# Розробка під Linux

Візир Владислав

# В будь-якій ситуації — man

**$ man hello**

```
HELLO(1)                        User Commands                        HELLO(1)


NAME
       hello - friendly greeting program


SYNOPSIS
       hello [OPTION]...


DESCRIPTION
       Print a friendly, customizable greeting.
       -h, --help
              display this help and exit
       -v, --version
              display version information and exit
       -t, --traditional
              use traditional greeting format
       -n, --next-generation
              use next-generation greeting format
       -g, --greeting=TEXT
              use TEXT as the greeting message
```

# Мови програмування

- C, тому що це UNIX
- Інші мови теж окей
- Але API все одно на C

# Як зібрати програму

# gcc — стандартний комбайн

- Компіляція — це складно
- Єдина точка входу: gcc
- Альтернативи: Clang, Intel C Compiler

# Простий приклад

```
$ cat hello.c
#include <stdio.h>

int main(void)
{
    printf(“Hello world!\n”);
}
```

```
$ gcc hello.c
$ ls
a.out
$ ./a.out
Hello world!
```

# Трохи складніше

**$ cat hello.c**

```
#include <stdio.h>

int main(void)
{
    printf("Hello world!\n")
}
```

```
$ gcc -o hello hello.c
hello.c: In function ‘main’:
hello.c:6:1: error: expected ‘;’ before ‘}’ token
 }
 ^
```

$ *виправляємо проблему*

```
$ gcc -o hello hello.c
$ ls
hello
$ ./hello
Hello world!
```

# Декілька модулів

**$ cat main.c**

```
#include “hello.h”

int main(void)
{
    print_hello();
}
```

**$ cat hello.c**

```
#include “hello.h”
#include <stdio.h>

void print_hello(void)
{
    printf(“Hello world!\n”);
}
```

# Декілька модулів

```
$ gcc main.c hello.c

$ ./a.out
Hello world!
```

```
$ gcc -c main.c
$ gcc -c hello.c

$ ls
hello.c hello.o main.c main.o

$ gcc hello.o main.o

$ ./a.out
Hello world!
```

# Makefile — білд система

```
$ cat Makefile
hello: main.o hello.o
<------>gcc -o hello main.o hello.o

main.o: main.c hello.h
<------>gcc -c main.c

hello.o: hello.c hello.h
<------>gcc -c hello.c

clean:
<------>rm *.o hello
```

```
$ make clean
rm *.o hello

$ make
gcc -c main.c
gcc -c hello.c
gcc -o hello main.o hello.o

$ ./hello
Hello world!
```

# Makefile — *примітивна білд система*

- В make є змінні
- Але по суті він простий як дерево
- Пошук залежностей — це робота Autotools або Cmake

# Бібліотеки

# Види бібліотек

- Статичні (libastral.a = astral.lib)
  - вшиваються в програму при компоновці
  - дублюються в пам’яті
- Динамічні (libastral.so = astral.dll)
  - завантажуються в процесі виконання
  - за можливістю не дублюються

# Enterprise-level Greeting Framework

```
$ cat hello.h
#ifndef HELLO_H
#define HELLO_H

extern "C"
void print_hello(const char *who);

#endif // HELLO_H
```

```
$ cat hello.c
#include "hello.h"

#include <stdio.h>

void print_hello(const char *who)
{
    printf("Hello %s!\n", who);
}
```

# Як зібрати бібліотеку

## Статичну:

```
$ gcc -c hello.c
$ ar cr libhello.a hello.o
```

## Динамічну:

```
$ gcc -c -fPIC hello.c
$ gcc -shared -o libhello.so hello.o
```

# Як зібрати програму

```
#include “hello.h”

int main(void)
{
    print_hello();
}
```

```
$ gcc main.c -L. -lhello

$ ./a.out
Hello world!
```

# Makefile для програми

```
hello: main.o libhello.so
<------->gcc -o hello -L. -lhello main.c

main.o: main.c hello.h
<------->gcc -c main.c

libhello.so: hello.o
<------->gcc -shared -o libhello.so hello.o

hello.o: hello.c hello.h
<------->gcc -c -fPIC hello.c
```

# Нічого не працює!!1

```
$ make
   . . .

$ ./hello
./hello: error while loading shared
libraries: libhello.so: cannot open shared
object file: No such file or directory
```

```
$ ldd hello
        . . .
        libhello.so => not found
        . . .
```

# Нічого не працює!!1

```
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$PWD


$ ldd hello
        . . .
        libhello.so => /tmp/libhello.so (0x61000000)
        . . .
$ ./hello
Hello world!
```

# Динамічне завантаження бібліотек

```
void* dlopen(const char *file_name, int flag);

void* dlsym(void *handle, const char *symbol);

void dlclose(void *handle);
```

# Динамічне завантаження бібліотек

```
#include <dlfcn.h>
typedef void (*hello_fcn)(const char*); // покажчик на функцію
int main(void)
{
    void* handle = dlopen(“./libhello.so”, RTLD_LAZY);

    hello_fcn hello = dlsym(handle, “print_hello”);

    hello(“me”);

    dlclose(handle);
}
```

# Системне API

# Змінні оточення

```
#include <unistd.h>
extern char **environ;

char* getenv(const char *name);

int setenv(const char *name, const char *val,
           int overwrite);

int unsetenv(const char *name);
```

# Логування (API)

```
void openlog(char *ident, int option, int facility);
void syslog(int level, char *format, ...);
void closelog();
```

**Facilities:** `LOG_DAEMON, LOG_KERN, LOG_SYSLOG, LOG_USER, ...`

**Levels:**

`LOG_EMERG, LOG_ALERT, LOG_CRIT, LOG_ERR, LOG_WARNING, LOG_NOTICE, LOG_INFO, LOG_DEBUG`

# Логування (приклад)

```c
#include <syslog.h>

int main(void)
{
    openlog("test", LOG_PID, LOG_USER);

    syslog(LOG_DEBUG, "sending 5 messages");

    for (int i = 0; i < 5; i++)
        syslog(LOG_INFO, "info message %i", i);

    closelog();
};
```

```
$ tail /var/log/debug
Nov 01 17:12:45 localhost test[7291]: sending 5 messages

$ tail /var/log/messages
Nov 01 17:12:45 localhost test[7291]: info message 0
Nov 01 17:12:45 localhost test[7291]: info message 1
Nov 01 17:12:45 localhost test[7291]: info message 2
Nov 01 17:12:45 localhost test[7291]: info message 3
Nov 01 17:12:45 localhost test[7291]: info message 4
```

# Ввід-вивід

- Все є файли
  - власне файли
  - сокети, пайпи тощо
  - і спільна пам’ять
  - Навіть інформація про процеси (/proc)
- int — файловий дескриптор

# Ввід-вивід (API)

```
int open(const char* filename, int flags);
int close(int fd);

ssize_t read(int fd, void *buf, size_t len);
ssize_t write(int fd, void *buf, size_t len);

off_t lseek(int fd, off_t offset, int whence);
```

# Процеси

# Управління процесами (API)

```
pid_t getpid();   // process id

pid_t getppid(); // parent process id

pid_t fork();     // клонування процеса

int execve(const char *path, char *const argv[],
           char *const envp[]);
```

# Управління процесами (приклад)

```
#include <stdio.h>
#include <unistd.h>

int main(void)
{
    if (fork())
        printf(“parent”);
    else
        printf(“child”);
}
```

# IPC (міжпроцесна взаємодія)

- файли, змінні оточення, спільна пам’ять
- сокети, пайпи
- сигнали

# Пайпи

- Неіменовані (пайпи)

```
int pipe(int pipefd[2]);
```

- Іменовані (FIFO)

```
int mkfifo(const char *fifoname, mode_t mode);
```

# Пайпи (приклад)

```c
#include <stdio.h>
#include <string.h>
#include <sys/types.h>

int main(void)
{
    int pipefds[2];
    pipe(pipefds);

    pid_t pid = fork();

    // тепер в нас два процеса та кожен має два відкритих пайпа
```

# Пайпи (приклад)

```c
if (pid > 0)
{
    const char *str =
        "Valuable data\n";

    close(pipefds[0]);

    write(pipefds[1], str,
            strlen(str) + 1);

    close(pipefds[1]);
}
```

```c
else
{
    int len;
    char buf[1024];

    close(pipefds[1]);

    while ((len = read(pipefds[0], buf, 1024)))
        write(2, buf, len);

    close(pipefds[0]);
}
```

# Сигнали

- Для асинхронних подій
- Існують обмеження на обробку
- І найголовніше:

kill, killall, raise

# Сигнали

```c
#include <stdlib.h>
#include <signal.h>

void handler(int signal)
{
    printf("Terminating\n");
    exit(EXIT_FAILURE);
}

int main(void)
{
    signal(SIGSEGV, handler);
    (int*) NULL = 42;
}
```

SIGSEGV, SIGBUS, SIGILL

SIGINT, SIGKILL, SIGTERM

SIGUSR1, SIGUSR2, ...

# Потоки

# Управління потоками

```
int pthread_create(pthread_t             *thread,
                   pthread_attr_t const *attr,
                   void* (*start_routine)(void*),
                   void                  *arg);

int pthread_join(pthread_t thread, void **retval);

void pthread_exit(void *retval);

int pthread_cancel(pthread_t thread);
```

# Синхронізація

## М’ютекси:

`pthread_mutex_t`

- *`pthread_mutex_lock`*`(mutex)`
- *`pthread_mutex_trylock`*`(mutex)`
- *`pthread_mutex_unlock`*`(mutex)`

## І ще різні штуки:

- `pthread_spinlock_t`
- `pthread_rwlock_t`
- `pthread_barrier_t`

## Умовні змінні:

`pthread_cond_t`

- *`pthread_cond_wait`*`(condition, mutex)`
- *`pthread_cond_signal`*`(condition)`
- *`pthread_cond_broadcast`*`(condition)`

## Що далі:

- `$ man 7 pthreads`

# Які існують IDE?

- Qt Creator
- Eclipse, NetBeans, Clion, ...
- vim / Emacs

# Останній, сорок другий слайд

- компіляція
- Makefile
- бібліотеки
- системні API
- ввід-вивід
- процеси
- IPC
- потоки
- синхронізація
- IDE