Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

CS Basics
16) More C APIs

E. Benoist & C. Grothoff

Fall Term 2018-19

Computer Science Division

More C APIs

- GNU libc
- Other Libraries
- Conclusion

Choosing a Programming Language for a Project

- **Can your team use it?**
  - More training needed to be productive?
  - Recuitment of new developers feasible?
- **Can it do the job?**
  - Supports all target platforms?
  - Performance requirements (RAM, CPU) met?
- **What tooling can you build upon?**
  - What development tools exist?
  - What libraries exist?

C rarely wins on the team side, but remains hard to beat for the others.

GNU libc

libc

- Most functions from course so far are from `libc`
- Standardized by the Portable Operating System Interface for Computer Envirionments (ISO/IEC 9945, also known as POSIX).
- Widely used implementation is the GNU libc of the GNU project
- Used by default with the GNU C compiler (gcc)
- Other implementations exist, sometimes with sublte incompatibilities!

Copying memory

```
#include <string.h>

void *
memcpy (void *restrict to,
        const void *restrict from,
        size_t size);
```

C99 `restrict` keyword says the pointer is the *only* thing that accesses the underlying object (no aliasing).

Copying memory

```
#include <string.h>

void *
memmove (void *to,
         const void *from,
         size_t size);
```

Copying strings

```
#include <string.h>

char *
strcpy (char *restrict to,
        const char *restrict from);

char *
stpcpy (char *restrict to,
        const char *restrict from);

char *
strdup (const char *s);
```

Concatenating strings

```
#include <string.h>

char *
strcat (char *restrict to,
        const char *restrict from);
```

Truncated copies

```c
#include <string.h>

char *
strncpy (char *restrict to,
         const char *restrict from,
         size_t size); // Dangerous!

char *
strndup (const char *s,
         size_t size);

char *
strncat (char *restrict to,
         const char *restrict from,
         size_t size);
```

Searching

```c
#include <string.h>

void *
memchr (const void *block, int c, size_t size);
void *
memrchr(const void *block, int c, size_t size);

char *
strchr (const char *string, int c);
char *
strrchr(const char *string, int c);

char *
strstr (const char *haystack,
        const char *needle);
char *
strcasestr (const char *haystack,
            const char *needle);
```

Binary Search

```
#include <search.h>

typedef int
(*comparison_fn_t)(const void *,
                   const void *);

void *
bsearch (const void *key,
         const void *array,
         size_t count,
         size_t size,
         comparison_fn_t compare);
```

Time

- Simple time: `time_t`, typically gives the number of seconds of elapsed time since the Unix Epoch (1.1.1970).
- High-resolution time: `struct timeval`
- Local time or broken-down time: `struct tm`

Getting and setting the time

```
#include <time.h>

time_t
time (time_t *result);


int
stime (const time_t *newtime);
```

Breaking it down

```
#include <time.h>

struct tm {
    int tm_sec;   int tm_min;   int tm_hour;
    int tm_mday;  int tm_mon;   int tm_year;
    int tm_wday;  int tm_yday;  int tm_isdst;
    long int tm_gmtoff;  const char *tm_zone;
};

struct tm *
localtime (const time_t *time);

struct tm *
gmtime (const time_t *time); /* UTC was ↘
→formerly called GMT */
```

Printing it out

```c
#include <time.h>

char *
asctime (const struct tm *brokentime);
/* => "Tue Dec 26 13:46:22 2017\n" */

size_t
strftime (char *s,
          size_t size,
          const char *template,
          const struct tm *brokentime)
/* Important: %z == RFC 822/ISO 8601:1988 */
```

Sleeping

```c
#include <unistd.h>

unsigned int
sleep (unsigned int seconds);

int
nanosleep (const struct timespec *req,
           struct timespec *remaining);
```

Processing variable argument lists

```
#include <stdarg.h>

void va_start(va_list ap, last);

type va_arg(va_list ap, type);

void va_end(va_list ap);
```

Processing variable argument lists: Example

```
void m (const char *format, ...) {
 va_list ap;
 va_start (ap, format);
 while (*format) switch (*(format++)) {
  case 'i':{int i = va_arg (ap, int); break;}
  case 'f':{float f = va_arg (ap, float);
            break;}
  case 'p':{void *p = va_arg (ap, void *);
            break;}
  default: exit (1);
 }
 va_end(ap);
}

m ("iffp", 4, 4.5, 5.5, &data);
```

Memory mapping

```
#include <sys/mman.h>

PROT_EXEC, PROT_READ, PROT_WRITE, PROT_NONE
MAP_SHARED ^ MAP_PRIVATE
MAP_ANONYMOUS, MAP_LOCKED, ...

void *
mmap (void *addr, size_t length, int prot, int ↘
→flags,
      int fd, off_t offset);

int
munmap (void *addr, size_t length);
```

Temporary files

```c
#include <stdlib.h>

int
mkstemp (char *template);

char *t = strdup ("/tmp/fooXXXXXX");
int fd = mkstemp (t);
```

/etc/passwd

```
#include <sys/types.h>
#include <pwd.h>

struct passwd {
  char    *pw_name;       /* username */
  char    *pw_passwd;     /* user password */
  uid_t    pw_uid;        /* user ID */
  gid_t    pw_gid;        /* group ID */
  char    *pw_gecos;      /* user information */
  char    *pw_dir;        /* home directory */
  char    *pw_shell;      /* shell program */
};

struct passwd *getpwnam (const char *name);
struct passwd *getpwuid (uid_t uid);
```

Dynamic libraries (plugins)

```
#include <dlfcn.h>

RTLD_LAZY, RTLD_NOW
RTLD_GLOBAL, RTLD_LOCAL, ...

void *dlopen (const char *filename, int flags);
void *dlsym (void *handle, const char *symbol);
int dlclose (void *handle);
```

Other Libraries

Other libraries

- libgmp – large numbers
- libcurl – network client
- libz – gzip compression
- libtidy – parse HTML
- libmagic – determine file type
- libjansson – JSON manipulation
- libsqlite3 – SQL database

GNU mp: Powers of two

```
#include <gmp.h>

    mpz_t val;
    mpz_init (val);
    mpz_set_ui (val, 1);
    for (unsigned int i=0;i<10000;i++) {
      mpz_out_str (stdout, 10, val);
      fprintf (stdout, "\n");
      mpz_mul_ui (val, val, 2);
    }
    mpz_clear (val);
    return 0;
```

libcurl: HTTP

```
#include <curl/curl.h>

    CURL *easy;
    easy = curl_easy_init ();
    curl_easy_setopt (easy, CURLOPT_URL,
                      "https://grothoff.org/↘
                      →christian/");
    curl_easy_perform (easy);
    curl_easy_cleanup (easy);
    return 0;
```

libz: compression

```
#include <zlib.h>

int
compress (Bytef *dest, ulingf *destLen,
          const Bytef *source, uLong sourceLen)↘
          →;
```

libgcyrpt: hashing

```
#include <gcrypt.h>
enum gcry_md_algos {
  GCRY_MD_MD5, GCRY_MD_SHA1, GCRY_MD_SHA256, ↘
  →GCRY_MD_SHA512,
  GCRY_MD_SHAKE256, GCRY_MD_WHIRLPOOL, ↘
  →GCRY_MD_BLAKD2B_512, ... }

void
gcry_md_hash_buffer (int algo,
                     void *digest,
                     const void *buffer, size_t↘
                     → length);
unsigned int
gcry_md_get_algo_dlen (int algo);
```

Conclusion

Conclusion

- **Libraries are essential**
  - Avoid re-inventing the wheel!
- **A lot of possibilities**
  - Use free software libraries: be in control!
  - Still many too choose from!
  - Good API design is hard, learn from existing APIs!
  - Be aware of potential compatibility issues, even with libc.

Bibliography

- The GNU C Library: `https://www.gnu.org/software/libc/`