PB173 Linux

03 Interakcia

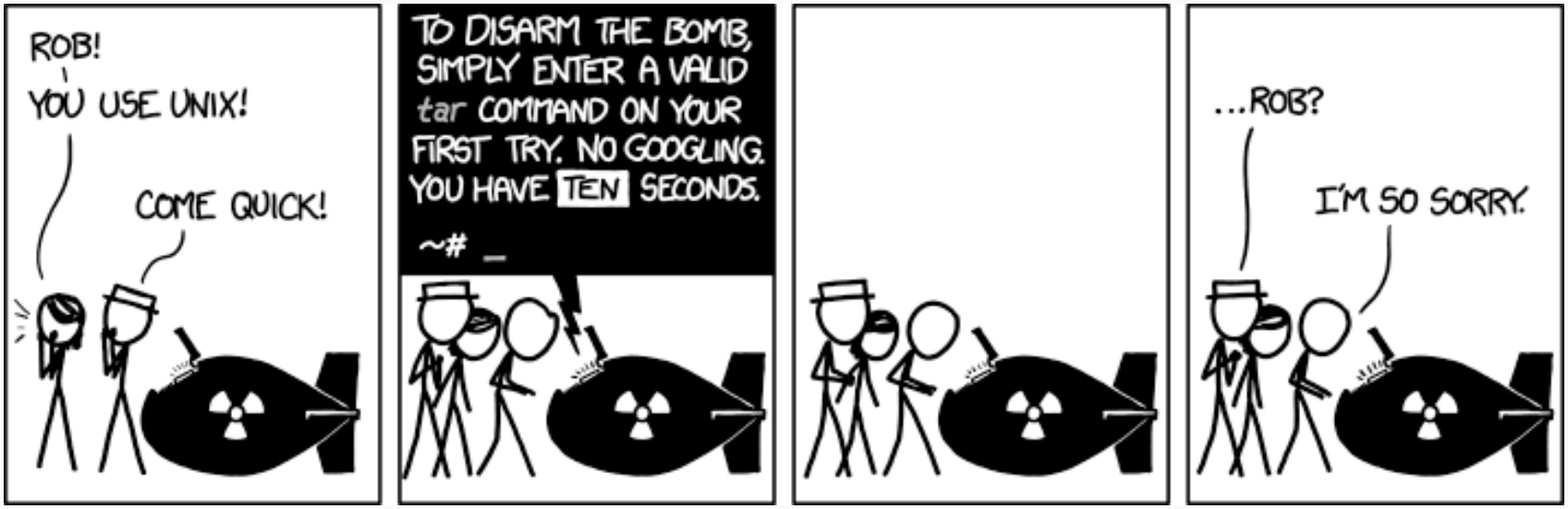
Roman Lacko xlacko1@fi.muni.cz

2022-09-29

Obsah

Interakcia

Interakcia


ROB!
YOU USE UNIX!
COME QUICK!
TO DISARM THE BOMB, SIMPLY ENTER A VALID tar COMMAND ON YOUR FIRST TRY. NO GOOGLING. YOU HAVE TEN SECONDS.
~# _
...ROB?
I'M SO SORRY.

Príkazový riadok

Základný spôsob predávania informácií programu.

argument Všeobecne akýkoľvek reťazec na príkazovom riadku.

prepínač (option)
Reťazec so špeciálnym prefixom a významom.

parameter Argument, ktorý nie je prepínač.

```
$ stat --format=%u Makefile
#      ————————                 Options
#               —— ————————— Parameters
#      ———————— —— ————————— Arguments
```

Príkazový riadok: Prepínače

Prepínače menia správanie alebo nastavenie programu.

Obvyklá syntax (α… znak, `A`… reťazec)

`-α` `ls -l`

`-αPARAM`

`-α PARAM` `make -f Makefile`

`-αβ…` „bundling,“ `grep -qE REGEX`

`--` koniec prepínačov

`--A` **(GNU)** `git log --oneline`

`--B=PARAM` **(GNU)** `diff --unified=1`

Existujú aj iné, menej časté schémy
viď napr. `man 1 javac`, `man 1 mcs`

Príkazový riadok: Prepínače

```
#include <unistd.h>
int getopt(int argc, char *const argv[], char *const optstring);
```

- Spracuje krátke (POSIX) prepínače.
- Volanie v cykle, až kým nevráti `-1`:

  ```
  while ((option = getopt(argc, argv, "ab:c")) != -1)
      // program -a -b ARG -c POS
  ```

- Pri chybnom prepínači vráti `?`
  a `optopt` nastaví na chybný prepínač.
- Parameter prepínača v `optarg`.
- Po spracovaní je `argv[optind]` prvý pozičný parameter.

Príkazový riadok: Dlhé prepínače (GNU)

```c
#define _GNU_SOURCE
#include <getopt.h>

struct option {
    const char *name;
    int  has_arg;
    int *flag;
    int  val;
}

int getopt_long(int argc, char *argv[], char *const optstring,
        const struct option longopts[], int *longindex);
```

- Spracuje dlhé a krátke prepínače (existuje aj `getopt_long_only()`).

Pole `longopts` **musí** byť ukončené vynulovanou štrukúrou.

Príkazový riadok: Ďalšie alternatívy

GNU Argp

- Viac parserov, ktoré sa dajú kombinovať.
- Automatické generovanie nápovedy.
- Viď `info argp`.

POPT Library

- Reentrantné funkcie.
- Aliasy.

Premenné prostredia

Sada reťazcov `NAME=VALUE`, ktoré program dedí od predka.

```
export EDITOR=notepad.exe
unset EDITOR

EDITOR=rm git commit
```

- Môžu ovplyvniť niektoré štandardné funkcie.
  `PATH`, `LANG`, `LC_*`, …
- `env(1)`, `printenv(1)` vypíšu premenné prostredia.
- `env VAR=VALUE… PROGRAM [ARGS]…`

Premenné prostredia: Prístup v programe

1. Globálny symbol `environ`

   ```
   extern char **environ;
   ```

2. Tretí parameter funkcie `main()`

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

3. Štandardné funkcie C

   ```
   #include <stdlib.h>
   char *getenv(const char *name);
   int setenv(const char *name, const char *value,
           int overwrite);
   int putenv(char *string);
   int unsetenv(const char *name);
   ```

Premenné prostredia: Bezpečnosť

- **Nedôveryhodný** vstup!
- Používajú sa typicky na nedôležité parametre spoločné pre viacero programov (preto „prostredie“).

Dezinfekcia prostredia

- Zmazanie jednej alebo všetkých premenných.

```
#include <stdlib.h>
int unsetenv(const char *name);
int clearenv();
```

- Prístup k premenným prostredia zo Set-UID programu (GNU):

```
#define _GNU_SOURCE
#include <stdlib.h>
char *secure_getenv(const char *name);
```

TUI, GUI

TUI Text User Interface

GUI Graphical User Interface

Umožňujú pohodlnejšie ovládanie zložitých programov.

Interaktívne ovládanie
GNU Readline, curses

Dynamický výpis
curses, TUI, GUI

Príliš veľa parametrov
Formuláre

TUI, GUI: GNU Readline

```
#include <stdio.h>
#include <readline/readline.h>
#include <readline/history.h>
char *readline(const char *prompt);
```

- Editácia vstupu (šípky, Ctrl-W, ...).
- Automatické doplňovanie.
- `man 3 readline`

TUI, GUI: ANSI útekové kódy

ANSI Escape Codes

- Sekvencie, ktoré interpretuje terminál ako príkazy.
- `0x1b <COMMAND BYTES>…`
- V C `0x1b` alebo `\e` (GNU)

Control Sequence Introducer

- Operácie s kurzorom a grafikou.
- `0x1b 0x5b <PARAMETER>… <COMMAND>`

  `CSI n [ABCD]` Pohyb kurzora.

  `CSI 6 n`, **`CSI <x> ; <y> H`**
  Zistí alebo nastaví polohu kurzora.

TUI, GUI: ANSI útekové kódy

Select Graphic Rendition

- Nastavenie dekorácií fontu.

`CSI 0 m` Vypne všetky dekorácie.

`CSI 1 m` Zapne tučné písmo.

`CSI <30-37,39|40-47,49> m`
Nastaví farbu popredia alebo pozadia.

`CSI <38|48> ; 5 ; <n> m`
Nastaví 8-bitovú farbu popredia alebo pozadia.

`CSI <38|48> ; 2 ; <R> ; <G> ; <B> m`
Nastaví 24-bitovú (TrueColor) farbu popredia alebo pozadia.

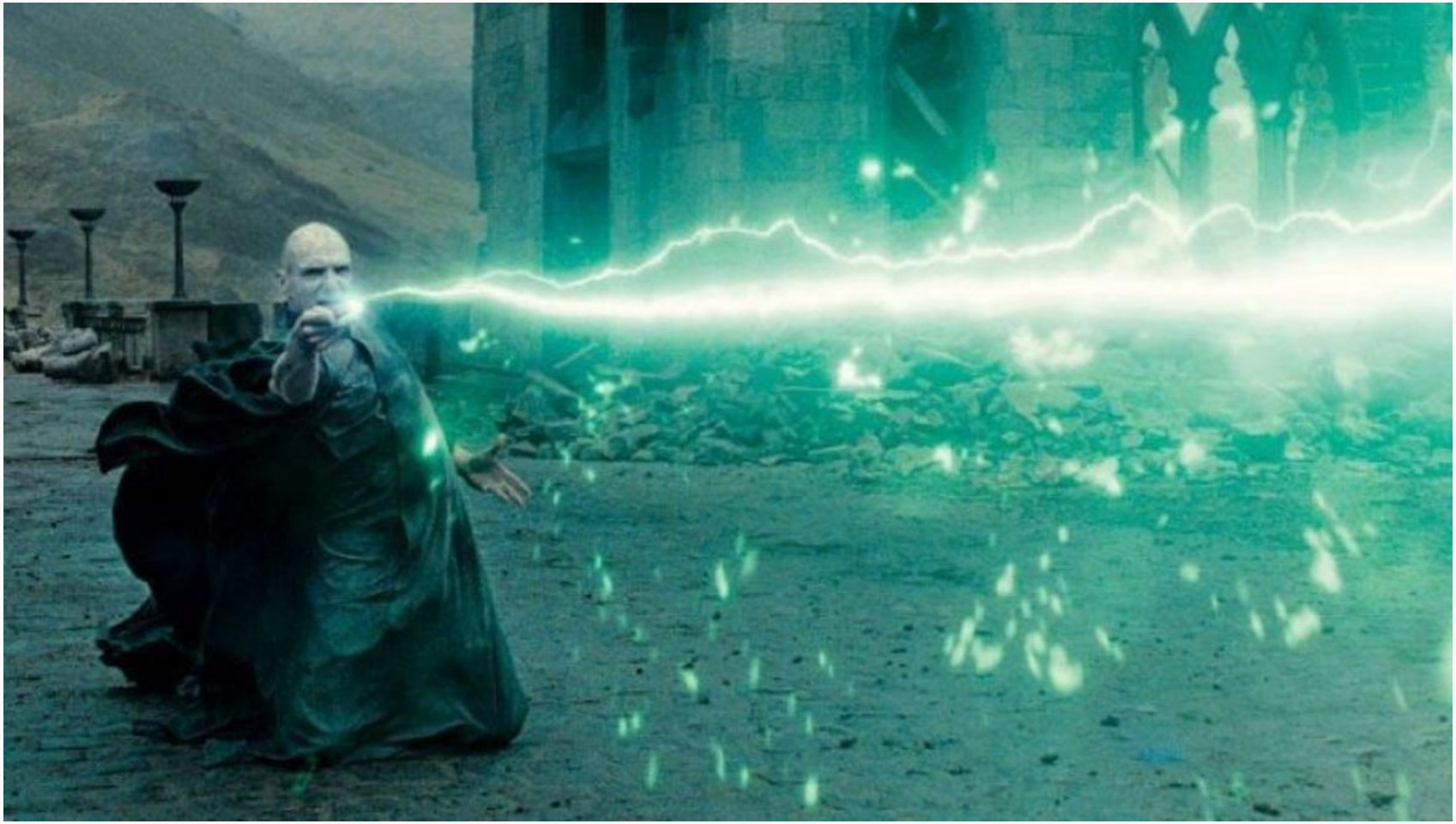
TUI, GUI: ANSI útekové kódy

CSI a SGR interpretuje terminál.
Ak výstup je bežný súbor, nemali by sa používať.

```
#include <unistd.h>
int isatty(int fd);
```

Niektoré terminály nemusia podporovať všetky SGR.
Podporované vlastnosti drží `terminfo(5)`.

TUI, GUI: Curses



TUI, GUI: Curses

NCurses

- (New) Cursor Optimisation
- Knižnica pre TUI.
- Komponenty rozhrania (*widgets*)
  tlačítka, zaškrtávacie políčka, ...

Alternatívy

- S-Lang
- newt
- libtickit

(a mnoho iných, ktoré nie sú v C)

TUI, GUI: Grafické rozhrania

GIMP Tool Kit (GTK)

- Skutočné grafické rozhranie.
- *Udalosťami riadené programovanie*.

Alternatívy

- Qt (C++)
- wxWidgets (C++)

Logovanie

Logovanie

Výstup programu čítajú *programátori* a *používatelia*.

Čo očakáva programátor?

- Presný zdroj v súbore.
- Hlášku, z ktorej je zjavná chyba.
- Všeobecne čo najviac *zmysluplných* správ.

Čo očakáva používateľ?

- Jasný popis toho, čo sa deje.
- Stručný popis chyby, ktorý by mohol nahlásiť.
- Čo najmenej správ, ktoré nesúvisia s jeho potrebami.

Logovanie

Delenie správ:

štandardný (chybový) výstup

- správy pre používateľa,
- ideálne v jazyku, ktorému rozumie (lokalizácia),

logovací mechanizmus

- správy pre programátora,
- rôzne úrovne správ (`info`, `error`, ...),
- zdroj správy (súbor a riadok).

Niekedy je používateľom iný programátor.

Logovanie: Typické riešenia

1. Vlastné logovacie mechanizmy
    - `__FILE__`, `__LINE__`, ...
    - Vynaliezanie kolesa.
2. `syslog`
    - Komplexný logovací mechanizmus v POSIX.
    - Implementuje všetky dôležité vlastnosti.
    - Obvykle len rozhranie, ktoré implementuje služba rsyslog, syslog-ng, systemd-journald, ...
3. `systemd-journald`
    - Moderný logovací mechanizmus.
    - Súčasť systemd.

Logovanie: Syslog

```c
#include <syslog.h>
void openlog(const char *ident, int options, int facility);
void syslog(int priority, const char *format, ...);
void closelog(void);
```

options LOG_PID, LOG_PERROR, ...

facility LOG_USER, LOG_DAEMON, LOG_KERN

level LOG_ERR, LOG_WARNING, LOG_NOTICE, LOG_INFO, LOG_DEBUG

```c
int setlogmask(int mask);
#define LOG_MASK(priority) /* ... */
#define LOG_UPTO(priority) /* ... */
```

Logovanie: systemd-journald

```c
#include <systemd/sd-journal.h>
int sd_journal_stream_fd(const char *ident, int priority,
        int prefix);
int close(int fd); // unistd.h

int sd_journal_print(int priority, const char *format, ...);
```

- Správy ukladá v štrukturovaném binárnom formáte.
- Rýchlejšie vyhľadávanie oproti textu.
- Je možné správy znova čítať (`sd_journal_get_data()`).

Preprocesor

Preprocesor

```
#define true false // happy debugging

#define ever (;;)
for ever {
    puts("Never gonna give you up...");
}
```

Preprocesor: `#include`

Vkladanie súborov

`#include <FILE>` Hľadá v systémových umiestneniach.
Tiež prehľadáva cesty zadané `-I PATH`.

`#include "FILE"` Hľadá relatívne od zdrojového súboru.

V akom poradí sa vkladajú hlavičky?

Neexistuje žiadna globálna konvencia.

- Pre súbor `a.c` sa vkladá ako prvé `a.h`, ak existuje.
- Poradie systémových hlavičiek obvykle „po skupinách“
  - Štandard C → POSIX → Projekt
  - Projekt → POSIX → Štandard C

Preprocesor: #define

Object-like Macros

```
#define DEBUG
#define COLOR 0
#define STRING "foo"
```

Function-like Macros

- Nahradí aj ().

```
#define EMPTY()
#define UNUSED(VAR) ((void) (VAR))
#define LOG(FMT, ...) \
    fprintf(stderr, FMT "\n", ## __VA_ARGS__)
```

Preprocesor: X-Macros

Preprocesor: `#define`

Preddefinované špeciálne makrá:

- `__FILE__` Názov aktuálneho súboru.
- `__LINE__` Riadok v súbore pred expanziou makier.
- `__DATE__`
- `__TIME__` Dátum a čas expanzie makra.

Preprocesor: Podmienky

```
#if CONDITION        // A && B, defined(X),
#elif CONDITION      // X >= 16, …
#else
#endif

#ifdef X             // if defined(X)
#ifndef X            // if !defined(X)
```

#define μ splní ifdef, **spôsobí chybu** s if

#define μ 0 splní ifdef, nesplní if

-Dμ splní ifdef aj if

Preprocesor: Špeciality

Pragmy

- Rozšírenia nad rámec jazyka

```
#pragma [PREFIX] PARAMS…
#pramga STDC …
#pragma GCC poison SYMBOL…
#pragma once
```

Diagnostika

```
#error "Error message"        // #pragma GCC error ...
#warning "Warning message"    // #pragma GCC warning ...
```

Preprocesor: Špeciality

- Expanzia hodnoty symbolu na reťazec

```
#define NAME Elrond
puts(#NAME);          // puts("Elrond");
```

- Spájanie symbolov

```
#define HANDLER(NAME) \
    void handle_ ## NAME(void)
HANDLER(list)       // void handle_list(void)
```

- Čiarka pred prázdnou elipsou (GNU)

```
#define LOG(FMT, ...) \
    fprintf(stderr, FMT "\n", ## __VA_ARGS__)
LOG("%s", name)     // fprintf(stderr, "%s" "\n", name)
LOG("Hello")        // fprintf(stderr, "Hello" "\n")
```

Preprocesor

Používajte len v dobre odôvodnených prípadoch.

- Ak niečo zvládnete urobiť funkciou, použite funkciu.
- Nemeňte tok programu z makra.

  Tj. žiadny `return`, `break`, ...

- Nepristupujte v makre k lokálnej premennej.
- Nedávajte do argumentov výrazy s bočnými efektami.

Zdroje

Zdroje

- POSIX Utility Conventions
- Parsovanie argumentov
  - man 3p getopt
  - man 3 getopt (+ GNU `getopt_long`)
  - GNU Argp
  - POPT
- Logovanie
  - man 3 syslog
  - man 3 setlogmask
  - man 3 sd-journal