šel přeložit s přepínači kompilátoru

```
-ffreestanding -fno-stack-protector -nostdlib -nostdinc -static -m32 -Wall
```

```c
#include <stdio.h>

int main()
{
    unsigned num;
    while (scanf("%u", &num) == 1)
        printf("0x%x\n", num);
    return 0;
}
```

Do BRUTE nahrávejte soubor `hexconv.c` se svou implementací.

Nápověda

- Pro vyvolání služeb jádra potřebujete znát *ABI* (application binary interface) jádra OS. To se dočtete v man syscall v řádcích architektury i386.
- V Ubuntu čísla jednotlivých systémových volání najdete v souboru `/usr/include/x86_64-linux-gnu/asm/unistd_32.h`.

> Pozor! Jak ABI, tak čísla systémových volání se liší mezi 32- a 64-bitovým jádrem. 64-bitové jádro je možné volat pomocí obou ABI, ale v této úloze používejte 32-bitové ABI, protože program je kompilován s přepínačem `-m32`.

- Můžete vyjít z kódu níže, který již obsahuje načítání vstupu místo funkce `scanf`.
- Pro tisk i načítání můžete použít pole pevné délky např. 20 (maximální výstup má 8 hexadecimálních znaků).

```c
#include <unistd.h> /* TODO: replace this by writing your own system call
                    /* wrappers for read(), write(), exit() */
#include <stdio.h>  /* TODO: replace this by your own implementation of */
                    /* sprintf() (for conversion of a number to hex string)
#include <string.h> /* TODO: replace this with your implementation of strle

int isnum(char ch)
{
    return ch >= '0' && ch <= '9';
}

int isspc(char ch)
{
    return ch == ' ' || ch == '\n';
}

static void print(unsigned num)
{
    char buf[20];
    /* TODO: Get rid of sprintf() and strlen() */
    sprintf(buf, "0x%x\n", num);
    int ret = write(STDOUT_FILENO, buf, strlen(buf));
    if (ret == -1)
        _exit(1); // TODO: your new exit
}

/* TODO: main() is called by libc. Without libc, the entry point is called
int main()
{
    char buf[20];
    unsigned num = 0;
    int i = 0;
    int num_digits = 0;
    unsigned chars_to_process = 0;

    for (/* no init */; /* no end condition */; i++, chars_to_process--) {
        if (chars_to_process == 0) {
            int ret = read(STDIN_FILENO, buf, sizeof(buf));
```

```
            if (ret < 0)
                return 1; // TODO: replace by exit
            i = 0;
            chars_to_process = ret;
        }
        if (
            num_digits > 0
            && (chars_to_process == 0 /* EOF */ || !isnum(buf[i]))
        ) {
            print(num);
            num_digits = 0;
            num = 0;
        }
        if (
            chars_to_process == 0 /* EOF */
            || (!isspc(buf[i]) && !isnum(buf[i]))
        )
            return 0; // TODO: replace by exit

        if (isnum(buf[i])) {
            num = num * 10 + buf[i] - '0';
            num_digits++;
        }
    }
}
```

Materiály

- inline assembler

Domácí příprava na další cvičení

viz stránka https://osy.pages.fel.cvut.cz/docs/cviceni/lab9/