Instrumente și tehnici de bază în informatică
Curs 2

Horațiu Cheval

De la sursă la executabil

- Codul sursă (C, C++, Rust, etc)
- Compilatorul traduce codul sursă în limbaj mașină
- Fișier obiect: rezultatul compilării unei surse
- Bibliotecă: fișier obiect care pune la dispoziție o anumită funcționalitate
- Executabilul poate fi
  - static: conține toate informațiile și bibliotecile necesare
  - dinamic: conține doar instrucțiunile proprii; bibliotecile necesare se leagă la runtime

Compilatorul

- Pe sistemele UNIX există în general comanda `cc` pentru apelarea unui compilator C (`c++` pentru C++).
- `cc` este doar un link către un compilator concret:
- Exemple de compilatoare:
  - `gcc` (GNU Compiler Collection)
  - `clang` (LLVM)
  - `msvc` (Windows)
- Compilatorul produce fișiere binare intermediare (`.o`) sau executabile.

Compilatorul – exemplu de utilizare

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

Opțiuni utile:

- `-o` – numele executabilului produs
- `-On` – nivelul de optimizare (0, 1, 2, 3)
- `-g` – adaugă simboluri pentru depanare
- `-Wall`, `-Werror` – controlul avertismentelor
- `-l` – opțiuni de linking. Exemplu: `$gcc main.c -o main -lcrypt`

Argumentele unui executabil

```
int main(int argc, char* argv[]) {
    ...
}
```

Exemplu: Pentru `$ ./args test 123` vom avea:

- `argc = 3`
- `argv[] = "./args", "test", "123"`

Coduri de ieșire; expresii logice

- Orice comanda are un cod de ieșire (0 - 255)
- Convenție:
  - 0 - succes
  - $\neq$ 0 - cod de eroare
- Valoarea returnată de `main` sau de un apel la `exit(3)`
- Variabila shell `$?` conține codul de ieșire al ultimei comenzi
- `cmd1 && cmd2` - execută `cmd1` și apoi `cmd2` doar dacă `cmd1` a avut succes
- `cmd1 || cmd2` - execută `cmd1` și apoi `cmd2` doar dacă `cmd1` a eșuat

Exemplu: compilăm un program și îl rulăm dacă s-a compilat cu succes

```
$ gcc main.c -o main && ./main
```

Debugging

- -g instruiește compilatorul să includă informații pentru debugging.
- Cu aceste simboluri putem folosi gdb pentru a analiza variabilele și liniile de cod.
- Fără -g, depanarea se reduce la nivel binar.

Exemplu:

```
$ gcc -g hello.c -o hello
$ gdb ./hello
(gdb) break main
(gdb) run
(gdb) print x
```

Optimizări și debugging

- Optimizările (`-O1`, `-O2`, `-O3`) pot modifica structura codului
- Variabilele temporare pot fi eliminate
- Pentru debugging fidel: `-O0 -g`

Exemplu:

```
$ gcc -g -O0 debug.c -o debug_O0
$ gcc -g -O3 debug.c -o debug_O3
```

Linking dinamic și libc

- În mod implicit, executabilele folosesc **linking dinamic**.
- Codul din biblioteci (ex: `libc.so.6`, biblioteca standard C) nu este inclus în executabil, ci încarcat la rulare de **dynamic loader-ul** (`ld.so`).

Exemplu:

```
$ gcc hello.c -o hello
$ ldd ./hello
    linux-vdso.so.1
    libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6
```

Exemplu: linking către biblioteca `crypt`

```
$ gcc main.c -o main -lcrypt
```

Static linking:

```
$ gcc hello.c -o hello -static
```

Executabil mai mare, dar independent de biblioteci externe.

Linking dinamic și libc

Exemplu: compilare și linking pentru o nouă bibliotecă

```
$ gcc -fPIC -c hello.c -o hello.o
$ cc -shared hello.o -o libhello.so
$ gcc main.c -o main -L. -lhello
```

Bibliotecile sunt căutate la runtime în `$LD_LIBRARY_PATH`.

Makefiles

make - folosit ca build system

```
CC = gcc
CFLAGS = -Wall -g

all: main

main: main.o utils.o
 $(CC) $(CFLAGS) -o main main.o utils.o

%.o: %.c
 $(CC) -c -o $@ $< $(CFLAGS)

clean:
 rm -f *.o main
```