

Segmentation Fault

DELTA - Střední škola informatiky a ekonomie, s.r.o.

Ing. Luboš Zápotočný

11.12.2025

CC BY-NC-SA 4.0

Segmentation fault

Co je Segmentation Fault?

Segmentation Fault (SIGSEGV) = program se pokusil o **nepovolený přístup k paměti**

Kdy nastává?

- Přístup k paměti, která programu nepatří
- Zápis do read-only paměti
- Použití NULL pointeru
- Přístup po konci pole (buffer overflow)
- Použití uvolněné paměti (use-after-free)

Co je Segmentation Fault?

Výsledek:

Segmentation fault (core dumped)

Top 5 příčin Segmentation Fault

1. Dereferencing NULL pointer

```
int *ptr = NULL;  
*ptr = 5;           // SEGFAULT  
printf("%d", *ptr); // SEGFAULT
```

Proč to padá?

- NULL = adresa 0x0
- OS chrání adresu 0x0 před přístupem
- Pokus o čtení → SIGSEGV

Top 5 příčin Segmentation Fault

Jak opravit?

```
int *ptr = malloc(sizeof(int));  
if (ptr != NULL) { // Always check!  
    *ptr = 42;  
}
```

Top 5 příčin Segmentation Fault

2. Buffer overflow

```
char *str = malloc(3);
for (int i = 0; i < 1000000; i++) {
    str[i] = 'A'; // Write out of bounds
}
```

Co se děje?

- Malloc alokoval 3 byty
- Program píše do 1,000,000 bytů
- Přepisuje cizí paměť
- Nakonec narazí na chráněnou oblast → SEGFAULT

Top 5 příčin Segmentation Fault

Zajímavost: Nemusí padat hned! Může trvat dlouho než narazí na chráněnou paměť (což je horší - těžko se debuguje)

Top 5 příčin Segmentation Fault

3. Use after free

```
int *p = malloc(sizeof(int));  
free(p);  
*p = 10; // Memory is not valid anymore
```

Top 5 příčin Segmentation Fault

4. Zápis do read-only paměti

```
char *str = "Hello"; // String literal (read-only)  
str[0] = 'h'; // SEGFAULT
```

Vysvětlení:

- String literály jsou v **text segmentu**
- Text segment je **read-only** (ochrana kódu)
- Pokus o zápis → SEGFAULT

Top 5 příčin Segmentation Fault

Řešení:

```
char str[] = "hello"; // Stack (writable)
str[0] = 'H';          // OK

// or
char *str = malloc(6);
strcpy(str, "hello");
str[0] = 'H'; // OK
```

Top 5 příčin Segmentation Fault

5. Stack overflow (rekurze)

```
void recursive() {  
    int arr[10000];  
    recursive(); // Infinite recursion  
}
```

Jak debugovat Segmentation Fault?

1. GDB (GNU Debugger)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void myfn() {
    char *str = "Hello"; // String literal (read-only)
    str[0] = 'h';
}
int main() {
    myfn();
    return 0;
}
```

Jak debugovat Segmentation Fault?

```
gcc -g program.c -o program # Compile with debug info  
gdb ./program  
(gdb) run  
(gdb) backtrace # Shows where it crashed  
(gdb) list      # Shows the code around the crash  
(gdb) print str # Shows the value of str  
(gdb) quit       # Exit GDB
```

Jak debugovat Segmentation Fault?

2. Valgrind

```
valgrind ./program  
# Shows memory leaks and invalid accesses
```

Jak debugovat Segmentation Fault?

3. AddressSanitizer

```
gcc -fsanitize=address program.c -o program  
./program # Detects errors at runtime
```

Prevence Segmentation Fault

Best practices:

- Vždy inicializujte pointery (NULL nebo platnou adresou)
- Kontrolujte návratové hodnoty `malloc()`
- Používejte `free()` jen jednou
- Nastavte pointer na NULL po `free()`
- Kontrolujte hranice polí
- Používejte bezpečné funkce (`strncpy`, `snprintf`)
- Používejte statickou analýzu a sanitizery
- Testujte s Valgrind

Prevence Segmentation Fault

Pamatujte: Prevence je lepší než debugging!