

Kódovací příklady

Jan Faigl

Katedra počítačů

Fakulta elektrotechnická

České vysoké učení technické v Praze

Přednáška 12

BAB36PRGA – Programování v C



Přehled témat

- Část 1 – Kódovací příklady

Ukazatele a Pole

Řazení řetězců

Jednoduchá kalkulačka

Přetypování ukazatele na pole

- Část 2 – Kódovací příklad vícevláknové programování (příklad z 11. přednáška prakticky)



Část I

Část 1 – Kódovací příklady



Obsah

Ukazatele a Pole

Řazení řetězců

Jednoduchá kalkulačka

Přetypování ukazatele na pole



Kódovací příklad – Pole a ukazatel na funkci 1/4

- Implementujte program, který vytvoří pole náhodných kladných, celých čísel voláním funkce `rand()` z `stdlib.h`. *Funkce fill.*
- Hodnota celých čísel je omezena na `MAX_NUM`, např. nastavena na 20, `#define MAX_NUM 20`.
- Počet náhodných čísel `LEN` může být nastaven při kompilaci – `clang -DLEN=10 program.c`.
- Pole je vypsáno na `stdout`. *Funkce print.*
- Pole je uspořádáno funkcí `qsort()` ze `stdlib.h`. *Seznamte s funkcí, viz man qsort.*
- Uspořádané pole je vypsáno na `stdout`.
- Program je dále rozšířen o zpracování argumentu programu definujícího počet náhodných čísel s využitím funkce `atoi()`.

```
#ifndef LEN
#define LEN 5
#endif

#define MAX_NUM 20

void fill_random(size_t l, int a[l]);
void print(const char *s, size_t l, int a[l]);

int main(void)
{
    int a[LEN]; // allocate the array
    fill_random(LEN, a); // fill the array
    print("Array random: ", LEN, a);
    // TODO call qsort
    print("Array sorted: ", LEN, a);
    return 0;
}
```



Kódovací příklad – Pole a ukazatel na funkci 2/4

```
void fill_random(size_t l, int a[l])
{
    for (size_t i = 0; i < l; ++i) {
        a[i] = rand() % MAX_NUM;
    }
}

void print(const char *s, size_t l, int a[l])
{
    if (s) {
        printf("%s", s);
    }
    for (size_t i = 0; i < l; ++i) {
        printf("%s%d", i > 0 ? " " : "", a[i]);
    }
    putchar('\n');
}
```

- Vizte man qsort.

```
void qsort(
    void *base, size_t nmemb, size_t size,
    int (*compar)(const void *, const void *)
);

    ■ base je ukazatel na první prvek;
    ■ nmemb je počet prvků;
    ■ size je velikost (každého) prvku;
    ■ compar je ukazatel na funkci porovnání.

int compare(const void *ai, const void *bi)
{
    const int *a = (const int*)ai;
    const int *b = (const int*)bi;
    //ascending
    return *a == *b ? 0 : (*a < *b ? -1 : 1);
}
```

Změňte pořadí na sestupné.



Kódovací příklad – Pole a ukazatel na funkci 3/4

- Název funkce použijte jako ukazatel na funkci.

```
int compare(const void *, const void *);
```

```
int main(void)
{
    int a[LEN]; // do not initialize
    fill_random(LEN, a);
    print("Array random: ", LEN, a);
    qsort(a, LEN, sizeof(int), compare);
    print("Array sorted: ", LEN, a);
    return 0;
}
```

- Kompilujte a spusťte program pouze pokud byla kompilace úspěšná použitím **shell logický and** operátor **&&**.

```
$ clang sort.c -o sort && ./sort
Array random: 13 17 18 15 12
Array sorted: 12 13 15 17 18
```

- Použijte argument kompilátoru **-DLEN=10** k definici velikosti pole 10.

```
$ clang -DLEN=10 sort.c -o sort && ./sort
Array random: 13 17 18 15 12 3 7 8 18 10
Array sorted: 3 7 8 10 12 13 15 17 18 18
```



Kódovací příklad – Pole a ukazatel na funkci 4/4

- Rozšiřte `main()` o předání argumentů.
- Definujte návratovou hodnotu při chybě.

```

1  enum { ERROR = 100 };

3  int main(int argc, char *argv[])
4  {
5      const size_t len = argc > 1 ?
6          atoi(argv[1]) : LEN;
7      if (len > 0) {
8          int a[len];
9          fill_random(len, a);
10         print("Array random: ", len, a);
11         qsort(a, len, sizeof(int), compare);
12         print("Array sorted: ", len, a);
13     }
14     return len > 0 ? EXIT_SUCCESS : ERROR;
15 }

```

- Použijeme **Variable Length Array (VLA)**, které umožňují definovat velikost pole za běhu.

```

$ clang sort-vla.c -o sort && ./sort
Array random: 13 17 18 15 12 3
Array sorted: 3 12 13 15 17 18

```

```

$ clang sort-vla.c -DLEN=7 -o sort && ./sort
Array random: 13 17 18 15 12 3 7
Array sorted: 3 7 12 13 15 17 18

```

```

$ clang sort-vla.c -o sort && ./sort 11
Array random: 13 17 18 15 12 3 7 8 18 10 19
Array sorted: 3 7 8 10 12 13 15 17 18 18 19

```

- Uvědomte si, že velikost pole `a` je omezena velikostí **zásobníku**, viz `ulimit -s`.



Obsah

Ukazatele a Pole

Řazení řetězců

Jednoduchá kalkulačka

Přetypování ukazatele na pole



Kódovací příklad – Řazení řetězců 1/5

- Implementujte program, který lexikograficky uspořádá argumenty programu použitím `strcmp` (z `string.h`) a `qsort` (z `stdlib.h`).
- Vypište argumenty. *Funkce print.*
- Zkopírujte předané argumenty `argv` do nově alokované paměti na haldě, abychom zamezili změněm `argv`.
 - Při chybě program končí hodnotu -1. *Vlastní funkce alokace.*
 - Kopírování řetězců: funkce `strncpy`.
- Řazení řetězců realizujeme s využitím funkce `strcmp` a `qsort`. *Porovnání řetězců.*
- Alokovanou paměť uvolněte. *Funkce release.*

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void print(int n, char *strings[n]);

char* copy(const char *str);
char** copy_strings(int n, char *strings[n]);

void* my_malloc(size_t size);
void release(int n, char **strings);

int string_compare(
    const void *p1, const void *p2);

enum { EXIT_OK = 0, EXIT_MEM = -1 };

int main(int argc, char *argv[]);
```



Kódovací příklad – Řazení řetězců 2/5

- Print funkce přímo iteruje přes pole řetězců.

```
void print(int n, char *strings[n])
{
    for (int i = 0; i < n; ++i) {
        printf("%3d. \"%s\"\n", i, strings[i]);
    }
}
```

- Alokace pole ukazatelů na char (znak) – pole textových řetězců.

```
char** copy_strings(int n, char *strings[n])
{
    char** ret = my_malloc(n * sizeof(char*));
    for (int i = 0; i < n; ++i) {
        ret[i] = copy(strings[i]);
    }
    return ret;
}
```

Alokace je úspěšná nebo program končí chybou.

- Copy volá `my_malloc` a používá `strncpy`.

```
char* copy(const char *str)
{
    char *ret = NULL;
    if (str) {
        size_t len = strlen(str);
        ret = my_malloc(len + 1); // +1 for '\0'
        strncpy(ret, str, len + 1); // +1 for '\0'
    }
    return ret;
}
```

- Délka řetězce (voláním `strlen`) je bez *null terminating* `'\0'`.
- Kopie řetězce musí obsahovat znak konce řetězce (*null terminating character*).



Kódovací příklad – Řazení řetězců 3/5

- Dynamická alokace volá `malloc` a při chybě program končí.

```
void* my_malloc(size_t size)
{
    void *ret = malloc(size);
    if (!ret) {
        fprintf(stderr,
            "ERROR: Mem allocation error!\n");
        exit(EXIT_MEM);
    }
    return ret;
}
```

- Dynamicky alokované pole ukazatelů na dynamicky alokované řetězce vyžaduje uvolnění paměti jednotlivých prvků (textových řetězců) a až následně paměti vlastního pole ukazatelů.

```
void release(int n, char **strings)
{
    if (strings && *strings)
        return;

    for (int i = 0; i < n; ++i) {
        if (strings[i]) {
            free(strings[i]); //free string
        }
    }
    free(strings); // free array of pointers
}
```



Kódovací příklad – Řazení řetězců 4/5

- Předpis funkce `qsort`, viz `man qsort`.

```
void qsort(void *base, size_t nmem, size_t size,
           int (*compar)(const void *, const void *)
           );
```

Předáváme ukazatele na prvky pole jako ukazatele na konstantní proměnné (hodnoty).

- Voláme `qsort` na pole ukazatelů na textové řetězce, což jsou ukazatele na znak (`char`).

```
char **strings = copy_strings(n, argv);
qsort(strings, n, sizeof(char*), string_compare);
```

- Ukazatel na prázdný typ (`void`) explicitně přetypujeme na ukazatel na ukazatel na znak (`char`) pro přístup k textovému řetězci.

```
int string_compare(const void *p1, const void *p2)
{
    char * const *s1 = p1; // qsort passes a pointer to the array item (string)
    char * const *s2 = p2;
    return strcmp(*s1, *s2);
}
```



Kódovací příklad – Řazení řetězců 5/5

- Volání `qsort` na pole ukazatelů.

```
int main(int argc, char *argv[])
{
    int ret = EXIT_OK;
    const int n = argc;
    printf("Arguments:\n");
    print(argc, argv);

    char **strings = copy_strings(n, argv);
    qsort(
        strings, n,
        sizeof(char*), string_compare
    );

    printf("\n Sorted arguments:\n");
    print(n, strings);
    release(n, strings);
    return ret;
}
```

- `clang str_sort.c && ./a.out 4 2 a z c`

Arguments:

```
0. "./a.out"
1. "4"
2. "2"
3. "a"
4. "z"
5. "c"
```

Sorted arguments:

```
0. "./a.out"
1. "2"
2. "4"
3. "a"
4. "c"
5. "z"
```

- Další úkoly.

- Implementujte `strings` jako pole ukazatelů bez explicitního počtu prvků, ale s koncem indikovaným hodnotou `NULL`.
- Implementujte alokaci řetězců jako jeden souvislý blok paměti ve kterém jsou všechny řetězce za sebou, ale oddělené `'\0'`.



Obsah

Ukazatele a Pole

Řazení řetězců

Jednoduchá kalkulačka

Přetypování ukazatele na pole



Kódovací příklad – Jednoduchá kalkulačka 1/6

- Implementujte kalkulačku celých čísel s operátory '+', '-', '*'. *Sum, sub, a mult funkce.*

Jednoduchá kalkulačka bez uvažování priorit operátorů.

- Program hlásí chybu a vrací 100 pokud vstup není celé číslo a hodnotu 101 pokud vstup obsahuje nepodporovaný operátor.
- Použijte ukazatel na funkci/e.
- Vstup zpracovávejte krok po kroku, bez nutnosti načítání celého vstupu, a vypisujte dílčí výsledky.
- Program reaguje na všechny možné chyby.
 - Vstup musí obsahovat alespoň jedno celé číslo.
 - Pokud je zadán operátor, musí být platný a musí být zadán druhý operand.
 - Pokud konec vstupu, a není zadán operátor,, vypište výsledek.

```
enum status { EXIT_OK = 0, ERROR_INPUT = 100,
              ERROR_OPERATOR = 101 };
```

```
enum status printe(enum status error);
```

```
int main(int argc, char *argv[])
{
    enum status ret = EXIT_OK;
    ...
    return printe(ret);
}
```

```
enum status printe(enum status error)
{
    if (error == ERROR_INPUT) {
        fprintf(stderr, "ERROR: Input value\n");
    } else if (error == ERROR_OPERATOR) {
        fprintf(stderr, "ERROR: Operator\n");
    }
    return error;
}
```



Kódovací příklad – Jednoduchá kalkulačka 2/6

- Implementujte kalkulačku celých čísel s operátory '+', '-', '*'. *Sum, sub, a mult funkce.*

Jednoduchá kalkulačka bez uvažování priorit operátorů.

- Program hlásí chybu a vrací 100 pokud vstup není celé číslo a hodnotu 101 pokud vstup obsahuje nepodporovaný operátor.
- Použijte ukazatel na funkci/e.
- Vstup zpracovávejte krok po kroku, bez nutnosti načítání celého vstupu, a vypisujte dílčí výsledky.
- Program reaguje na všechny možné chyby.
 - Vstup musí obsahovat alespoň jedno celé číslo.
 - Pokud je zadán operátor, musí být platný a musí být zadán druhý operand.
 - Pokud konec vstupu, a není zadán operátor,, vypište výsledek.

```
int sum(int a, int b); // return a + b
int sub(int a, int b); // return a - b
int mult(int a, int b); // return a * b
```

```
//define a pointer to a function
typedef int (*ptr)(int, int);
```

```
//typedef ptr is needed for the return value
ptr getop(const char *op)
{
    int (*operation)(int, int) = NULL;
    if (op[0] == '+') {
        operation = sum;
    } else if (op[0] == '-') {
        operation = sub;
    } else if (op[0] == '*') {
        operation = mult;
    }
    return operation;
}
```



Kódovací příklad – Jednoduchá kalkulačka 3/6

- Implementujte kalkulačku celých čísel s operátory '+', '-', '*'. *Sum, sub, a mult funkce.*

Jednoduchá kalkulačka bez uvažování priorit operátorů.

- Program hlásí chybu a vrací 100 pokud vstup není celé číslo a hodnotu 101 pokud vstup obsahuje nepodporovaný operátor.
- Použijte ukazatel na funkci/e.
- Vstup zpracovávejte krok po kroku, bez nutnosti načítání celého vstupu, a vypisujte dílčí výsledky.
- Program reaguje na všechny možné chyby.
 - Vstup musí obsahovat alespoň jedno celé číslo.
 - Pokud je zadán operátor, musí být platný a musí být zadán druhý operand.
 - Pokud konec vstupu, a není zadán operátor,, vypište výsledek.

```
int r = 1; //the first v1
char opstr[2] = {}; //store the operator
ptr op = NULL; // function pointer
int v2; //store the second operand
while (r == 1 && ret == EXIT_OK) {
    r = (op = readop(opstr, &ret)) ? 1 : 0;
    // operator is valid and second operand read
    int v3 = op(v1, v2);
    printf("%3d %s %3d = %3d\n",
           v1, opstr, v2, v3);
    v1 = v3; //shift the results
} else if (!op) { // no operator
    printf("Result: %3d\n", v1);
    r = 0;
} else if (r != 1) { //no operand
    ret = ERROR_INPUT;
}
} //end of while
```



Kódovací příklad – Jednoduchá kalkulačka 4/6

- Implementujte kalkulačku celých čísel s operátory '+', '-', '*'. *Sum, sub, a mult funkce.*

Jednoduchá kalkulačka bez uvažování priorit operátorů.

- Program hlásí chybu a vrací 100 pokud vstup není celé číslo a hodnotu 101 pokud vstup obsahuje nepodporovaný operátor.
- Použijte ukazatel na funkci/e.
- Vstup zpracovávejte krok po kroku, bez nutnosti načítání celého vstupu, a vypisujte dílčí výsledky.
- Program reaguje na všechny možné chyby.
 - Vstup musí obsahovat alespoň jedno celé číslo.
 - Pokud je zadán operátor, musí být platný a musí být zadán druhý operand.
 - Pokud konec vstupu, a není zadán operátor,, vypište výsledek.

```
enum status ret = EXIT_OK;
int v1;

int r = scanf("%d", &v1) == 1;
ret = r == 0 ? ERROR_INPUT : ret;
if (ret == EXIT_OK) {
    ret = process(ret, v1);
}
```

...

```
ptr readop(char *opstr, enum status *error)
{
    ptr op = NULL; // pointer to a function
    int r = scanf("%1s", opstr);
    if (r == 1) {
        *error = (op = getop(opstr)) ? *error :
            ERROR_OPERATOR;
    } // else end-of-file
    return op;
}
```



Kódovací příklad – Jednoduchá kalkulačka 5/6

```
43 enum status process(enum status ret, int v1)
44 {
45     int r = 1; //the first operand is given in v1
46     char opstr[2] = {}; //store the operator
47     ptr op = NULL; // function pointer to operator
48     int v2; //store the second operand
49     while (r == 1 && ret == EXIT_OK) {
50         r = (op = readop(opstr, &ret)) ? 1 : 0; // operand read succesfully
51         if (r == 1 && (r = scanf("%d", &v2)) == 1) { // while ends for r == 0 or r == -1
52             int v3 = op(v1, v2);
53             printf("%3d %s %3d = %3d\n", v1, opstr, v2, v3);
54             v1 = v3; //shift the results
55         } else if (!op) { // no operator in the input
56             printf("Result: %3d\n", v1); //print the final results
57             r = 0;
58         } else if (r != 1) { //no operand on the input
59             ret = ERROR_INPUT;
60         }
61     } //end of while
62     return ret;
63 }
```

Po načtení operandu v2, můžeme načíst další operátor a kontrolovat prioritu.



Kódovací příklad – Jednoduchá kalkulačka 6/6

```
1 enum status { EXIT_OK = 0, ERROR_INPUT =
    100, ERROR_OPERATOR = 101 };
2 ...
3 typedef int (*ptr)(int, int);
4 ptr getop(const char *op);
5 enum status printe(enum status error);
6 enum status process(enum status ret, int v1);

9 int main(int argc, char *argv[])
10 {
11     enum status ret = EXIT_OK;
12     int v1;

14     int r = scanf("%d", &v1) == 1;
15     ret = r == 1 ? ret : ERROR_INPUT;
16     if (ret == EXIT_OK) {
17         ret = process(ret, v1);
18     }
19     return printe(ret);
20 }
```

■ Příklad výstupu programu.

```
$ clang calc.c -o calc
$ echo "1 + 2 * 6 - 2 * 3 + 19" | ./calc
1 + 2 = 3
3 * 6 = 18
18 - 2 = 16
16 * 3 = 48
48 + 19 = 67
Result: 67

$ echo "1 + 2 *" | ./calc; echo $?
1 + 2 = 3
ERROR: Input value
100

$ echo "1 + 2 a" | ./calc; echo $?
1 + 2 = 3
Result: 3
ERROR: Operator
```



Obsah

Ukazatele a Pole

Řazení řetězců

Jednoduchá kalkulačka

Přetypování ukazatele na pole



Kódovací příklad – Přetypování ukazatele na pole 1/4

- Alokujte pole o velikosti `ROWS × COLS` a vyplňte jej náhodnými celými čísly s maximálně dvěma ciframi a vypište hodnoty jako pole.
- Implementujte funkce `fill` a `print`.
- Implementujte funkci `print`, která vytiskne matici o velikosti `rows × cols`.
- Přetypujte pole `int` hodnot na ukazatel `m`, ukazatel na pole o velikosti `cols`.

Přetypování nám může pomoci pochopit, že paměť je paměť a proměnná nám umožňuje interpretovat hodnoty v paměti. Zde je zásadní, že se jedná o souvislý blok paměti.
- Předejte `m` funkci pro vypis 2D pole (matice) s `cols` sloupci.

```
#define MAX_VALUE 100
#define ROWS 3
#define COLS 4

void fill(int n, int *v);
void print_values(int n, int *a);

int main(int argc, char *argv[])
{
    const int n = ROWS * COLS;
    int array[n];
    int *p = array;

    fill(n, p);
    print_values(n, p);
    return 0;
}
```



Kódovací příklad – Přetypování ukazatele na pole 2/4

- Alokujte pole o velikosti `ROWS × COLS` a vyplňte jej náhodnými celými čísly s maximálně dvěma ciframi a vypište hodnoty jako pole.
- Implementujte funkce `fill` a `print`.
- Implementujte funkci `print`, která vytiskne matici o velikosti `rows × cols`.
- Přetypujte pole `int` hodnot na ukazatel `m`, ukazatel na pole o velikosti `cols`.

Přetypování nám může pomoci pochopit, že paměť je paměť a proměnná nám umožňuje interpretovat hodnoty v paměti. Zde je zásadní, že se jedná o souvislý blok paměti.

- Předejte `m` funkci pro vypis 2D pole (matice) s `cols` sloupci.

```
void fill(int n, int *v)
{
    for (int i = 0; i < n; ++i) {
        v[i] = rand() % MAX_VALUE;
    }
}

void print_values(int n, int *a)
{
    for (int i = 0; i < n; ++i) {
        printf("%s%i",
            (i > 0 ? " " : ""),
            a[i]
        );
    }
    putchar('\n');
}
```



Kódovací příklad – Přetypování ukazatele na pole 3/4

- Alokujte pole o velikosti `ROWS × COLS` a vyplňte jej náhodnými celými čísly s maximálně dvěma ciframi a vypište hodnoty jako pole.
- Implementujte funkce `fill` a `print`.
- Implementujte funkci `print`, která vytiskne matici o velikosti `rows × cols`.
- Přetypujte pole `int` hodnot na ukazatel `m`, ukazatel na pole o velikosti `cols`.

Přetypování nám může pomoci pochopit, že paměť je paměť a proměnná nám umožňuje interpretovat hodnoty v paměti. Zde je zásadní, že se jedná o souvislý blok paměti.

- Předejte `m` funkci pro vypis 2D pole (matice) s `cols` sloupci.

```
void print(int rows, int cols, int m[][cols])
{
    for (int r = 0; r < rows; ++r) {
        for (int c = 0; c < cols; ++c) {
            printf("%3i", m[r][c]);
        }
        putchar('\n');
    }
}
```

- Počet sloupců je nezbytný pro výpočet adresy buňky matice `m[r][c]` reprezentované 2D polem (maticí) `m`.
- Ukazatel `m` může odkazovat na paměť s libovolným počtem řádků.



Kódovací příklad – Přetypování ukazatele na pole 4/4

- Alokujte pole o velikosti `ROWS × COLS` a vyplňte jej náhodnými celými čísly s maximálně dvěma ciframi a vypište hodnoty jako pole.
- Implementujte funkce `fill` a `print`.
- Implementujte funkci `print`, která vytiskne matici o velikosti `rows × cols`.
- Přetypujte pole `int` hodnot na ukazatel `m`, ukazatel na pole o velikosti `cols`.

Přetypování nám může pomoci pochopit, že paměť je paměť a proměnná nám umožňuje interpretovat hodnoty v paměti. Zde je zásadní, že se jedná o souvislý blok paměti.

- Předejte `m` funkci pro vypis 2D pole (matice) s `cols` sloupci.

Zkuste vytisknout pole jako matic s `cols` sloupci a jako matici s `rows` slouci, což je matice s rozměry `rows × cols` nebo `cols × rows`.

```
#define MAX_VALUE 100
#define ROWS 3
#define COLS 4
...
void print(int rows, int cols, int m[][cols]);

int main(int argc, char *argv[])
{
    const int n = ROWS * COLS;
    int array[n];
    int *p = array;

    int (*m)[COLS] = (int(*)[COLS])p;
    printf("\nPrint as matrix %d x %d\n",
           ROWS, COLS);
    print(ROWS, COLS, m);
    return 0;
}
```



Část II

Část 2 – Kódovací příklad vícevláknové programování (příklad z 11. přednáška prakticky)



Vlákna POSIX – Příklad 1/10

- Vytvoření aplikace se třemi aktivními vlákny.
 - Obsluha uživatelského vstupu – funkce `input_thread()`.
 - Uživatel zadá periodu výstupu obnovení stisknutím vyhrazených kláves.
 - Zobrazení výstupu – funkce `output_thread()`.
 - Aktualizce výstupu pouze tehdy, když uživatel interaguje s aplikací nebo když alarm signalizuje, že uplynula perioda.
 - Alarm s periodou definovanou uživatelem – funkce `alarm_thread()`.
 - Obnovení výstupu nebo provedení jiné akce.
- Pro zjednodušení program používá `stdin` a `stdout` s hlášením aktivity vlákna do `stderr`.
- Synchronizační mechanismy jsou demonstrují použití mutexu a podmíněné proměnné.
 - `pthread_mutex_t mtx` – výhradní přístup k `data_t data`;
 - `pthread_cond_t cond` – signalizace vláken.

Sdílená data se skládají z aktuální periody alarmu (`alarm_period`), požadavku na ukončení aplikace (`quit`) a počtu vyvolání alarmu (`alarm_counter`).



Vlákna POSIX – Příklad 2/10

- Včetně hlavičkových souborů, definice datových typů, deklarace globálních proměnných.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <stdbool.h>
4 #include <termios.h>
5 #include <unistd.h> // for STDIN_FILENO
6 #include <pthread.h>

8 #define PERIOD_STEP 10
9 #define PERIOD_MAX 2000
10 #define PERIOD_MIN 10

12 typedef struct {
13     int alarm_period;
14     int alarm_counter;
15     bool quit;

17     pthread_mutex_t *mtx; // avoid global variables for mutex and
18     pthread_cond_t *cond; // conditional variable
19 } data_t; // data structure shared among the threads
```



Vlákna POSIX – Příklad 3/10

■ Funkce prototypů a inicializace proměnných a struktur.

```
21 void call_termios(int reset); // switch terminal to raw mode
22 void* input_thread(void*);
23 void* output_thread(void*);
24 void* alarm_thread(void*);

26 // - main function -----
27 int main(int argc, char *argv[])
28 {
29     data_t data = { .alarm_period = 100, .alarm_counter = 0, .quit = false };
30     enum { INPUT, OUTPUT, ALARM, NUM_THREADS }; // named ints for the threads
31     const char *threads_names[] = { "Input", "Output", "Alarm" };
32     void* (*thr_functions[])(void*) = {
33         input_thread, output_thread, alarm_thread // array of thread functions
34     };

36     pthread_t threads[NUM_THREADS]; // array for references to created threads
37     pthread_mutex_t mtx;
38     pthread_cond_t cond;
39     pthread_mutex_init(&mtx, NULL); // initialize mutex with default attributes
40     pthread_cond_init(&cond, NULL); // initialize condition variable with default attributes
41     data.mtx = &mtx; // make the mutex accessible from the shared data structure
42     data.cond = &cond; // make the cond accessible from the shared data structure
```



Vlákna POSIX – Příklad 4/10

- Vytvoření vláken a čekání na ukončení všech vláken.

```
43     call_termios(0); // switch terminal to raw mode
44     for (int i = 0; i < NUM_THREADS; ++i) {
45         int r = pthread_create(&threads[i], NULL, thr_functions[i], &data);
46         printf("Create thread '%s' %s\r\n", threads_names[i], ( r == 0 ? "OK" : "FAIL" ) );
47     }

49     int *ex;
50     for (int i = 0; i < NUM_THREADS; ++i) {
51         printf("Call join to the thread %s\r\n", threads_names[i]);
52         int r = pthread_join(threads[i], (void*)&ex);
53         printf("Joining the thread %s has been %s - exit value %i\r\n", threads_names[i],
54             (r == 0 ? "OK" : "FAIL"), *ex);
55     }

56     call_termios(1); // restore terminal settings
57     return EXIT_SUCCESS;
58 }
```



Vlákna POSIX – Příklad 5/10 (Přepnutí terminálu)

- Přepnutí terminálu do režimu *raw*.

```
59 void call_termios(int reset)
60 {
61     static struct termios tio, tioOld; // use static to preserve the initial
        settings
62     tcgetattr(STDIN_FILENO, &tio);
63     if (reset) {
64         tcsetattr(STDIN_FILENO, TCSANOW, &tioOld);
65     } else {
66         tioOld = tio; //backup
67         cfmakeraw(&tio);
68         tcsetattr(STDIN_FILENO, TCSANOW, &tio);
69     }
70 }
```

Volající je zodpovědný za vhodné volání funkce, např. pro zachování původního nastavení musí být funkce volána s argumentem 0 pouze jednou.



Vlákna POSIX – Příklad 6/10 (Vstupní vlákno 1/2)

```
72 void* input_thread(void* d)
73 {
74     data_t *data = (data_t*)d;
75     static int r = 0;
76     int c;
77     while (( c = getchar()) != 'q') {
78         pthread_mutex_lock(data->mtx);
79         int period = data->alarm_period; // save the current period
80         // handle the pressed key detailed in the next slide
81     }
82     ...
83     if (data->alarm_period != period) { // the period has been changed
84         pthread_cond_signal(data->cond); // signal the output thread to refresh
85     }
86     data->alarm_period = period;
87     pthread_mutex_unlock(data->mtx);
88 }
89 r = 1;
90 pthread_mutex_lock(data->mtx);
91 data->quit = true;
92 pthread_cond_broadcast(data->cond);
93 pthread_mutex_unlock(data->mtx);
94 fprintf(stderr, "Exit input thread %lu\r\n", pthread_self());
95 return &r;
96 }
```



Vlákna POSIX – Příklad 7/10 (Vstupní vlákno 2/2)

- `input_thread()` – zpracuje požadavek uživatele na změnu periody.

```
81 switch(c) {
82     case 'r':
83         period -= PERIOD_STEP;
84         if (period < PERIOD_MIN) {
85             period = PERIOD_MIN;
86         }
87         break;
88     case 'p':
89         period += PERIOD_STEP;
90         if (period > PERIOD_MAX) {
91             period = PERIOD_MAX;
92         }
93         break;
94 }
```



Vlákna POSIX – Příklad 8/10 (výstupní vlákno)

```
96 void* output_thread(void* d)
97 {
98     data_t *data = (data_t*)d;
99     static int r = 0;
100    bool q = false;
101    pthread_mutex_lock(data->mtx);
102    while (!q) {
103        pthread_cond_wait(data->cond, data->mtx); // wait for next event
104        q = data->quit;
105        printf("\rAlarm time: %10i    Alarm counter: %10i", data->alarm_period,
106            data->alarm_counter);
107        fflush(stdout);
108    }
109    pthread_mutex_unlock(data->mtx);
110    fprintf(stderr, "Exit output thread %lu\r\n", (unsigned long)pthread_self());
111    return &r;
112 }
```



Vlákna POSIX – Příklad 9/10 (Alarm vlákno)

```
113 void* alarm_thread(void* d)
114 {
115     data_t *data = (data_t*)d;
116     static int r = 0;
117     pthread_mutex_lock(data->mtx);
118     bool q = data->quit;
119     useconds_t period = data->alarm_period * 1000; // alarm_period is in ms
120     pthread_mutex_unlock(data->mtx);

121
122     while (!q) {
123         usleep(period);
124         pthread_mutex_lock(data->mtx);
125         q = data->quit;
126         data->alarm_counter += 1;
127         period = data->alarm_period * 1000; // update the period if it has been changed
128         pthread_cond_broadcast(data->cond);
129         pthread_mutex_unlock(data->mtx);
130     }
131     fprintf(stderr, "Exit alarm thread %lu\r\n", pthread_self());
132     return &r;
133 }
```



Vlákna POSIX – Příklad 10/10

- Příkladový program `lec11/threads.c` lze zkompilovat a spustit.

```
clang -c threads.c -std=gnu99 -O2 -pedantic -Wall -o threads.o
clang threads.o -lpthread -o threads
```

- Periodu lze změnit klávesami 'r' a 'p'.
- Aplikace je ukončena po stisknutí 'q'.

```
./threads
Create thread 'Input' OK
Create thread 'Output' OK
Create thread 'Alarm' OK
Call join to the thread Input
Alarm time:          110   Alarm counter:          20Exit input thread 750871808
Alarm time:          110   Alarm counter:          20Exit output thread 750873088
Joining the thread Input has been OK - exit value 1
Call join to the thread Output
Joining the thread Output has been OK - exit value 0
Call join to the thread Alarm
Exit alarm thread 750874368
Joining the thread Alarm has been OK - exit value 0
```

`lec11/threads.c`



Shrnutí přednášky



Diskutovaná témata

- Kódovací příklady
- Vícevláknové programování

