

POSIX

Konečně něco více než “Hello World!” pomocí malloc

POSIX – rodina norem OS

- Svázáno s operačními systémy typu UNIX
- Jazyk C je součástí POSIXu (C vyvinuto pro Unix)
- Zastřešil předchozí vývoj v systémovém API
- POSIX.1 (1988)
- POSIX.1-2008 (2008)

POSIX C library

● Snaha o vytvoření jednotného okolního prostředí (API) umožňující přenositelnost programů vyžadujících interakci s OS

- http://en.wikipedia.org/wiki/C_POSIX_library

● Knihovna funkcí pro interakci s OS, pokrývá širokou oblast

- kontrola procesů (spouštění, komunikace, ukončení)
- práce se vstupními a výstupními zařízeními (pipes...)
- práce s vlákny, synchronizační mechanismy (mutex)
- spouštění příkazů shellu
- ...

POSIX - kompatibilita

● Unix/Linux

- dobrá dlouhodobá podpora, bez problémů
- Linux Programmer's Manual (`alisa:~ # man -s 2,3 jmenofunkce`)

● MS Windows

- omezená implementace standardu, jen některé distribuce,
- funkce obvykle začínají `_`
- <https://msdn.microsoft.com/en-us/library/z0kc8e3z.aspx>
- MS preferuje využívání svých funkcí – Win32 API...
- Cygwin – implementace jádra POSIX normy pro C
- MinGW – podobné Cygwin, ale windows-native backend
 - viz absence `%lld`

POSIX – feature test macros

- POSIX rozšiřuje standardní knihovnu
- některé funkce potřeba „povolit“

- feature_test_macros

```
0: #define _POSIX_C_SOURCE 2
1: #include <unistd.h>
```

- Položka feature test macros v manuálu

Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

lstat():

```
/* glibc 2.19 and earlier */ _BSD_SOURCE ||
/* Since glibc 2.20 */ _DEFAULT_SOURCE ||
_XOPEN_SOURCE >= 500 || _XOPEN_SOURCE && _XOPEN_SOURCE_EXTENDED
|| /* Since glibc 2.10: */ _POSIX_C_SOURCE >= 200112L
```

Ohlašování chyb

- `#include <errno.h>`
- Součást standardu jazyka C
- Globální proměnná `int` `errno`
- Pokud funkce standardní knihovny či POSIXu selže, obnáší chybový kód
 - chybové kódy definovány jako konstanty
 - například pro chyby matematických funkcí `EDOM`
 - nulujte před použitím funkce, která chybu nastaví
- Specifikace C99 sekce 7.5
- Příkaz `man errno`

Pohodlnější práce s argumenty – getopt

- `#include <unistd.h>`

- Usnadňuje práci s argumenty

- Iterativní přístup, podpora různých způsobů

- `int` getopt(`int` argc, `char * const` argv[],
`const char *`optstring)

- Proměnná optstring – řetězec prepínačů

- Iterativně volána, stav v návratovém kódu

- <0 konec, jinak prepínač

- Příkazové řádce `./program -r 8 -R` odpovídá
optstring `"r:R"`

Práce se soubory a adresáři

- `#include <dirent.h>`

- Popisovač souborů (file descriptor)

- index do tabulky souborů v jádře OS (handle)

- `DIR *opendir(const char *dirname);`

- `struct dirent *readdir(DIR *dirp);`

- `int closedir(DIR *dirp);`

- Pozor, funkce mohou být stavové

(Ne)stavovost funkcí

- Jak vrátit jména pro 1001 souborů v adresáři?
- Jediným funkčním zavoláním?
 - je nutné naformátovat jména do jediného řetězce
- Co použít jako oddělovač?
 - speciální znak – nepraktické, který zvolit?
 - koncová nula? Používá se, jak byste udělali?
- Zavedení stavové funkce
 - každé zavolání vrátí další soubor
 - např. `readdir()` vrátí po každém zavolání další soubor v adresáři
 - stav musí být uchováván (OS) a uvolněn!

Zjištění obsahu adresáře

- 1. Otevření adresáře** (funkce opendir)
 - vytvoří stavovou strukturu u OS (DIR)
 - připojenou na daný adresář
- 2. Postupné procházení adresáře** (funkce readdir)
 - každé zavolání vrátí další soubor v adresáři
 - formou struktury (struct dirent)
- 3. Práce s nalezeným souborem** (struct dirent)
 - dirent.d_name
 - např. pomocí C funkce fopen()
- 4. Ukončení práce s adresářem** (funkce closedir)
 - uvolní stavovou strukturu u OS

Výpis obsahu adresáře

```
void PosixPrintFiles(const char* path) {  
    DIR *dir = NULL;  
    if ((dir = opendir(path)) { // connect to directory  
        struct dirent *dirEntry = NULL;  
        while ((dirEntry = readdir(dir)) != NULL) { // get next item  
            printf("File %s\n", dirEntry->d_name); // get name  
        }  
        closedir(dir); // finish work with directory  
    }  
}
```

- Jak rozlišit podadresář od souboru?
- Jak zjistit další informace o souboru? (čas, práva)
- Jak projít zanořenou strukturu adresářů?

Rozlišení adresáře od souboru

● Problém v rozdílné podpoře v Unixu / Windows

● Linux:

- struct dirent.d_type
 - 4, 10 adresář, 8 soubor
- makro S_ISDIR, funkce stat nebo lstat <sys/stat.h>

```
#define DT_UNKNOWN 0
#define DT_DIR 4
#define DT_REG 8
#define DT_LNK 10
```

● Windows:

- omezená podpora, dirent.d_type nemusí být dostupný
 - např. není defaultně v MinGW
- zkusit otevřít položku pomocí opendir()
 - pokud se nepodaří, nemusí být adresář (proč?)

```
struct dirent    *dp;
struct stat      statbuf;
struct passwd    *pwd;
struct group     *grp;
struct tm        *tm;
char             datestring[256];

... // Open directory opendir etc.

/* Loop through directory entries. */
while ((dp = readdir(dir)) != NULL) {
    /* Get entry's information. */
    if (stat(dp->d_name, &statbuf) == -1) continue;
    /* Print out type, permissions, and number of links. */
    printf("%10.10s", sperm (statbuf.st_mode));
    printf("%4d", statbuf.st_nlink);
    /* Print out owner's name if it is found using getpwuid(). */
    if ((pwd = getpwuid(statbuf.st_uid)) != NULL)
        printf(" %-8.8s", pwd->pw_name);
    else
        printf(" %-8d", statbuf.st_uid);
    /* Print out group name if it is found using getgrgid(). */
    if ((grp = getgrgid(statbuf.st_gid)) != NULL)
        printf(" %-8.8s", grp->gr_name);
    else
        printf(" %-8d", statbuf.st_gid);
    /* Print size of file. */
    printf(" %9jd", (intmax_t)statbuf.st_size);

    tm = localtime(&statbuf.st_mtime);
    /* Get localized date string. */
    strftime(datestring, sizeof(datestring), nl_langinfo(D_T_FMT), tm);
    printf(" %s %s\n", datestring, dp->d_name);
}
```

Vlákna v POSIXu

● Vlákna umožňují spustit několik úkolů paralelně

- při jednom jádře CPU se střídají
- při více jádrech CPU mohou běžet paralelně

● Pracovní vlákno (Worker thread)

- funkce obsahující kód pro vykonání
- data předaná při spuštění funkci jako argument
- spuštění potřebného počtu vláken

● Použití vláken může vyžadovat synchronizaci

- ochrana před nevhodným souběžným použitím zdrojů
- zápis do paměti, přístup k souboru...

● <https://computing.llnl.gov/tutorials/pthreads/>

● <http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html>

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

funkce pro vlákno

zpracování argumentu

```
void *print_message_function( void *ptr ) {
    char *message;
    message = (char *) ptr;
    printf("%s \n", message);
}
```

spuštění vlákna

funkce pro vlákno

struktura pro kontrolu vlákna

```
int main() {
    pthread_t thread1, thread2;
    char *message1 = "Thread 1";
    char *message2 = "Thread 2";
    int iret1, iret2;
```

```
/* Create independent threads each of which will execute function */
iret1 = pthread_create( &thread1, NULL, print_message_function, (void*) message1);
iret2 = pthread_create( &thread2, NULL, print_message_function, (void*) message2);
```

počkáme na dokončení vláken

```
/* Wait till threads are complete before main continues. Unless we
/* wait we run the risk of executing an exit which will terminate
/* the process and all threads before the threads have completed. */
pthread_join( thread1, NULL);
pthread_join( thread2, NULL);
```

argument pro funkci

```
printf("Thread 1 returns: %d\n",iret1);
printf("Thread 2 returns: %d\n",iret2);
return 0;
```

Další práce se soubory

● Vytvoření dočasného souboru

- `FILE* tmpfile(void);`

● Přejmenování a přesunutí souboru

- `int rename(const char *from, const char *to);`

● Smazání souboru či adresáře

- `int remove(const char *pathname);`

● Asynchronní přístup

- `int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout);`

Další funkce POSIXu

Naleznete v přednášce “Návaznost jazyka C na OS” (Šimon Tóth), web předmětu

BONUS – lokalizace – gettext

- `#include <libintl.h>`

- Výběr katalogu zpráv

 - `char * textdomain(const char *domainname)`

- Získání formátovacího řetězce zprávy

 - `char * gettext(const char *msgid) ;`

- Makro `_(msg)`

 - syntaktický cukr pro `gettext(msg)`

- Přeuspořádání argumentů: `"%2$s %1$s"`

- https://fedoraproject.org/wiki/How_to_do_I18N_through_gettext