Operációs rendszerek alapjai

4. gyakorlat

Mai gyakorlat

- Archívumkezelés, tömörítés
- Alapvető programnyelvek fordítása terminálból
  - C, C++
    - röviden make és cmake is
  - Python
  - LaTeX

Archívumok

- Egy vagy több fájl vagy mappa és metaadatok tárolására alkalmas
- Önmagában nem tömörítenek, de hatékonyabb a tárolás
  - 1 byte-os fájl elhasznál (legalább) egy IO blokkot (jellemzően 4 kB)
  - 1000 ilyen fájl elhasznál 4 MB-t
  - archívumot létrehozva szekvenciálisan tárolja, így elég 1 kB (nagyjából, plusz metaadatok)
  - természetesen ez nagyobb fájloknál nem számottevő, de kényelmes egy fájlt használni sok helyett
- Legfontosabb archívum fájltípusok
  - zip - általában tömörít is a deflate algoritmussal (Lempel-Ziv egy fajtája és Huffman kódolás)
  - tar

Tar - tape archive

- Az elkészült archívumot gyakran tarball-nak hívják
- A tarball-t tömöríti sokféle választható algoritmussal
  - ezek az algoritmusok önmagukban is léteznek, a tar parancs implementálja ezeket az algoritmusokat
- Választható algoritmusok:
  - gzip - .gz
    - LZ77 (g a GNU-ra utal, ugyanis a zip nem szabad szoftver)
  - bzip2 - .bz2
    - Burrows-Wheeler transzformáció után Huffman kódolás
  - xz - .xz
    - LZMA (Lempel-Ziv-Markov chain Algorithm)
  - Zstandard - .zst
    - hardcore algoritmus, 2016-ban fejlesztette Yann Collet (Facebook)
    - gyorsabb mindennél és jellemzően a tömörítés is hatékonyabb

Tömörítés terminálban - zip

- `zip archive.zip files`
  - `-r` kapcsoló: rekurzívan tömörít mappát
  - `-0` ... `-9` kapcsoló: tömörítés hatékonyságát állítja be (0 esetén csak archívum fájlt készít)
  - -@ kapcsoló: standard inputról fogadja a fájlneveket (tehát lehet pipe-olni is)
- `unzip archive.zip`
  - `-l` kapcsoló: archívum listázása
  - `-f` kapcsoló: freshen, tehát csak azokat a fájlokat csomagolja ki, amiből már létezik egy régebbi verzió
  - `-u` kapcsoló: update, olyan mint a -f, csak nem eddig létezett fájlokat is kicsomagol

Tömörítés terminálban - tar

- `tar` paranccsal történik minden
  - `-c` kapcsoló: tömörítés
  - `-x` kapcsoló: kitömörítés
  - `-z` kapcsoló: gzip
  - `-j` kapcsoló: bzip2
  - `-J` kapcsoló: xz
  - `--zstd` kapcsoló: Zstandard
  - `-v` kapcsoló: verbose mód, tehát sok információt kiír
  - `-f` kapcsoló: fájlok megadása
- Gyakorló feladatok megoldásai gyakorlatonként .tar.gz tömörítéssel vannak feltöltve
  - `tar -xzvf Gxx.tar.gz`

C, C++ terminálban

- `gcc` - .c és .cpp fájlokat fordít, C és C++ kódként kezelve őket
- `g++` - .c és .cpp fájlokat fordít, C++ kódként kezelve őket
- Alapvető használat
  - `g++ files -o binary`
- Kapcsolók
  - `-std=<standard>`, például `-std=c++11`
  - `-Werror`: minden warning error legyen
  - `-Wall`: (majdnem) minden warning bekapcsolása
  - `-Wextra`: maradék warning bekapcsolása
  - `-pedantic`: csak ISO C és ISO C++ kódok fordítása

Feladat

- Készíts egy Hello World! programot C++-ban!
- Fordítsd le `g++`-al!
  - `g++ main.cpp -o hello`

Feladat

- Készíts egy Hello World! függvényt külön fájlban (.hpp és .cpp)!
- Készíts egy main függvényt, ahol meghívod a függvényt!
- Fordítsd le `g++`-al!
  - `g++ main.cpp func.cpp -o hello`
  - `g++ *cpp -o hello`

Object files

- Gépi kóddá fordított forráskódot tartalmaz
- `-c` kapcsolóval a .cpp fájlokból .o fájlokat gyárthatunk
- Lehet .o fájlokat linkelni a bináris elkészítéséhez
  - akár kombinálhatjuk is a .o és .cpp fájlokat
  - például egy garantáltan jól megírt kódbázist érdemes előre lefordítani, így meggyorsítva a későbbi fordításokat
- Object file-ok és header-k lehetnek más mappában is
  - `-Idir` kapcsolóval hozzáadhatjuk a dir mappát az include path-hoz

Feladat

- Készíts object file-t a Hello World! függvényből!
  - `g++ -c func.cpp`
- Linkeld a .o és .cpp fájlt a bináris elkészítéséhez!
  - `g++ main.cpp func.o -o hello`
- Próbált ki más mappából is!
  - például csinálj egy include mappát, ahol csak header-k vannak
  - `g++ main.cpp func.o -o hello -Iinclude`

Könyvtárak

- Static library
  - .a fájl Linux-on és Mac-n, .lib fájl Windows-on
  - a használt rutinok gépi kódja bekerül a binárisba
  - ha változtatsz a library-n, akkor újra kell fordítani a library-t és az azt használó programokat is
- Shared library
  - .so fájl Linux-on és Mac-n, .dll fájl Windows-on
  - a használt rutinok információi bekerülnek a binárisba, de maga a kód nem
  - futás időben az operációs rendszer betölti a kódot (dynamic linking)
  - kisebb tárhelyet igényel, hiszen a shared library-t több program is használhatja
  - ha változtatsz a library-n, akkor csak a library-t kell újra fordítani

Könyvtárak linkelése

- `-lexample` kapcsolóval libexample.a vagy libexample.so fájlt linkeljük
- Könyvtárak lehetnek más mappában is
  - `-Ldir` kapcsolóval hozzáadhatjuk a dir mappát a linker path-hoz
- .a létrehozása
  - `ar` paranccsal
  - `d`, `p`, `r` opciókkal törölhetünk, listázhatunk, beilleszthetünk archívumba
  - `ar r libfunca.a func.o`
- .so létrehozása
  - `g++ -fpic func.cpp -shared -o libfuncso.so`
  - lehet .o fájlból is, ehhez eleve Position Independent Code kapcsolóval kellett fordítani
    - `g++ -c -fpic func.cpp`
  - elérhetőséghez hozzá kell adni a helyét az $LD_LIBRARY_PATH változóhoz

Feladat

- Hozz létre static library-t és shared library-t az object file-ból!
  - `g++ -c -fpic func.cpp`
  - `ar r libfunca.a func.o`
  - `g++ -fpic func.o -shared -o libfuncso.so`
- Próbáld fordítani és futtatni a binárist static és shared library-vel is!
  - `g++ main.cpp -o hello -Iinclude -L. -lfunca`
  - `g++ main.cpp -o hello -Iinclude -L. -lfuncso`

make I.

- Láthatjuk, hogy nem egyszerű munka a header-k behúzása és a library-k linkelése
  - gyorsan eszkalálódik a végrehajtandó `g++` parancs mérete
- make segítségével könnyen lehet akár több target-re build-elni
  - például .o fájlok és bináris készítése is
- Makefile nevű fájlban vannak leírva a szabályok
  - alapvető struktúra

    ```
    target: pre-req-1 pre-req2 …
          command
    ```
  - `clean` szabályt is megadhatunk a binárisok törlésére
  - változókat is használhatunk
  - `make install` - PATH-be teszi a binárist

make II.

```
hello: func.cpp main.cpp
     g++ -o hello func.cpp main.cpp -Iinclude
```

make III.

```
all: hello

hello: func.o main.cpp
	g++ -o hello func.o main.cpp -Iinclude

func.o: func.cpp
	g++ -c func.cpp

clean:
	rm func.o hello
```

make IV.

```
CC=g++
CFLAGS=-Iinclude

all: hello

hello: func.o main.cpp
	$(CC) -o hello func.o main.cpp $(CFLAGS)

func.o: func.cpp
	$(CC) -c func.cpp
```

make V.

```
CC=g++
CFLAGS=-Iinclude -Llib -lfunc

all: hello

hello: main.cpp
	$(CC) -o $@ $^ $(CFLAGS)
```

cmake I.

- Láthatjuk, hogy nem egyszerű munka a make használata
  - gyorsan eszkalálódik a végrehajtandó `Makefile` mérete
- cmake segítségével könnyen lehet Makefile-t generálni
- CMakeLists.txt nevű fájlban vannak leírva a szabályok
- Szokás létrehozni egy build mappát
  - ekkor a fordítás menete a következő
    - `cd build`
    - `cmake ..` - cmake elkészíti a Makefile-t
    - `make`

cmake II.

```
project(Hello)
add_executable(hello main.cpp func.cpp)
target_include_directories(hello PUBLIC include)
```

cmake III.

```
project(Hello)
add_executable(hello main.cpp)
target_include_directories(hello PUBLIC include)
target_link_directories(hello PUBLIC lib)
target_link_libraries(hello func)
```

Feladat

- Töltsd le a bc forráskódját!
- Csomagold ki!
- Hívd meg a `configure` script-t!
  - beállít rendszerfüggő változókat és generál Makefile-t
- Futtasd le a `make` parancsot!

Python terminálból

- `python`, illetve `python3` paranccsal futtatható
  - fájlkiterjesztés nem számít
  - ha nem adunk meg fájlt, akkor elindul interaktív módban
    - ekkor is figyelni kell a megfelelő indentálásra
    - ez gyors teszteléshez nagyon hasznos
  - használható interpreter direktívaként futtatható fájlban

LaTeX terminálból

- Donald Knuth írta 1978-ban
  - 2014 óta nincs frissítés, aktuális verziószám: 3.14159265
  - verziószám változáskor π újabb számjegyét hozzáírják
- Több fordító közül lehet választani
  - MiKTeX
  - **TexLive** - legteljesebb, legnagyobb projekt, legkönnyebb használni
  - MacTex
- `pdflatex filename`
  - `-draftmode` kapcsolóval nem használ képeket és nem készít kimenetet
  - tartalomjegyzék frissítéséhez kétszer kell futtatni (elsőre elkészül a .toc fájl)
  - irodalomjegyzék készítése: `pdflatex`, `bibtex` futtatása a .aux fájlra, újra `pdflatex`

Markup nyelvek

- Markdown - .md fájlok
  - gyakran használt, például git-n beágyazott README.md
  - Pandoc nevű programmal ajánlott fordítani (tud LaTeX-t is egyébként)
- groff - GNU troff
  - (t)roff Unix rendszerre írt szövegformázó program
  - különböző makrók
    - man - manual oldalak készítéséhez
    - me - cikkek írásához
    - ms - könyvek, jelentések
  - sokan preferálják a LaTeX-el szemben, mert sokkal egyszerűbb és gyorsabb

groff man

- groff_man(7)
- `groff -man filename`
  - standard outputra ír, alapvetően PostScript formátumban (.ps fájl)
    - ezt lehet fájlba irányítani, vagy profibb olvasóba pipe-olni (például zathura)
  - `-Tdev` paranccsal más formátumban is, például `-Tpdf` és `-Thtml`
    - ehhez általában fel kell telepíteni a `groff` csomagot
      - gitről letölthető a forráskód, `bc`-hez hasonlóan kell eljárni
- Megfelelő formátumú fájlt megnyithatunk a `man -l filename` paranccsal
  - szokás odaírni a manual jellegét is, például `ls.1`
- `gzip`-el való tömörítés után `man filename`
  - .gz fájlt `/usr/share/man/manx` mappába helyezve nem kell elérési útvonalat sem megadni
  - .gz fájl tartalmának kiírása `zcat` paranccsal

HF01 - 10 pont

- Másold a könyvtáradba /home/vagmi/Desktop mappából HF01.tar.gz fájlt (tempus szerveren)
  - Unix vagy Linux rendszerre töltsd le a HF01.tar.gz fájlt a honlapról
- Csomagold ki!
- HF01.1.gz manual oldalnak megfelelően írd meg a `HF01.sh` script-t!
- Tesztelhetsz a `test.sh` script-el, vagy manuálisan is!
- `bc` nincs egyik szerveren se és még cortex-n sincsenek a compile-hoz megfelelő könyvtárak
  - a fenti mappában van egy `bc` nevű python script, ami fogad `a+b` alakú kifejezéseket
  - a kifejezés értékét kiírja (command substitution-el elmenthető)
- Figyelj a megfelelő adatszerkezetek, vezérlési szerkezetek és függvények használatára!
- Leadás: töltsd fel a /home/vagmi/Desktop mappába NEPTUN.sh néven!
  - határidő: 2020.10.14. 23.59.59

Következő gyakorlat

- Szövegkezelés

Érdekességek, olvasnivalók

- Donald Knuth: The Art of Computer Programming
- Bisqwuit: How to Create a Compiler
- suckless: software that sucks less
- Scipiades Ármin: A Code::Blocks és az Eclipse káráról - avagy miért ne használjunk IDE-ket a programozásoktatásban