6. Forráskódok menedzselése

6.1. GNU make

6.2. Egyszerű Makefile létrehozása

Az előző fejezetben készítettünk ugyan bináris és forrás csomagot is, azonban ha a egy több forrásból és más könyvtárakra is építő csomagot készítünk, a make.sh jellegű fordító script nagyon bonyolulttá, nehezen kezelhetővé válhat. Ennek megoldására dolgozták ki a make script nyelvet, amelyet csomagok fordítására használhatunk. A make egy bináris futtatható program, amelynek paramétere a GNU make scriptnyelven írt Makefile és ez a Makefile tartalmazza a fordításhoz és linkeléshez szükséges szabályokat.

A Makefile-okban definiálható szabályok szerkezete a következő:

```
target : fuggosegek
      vegrehajtando_utasitas
```

Makefile-ok terminológiájában target-nek nevezzük a célt, amit meg akarunk valósítani, a függőségek között azokat a targeteket és fájlokat soroljuk fel, amelyeknek létezniük kell az aktuális target létrehozásához, a végrehajtandó utasítás pedig a target létrehozásához használandó parancs.

Az előző fejezetben fordított forrásállományokhoz a következő Makefile-t hozhatjuk létre:

6.2.1. forráskód: Makefile

```
all: mainStatic mainDynamic mainStaticAndDynamic mainOptional

testStaticLibrary.o: testStaticLibrary.h testStaticLibrary.c
  gcc -I. -c testStaticLibrary.c -o testStaticLibrary.o

testDynamicLibrary.o: testDynamicLibrary.h testDynamicLibrary.c
  gcc -I. -c -fPIC testDynamicLibrary.c -o testDynamicLibrary.o

libtestStaticLibrary.a: testStaticLibrary.o
  ar rcs libtestStaticLibrary.a testStaticLibrary.o

libtestDynamicLibrary.so: testDynamicLibrary.o
  gcc -shared -Wl,-soname,libtestDynamicLibrary.so.1 -o libtestDynamicLibrary.so.1.0.1
      testDynamicLibrary.o
  ln -fs libtestDynamicLibrary.so.1.0.1 libtestDynamicLibrary.so
  ln -fs libtestDynamicLibrary.so.1.0.1 libtestDynamicLibrary.so.1

mainStatic: mainStatic.c libtestStaticLibrary.a
  gcc -I. -static mainStatic.c -L. -ltestStaticLibrary -o mainStatic
```

```
mainDynamic: mainDynamic.c libtestDynamicLibrary.so
  gcc -I. mainDynamic.c -L. -ltestDynamicLibrary -o mainDynamic

mainStaticAndDynamic: mainStaticAndDynamic.c libtestStaticLibrary.a
    libtestDynamicLibrary.so
  gcc -I. mainStaticAndDynamic.c -L. -Wl,-Bstatic -ltestStaticLibrary -Wl,-Bdynamic -
      ltestDynamicLibrary -o mainStaticAndDynamic

mainOptional: mainOptional.c
  gcc mainOptional.c -o mainOptional
```

A Makefile-t tartalmazó könyvtárban állva a

```
make target_nev
```

utasítással hozhatjuk létre a megnevezett target-et. A make rendszer tehát lehetőséget biztosít arra, hogy a Makefile-t felhasználva könyvtárunknak csak egyes részeit fordítsuk le. Paraméter nélkül a make csak az első target-ethez tartozó szabályt alkalmazza. Létrehozhatunk további targeteket, amelyeknek csak előfeltételeket adunk meg, így azokkal bizonyos más targetek létrehozását fogjuk össze, megadható továbbá olyan target is, amelyhez nem tartozik előfeltétel. Gyakran használt target-ek az "all", "clean" és az "install". Egészítsük ki előző Makefile-unkat a következő targetekkel:

```
all: mainStatic mainDynamic mainStaticAndDynamic mainOptional

clean:
      rm mainStatic mainDynamic mainStaticAndDynamic mainOptional libtestDynamicLibrary.
          so \
      libtestDynamicLibrary.so.1 libtestDynamicLibrary.so.1.0.1 libtestStaticLibrary \
      testStaticLibrary.o testDynamicLibrary.o

install: mainStatic mainDynamic mainStaticAndDynamic mainOptional
  cp main* /usr/bin
  cp libtestDynamicLibrary.so* /usr/lib
  cp libtestStaticLibrary.a /usr/lib
  cp *.h /usr/include
```

Az **all** target mindig a Makefile első target-e, így target név nélkül a **make** utasítás mindent létrehoz. A **clean** targetet a létrehozott fájlok törlésére használják, míg az **install** target elhelyezi a bináris állományokat a megfelelő /**usr**/**bin** és /**usr**/**lib** könyvtárakban.

6.2.1. Fordítási opciók

A Makefile-ok egyik erőssége, hogy változókat hozhatunk létre, ezáltal rövidítve a fordítási parancsokat. A változók a környezeti változókhoz hasonlóan működnek, nem kell deklarálni őket, értékadásnál jönnek létre, és hivatkozni az értékükre $**(VALTOZO_NEV)** módon lehet.

A változók használatának szemléltetésére létrehozom a CFLAGS változót, amelyben a források fordításánál használandó kapcsolókat adom meg és a LIB_DIRS változót, amelyben a -L kapcsolóval megadandó könyvtárakat fogjuk össze. Hozzuk létre tehát az alábbi változókat a Makefile első soraiban, és módosítsuk a megfelelő fordítási és linkelési parancsokat (példaként megadom a mainDynamic targethez tartozó parancs módosítását):

```
CFLAGS=-I.
LIB_DIRS=-L.

mainDynamic: MainDynamic.c libtestDynamicLibrary.so
      gcc $(CFLAGS) mainDynamic.c $(LIB_DIRS) -ltestDynamicLIbrary -o mainDynamic
```

Változókhoz hozzáadhatunk további értékeket, azaz sztringeket fűzhetünk hozzájuk a += operátorral, például:

```
CFLAGS+=-O3
```

a CFLAGS változóhoz hozzáfűzi a -O3 kapcsolót.

Változókhoz értéket rendelhetünk a make utasításban is, a következő módon:

```
make CFLAGS=-O3
```

Ebben az esetben a CFLAGS változó értéke a Makefile során az -O3 kapcsoló lesz, függetlenül attól, hogy a Makefile elején értékeket rendeltünk hozzá, azaz a Makefile-ban történt értékadások felül lesznek definiálva.

Az utolsó konstrukció, amit a GNU make-hez kapcsolódóan megnézünk, az a feltételes szerkezet, ugyanis változók értékétől függően feltételesen is beállíthatjuk a fordítási flag-eket. Egy feltételes szerkezet általános formája:

```
ifeq (arg1,arg2)
   utasitasok
[else
   utasitasok]
endif
```

Az fenti feltételes szerkezet az argumentumok egyenlőségét vizsgálja, minden esetben. A [] zárójel az **else** ág opcionalitását jelzi. Felhasználva az utóbbi három eszközt, módosítsuk úgy a Makefile-t, hogy a make hívásban beállított BUILD=release vagy BUILD=debug változó alapján hozzáfűzze a -O3 vagy -O0 kapcsolót a CFLAGS változóhoz a Makefile-ban! A módosított, végleges Makefile az alábbi lesz:

6.2.2. forráskód: Makefile

```
INCLUDE_DIRS:=-I.
LIB_DIRS:=-L.

ifeq ($(BUILD),debug)
    CFLAGS+=-O0 -g
endif
ifeq ($(BUILD),release)
    CFLAGS+=-O3
endif

all: mainStatic mainDynamic mainStaticAndDynamic mainOptional

testStaticLibrary.o: testStaticLibrary.h testStaticLibrary.c
  gcc $(CFLAGS) $(LIB_DIRS) $(INCLUDE_DIRS) -c testStaticLibrary.c -o testStaticLibrary.
      o

testDynamicLibrary.o: testDynamicLibrary.h testDynamicLibrary.c
  gcc $(CFLAGS) $(LIB_DIRS) $(INCLUDE_DIRS) -c -fPIC testDynamicLibrary.c -o
      testDynamicLibrary.o

libtestStaticLibrary.a: testStaticLibrary.o
  ar rcs libtestStaticLibrary.a testStaticLibrary.o

libtestDynamicLibrary.so: testDynamicLibrary.o
  gcc -shared -Wl,-soname,libtestDynamicLibrary.so.1 -o libtestDynamicLibrary.so.1.0.1
      testDynamicLibrary.o
  ln -fs libtestDynamicLibrary.so.1.0.1 libtestDynamicLibrary.so
```

```
  ln -fs libtestDynamicLibrary.so.1.0.1 libtestDynamicLibrary.so.1

mainStatic: mainStatic.c libtestStaticLibrary.a
  gcc $(CFLAGS) -static mainStatic.c $(LIB_DIRS) $(INCLUDE_DIRS) -ltestStaticLibrary -o
      mainStatic

mainDynamic: mainDynamic.c libtestDynamicLibrary.so
  gcc $(CFLAGS) mainDynamic.c $(LIB_DIRS) $(INCLUDE_DIRS) -ltestDynamicLibrary -o
      mainDynamic

mainStaticAndDynamic: mainStaticAndDynamic.c libtestStaticLibrary.a
    libtestDynamicLibrary.so
  gcc $(CFLAGS) mainStaticAndDynamic.c $(LIB_DIRS) $(INCLUDE_DIRS) -Wl,-Bstatic -
      ltestStaticLibrary -Wl,-Bdynamic -ltestDynamicLibrary -o mainStaticAndDynamic

mainOptional: mainOptional.c
  gcc $(CFLAGS) mainOptional.c -o mainOptional

clean:
  rm testStaticLibrary.o testDynamicLibrary.o libtestStaticLibrary.a
      libtestDynamicLibrary.so mainStatic mainDynamic mainStaticAndDynamic mainOptional

install: mainStatic mainDynamic mainStaticAndDynamic mainOptional
  cp main* /usr/bin
  cp libtestDynamicLibrary.so* /usr/lib
  cp libtestStaticLibrary.a /usr/lib
  cp *.h /usr/include
```

Ezen új Makefile használata a következő:

- **make BUILD=debug** lefordít mindent debug módban
- **make BUILD=release** lefordít mindent release módban
- **make clean** letöröl minden állományt, amit létrehozott
- **make install** bemásolja az létrehozott könyvtárakat és programokat a megfelelő rendszerkönyvtárakba

A könyvtárak fordítása és linkelése összetett feladat, amely nehezen automatizálható, azonban az egyszerű forráskódok tárgykódra történő fordítását lehet tovább egyszerűsíteni, a következő módon: a make automatikusan tudja, hogy .o kiterjesztésű fájl előállításához a gcc parancsot kell futtatni egy azonos nevű, .c kiterjesztésű forráskód paraméterrel, ezért ilyen egyszerű esetben nem kell kiírnunk a fordítási parancsot, tehát a

```
testStaticLibrary.o: testStaticLibrary.h testStaticLibrary.c
      gcc \$(CFLAGS) -c testStaticLibrary.c -o teststaticLibrary.o
```

szabály egyszerűsíthető az alábbival

```
testStaticLibrary.o: testStaticLibrary.h testStaticLibrary.c
```

Az egyetlen kérdés, hogy hogyan adhatunk meg fordítónak szóló kapcsolókat ha nem mi szerkesztjük a fordítási parancsot? A válasz az, hogy a CFLAGS változó egy speciális változó abban az értelemben, hogy a fenti egyszerűsítés esetén a CFLAGS változó értékét a make belerakja a fordítási parancsba, így megfelelően beállított CFLAGS változóval (esetünkben) ugyanazt az eredményt érjük el, mint ha teljesen kiírnánk a szabályt.

6.1. Feladat: Alakítsuk át a fenti létrehozott **Makefile**-t a feltételes szerkezet használatával úgy, hogy a fordítási sorban beállított **WARNINGS=yes**, **BUILD=debug/relese**, **OPTIONAL=yes** változók értelmében a warning-ok, a fordítási mód és az opcionális dinamikusan linkelendő könyvtár használata megfelelően beállítódjon.

6.1. Megoldás:

6.2.3. forráskód: Makefile

```
INCLUDE_DIRS:=-I.
LIB_DIRS:=-L.

ifeq ($(BUILD),debug)
    CFLAGS+=-O0 -g
endif
ifeq ($(BUILD),release)
    CFLAGS+=-O3
endif
ifeq ($(WARNINGS),yes)
    CFLAGS+=-Wall -Wextra
endif
ifeq ($(OPTIONAL),yes)
    O:=-DUSE_DYNAMIC_LIBRARY $(INCLUDE_DIRS) $(LIB_DIRS) -ltestDynamicLibrary
endif

all: mainStatic mainDynamic mainStaticAndDynamic mainOptional

testStaticLibrary.o: testStaticLibrary.h testStaticLibrary.c
  gcc $(CFLAGS) $(LIB_DIRS) $(INCLUDE_DIRS) -c testStaticLibrary.c -o testStaticLibrary.
      o

testDynamicLibrary.o: testDynamicLibrary.h testDynamicLibrary.c
  gcc $(CFLAGS) $(LIB_DIRS) $(INCLUDE_DIRS) -c -fPIC testDynamicLibrary.c -o
      testDynamicLibrary.o

libtestStaticLibrary.a: testStaticLibrary.o
  ar rcs libtestStaticLibrary.a testStaticLibrary.o

libtestDynamicLibrary.so: testDynamicLibrary.o
  gcc -shared -Wl,-soname,libtestDynamicLibrary.so.1 -o libtestDynamicLibrary.so.1.0.1
      testDynamicLibrary.o
  ln -fs libtestDynamicLibrary.so.1.0.1 libtestDynamicLibrary.so
  ln -fs libtestDynamicLibrary.so.1.0.1 libtestDynamicLibrary.so.1

mainStatic: mainStatic.c libtestStaticLibrary.a
  gcc $(CFLAGS) -static mainStatic.c $(LIB_DIRS) $(INCLUDE_DIRS) -ltestStaticLibrary -o
      mainStatic

mainDynamic: mainDynamic.c libtestDynamicLibrary.so
  gcc $(CFLAGS) mainDynamic.c $(LIB_DIRS) $(INCLUDE_DIRS) -ltestDynamicLibrary -o
      mainDynamic

mainStaticAndDynamic: mainStaticAndDynamic.c libtestStaticLibrary.a
    libtestDynamicLibrary.so
  gcc $(CFLAGS) mainStaticAndDynamic.c $(LIB_DIRS) $(INCLUDE_DIRS) -Wl,-Bstatic -
      ltestStaticLibrary -Wl,-Bdynamic -ltestDynamicLibrary -o mainStaticAndDynamic

mainOptional: mainOptional.c
  gcc $(CFLAGS) $(O) mainOptional.c -o mainOptional

clean:
  rm testStaticLibrary.o testDynamicLibrary.o libtestStaticLibrary.a
      libtestDynamicLibrary.so mainStatic mainDynamic mainStaticAndDynamic mainOptional

install: mainStatic mainDynamic mainStaticAndDynamic mainOptional
  cp main* /usr/bin
  cp libtestDynamicLibrary.so* /usr/lib
  cp libtestStaticLibrary.a /usr/lib
  cp *.h /usr/include
```

6.2. Feladat: Módosítsuk a fenti **Makefile**-t úgy, hogy a dinamikus könyvtár használatát szabályozó **define** direktíva a **mainOptional.c**-be a **config.h** header-ön keresztül kerüljön be!

6.2. Megoldás: A megoldás rafinált, ugyanis **Makefile**-okban nem helyezhetünk el csak úgy shell parancsokat. Létre kell hoznunk két új target-et:

```
config-use:
    echo "#define USE_DYNAMIC_LIBRARY" > config.h config-
nouse:
    echo "" > config.h
```

Ezt követően a **make.sh** esetén született megoldáshoz hasonlóan az **O** változónak nem adjuk értékül a **–DUSE_DYNAMIC_LIBRARY** kapcsolót, viszont egy új, **USING_TARGET** nevű változónak értékül adjuk a **config.h** fájlba író targetek valamelyikét az **if** feltételének teljesülésétől függően:

```
ifeq ($(OPTIONAL),yes)
  O:=$(INCLUDE_DIRS) $(LIB_DIRS) -ltestDynamicLibrary
  USING_TARGET:=config-use
else
  USING_TARGET:=config-nouse
endif
```

Nincs más hátra, mint a **USING_TARGET** változó értékét, azaz a **config.h** helyes kitöltését, mint target-et elhelyezni a **mainOptional** target előfeltételei között:

```
mainOptional: mainOptional.c $(USING_TARGET)
  gcc $(CFLAGS) $(O) mainOptional.c -o mainOptional
```

A fenti módosításokat követően

```
make BUILD=debug WARNINGS=yes OPTIONAL=yes
```

paranccsal a debug fordítást, a warningok használatát és az **makeOptional** alkalmazásban az opcionális dinamikus könyvtár használatát írhatjuk elő.

6.2.2. Külső könyvtárak használata

Amiről eddig nem volt szó, az külső könyvtárak használatának módja. A következő példában a **libpng** könyvtárat fogjuk használni. Az alábbi példaalkalmazás az első parancssori argumentumaként kapott fájlnevet megnyitja, beolvassa az első 8 bájtját, és a **libpng** könyvtár egy függvényével eldönti, hogy a paraméterként kapott fájlnév png szerkezetű fájlt takar-e vagy sem. Az alkalmazás forráskódja tehát:

6.2.4. forráskód: mainPNG.c

```
#include <png.h>

int main(int argc, char** argv)
{
    FILE* fp= fopen(argv[1], "rb");
    char header[8];

    if (!fp)
  return 1;
    fread(header, 1, 8, fp);

    printf("%d\n", png_sig_cmp(header, 0, 8));

    return 0;
}
```

Az alkalmazás tehát 0-t ír a képernyőre, ha az argumentum fájl PNG szerkeztű és 1-t, ha nem az. Ahhoz, hogy le tudjuk fordítani az alkalmazást a korábbi példákhoz hasonlóan, három dolgot kell megadnunk a fordítónak, linkernek:

- a **libpng** könyvtár header-jeinek elérési útját (**–I** kapcsoló);
- a **libpng** könyvtár lib-jeinek elérési útját (**–L** kapcsoló);
- a **libpng** könyvtár lib-jeinek nevét, ügyelve arra, hogy a **lib** szócskát ne írjuk ki és kiterjesztéseket ne használjunk (**–l** kapcsoló).

Ezek a kapcsolók megadhatók explicit elérési utakkal egy adott rendszeren, azonban ez nagyban gyengíti alkalmazásunk hordozhatóságát. Ahhoz, hogy a hordozhatóságot megtartsuk, egy általánosabb megoldást kell találnunk, amelyet a **pkg–config** alkalmazás biztosít számunkra. A **pkg–config** egy interoperábilis alkalmazás. A koncepció a következő: minden forráskód csomag, amelyet feltelepítünk egy rendszerre, installál egy **.pc** kiterjesztésű úgynevezett package-config fájlt. Ezen package-config fájl az installált könyvtár jellemzőit tartalmazza, azaz a header fájlok elérési útját, a library-k elérési útját, a library-k nevét és még néhány egyszerű leíró tulajdonságot. A **libpng** könyvtár .pc fájlja a saját rendszeremen:

6.2.5. forráskód: /usr/lib/pkgconfig/libpng.pc

```
prefix=/usr
exec_prefix=${prefix}
libdir=${exec_prefix}/lib
includedir=${prefix}/include/libpng12

Name: libpng
Description: Loads and saves PNG files
Version: 1.2.44
Libs: -L${libdir} -lpng12
Libs.private: -lz -lm
Cflags: -I${includedir}
```

A fenti package-config két fontos változót definiál:

- **Cflags**, amely a fordítónak szóló flag-eket tartalmazza (**–I**),
- **Libs**, amely a linkernek szóló flag-eket tartalmazza (**–L**, **–l**).

A package-config fájlok standard helye a **/usr/lib/pkgconfig** könyvtár, azonban tetszőleges helyen szereplő package-config fájlok is teljesértékűen használhatóak, ha elérési útjuk szerpeel a **PKG_CONFIG_PATH** környezeti változóban. A **pkg–config** alkalmazás parancssori argumentumként egy könyvtár nevét várja (pl **libpng**), és a **––cflags** kapcsoló hatására kiírja azon fordítási flag-eket, amelyekre az argumentumként kapott könyvtár használatához szükség van, a **––libs** kapcsoló hatására pedig azon linker flag-eket, amelyek az argumentumként kapott könyvtár linkeléséhez kellenek. Ezt követően sokkal általánosabb fordítási parancsot kapunk, ha abba beágyazzuk a **pkg–config** hívását:

```
gcc `pkg-config --cflags libpng` `pkg-config --libs libpng` -o mainPNG mainPNG.c
```

A fenti sort hozzáadva az eddig használt **Makefile**-hoz egy megfelelő **mainPNG** targettel, épp a kívánt működést kapjuk, anélkül, hogy bármi rendszerspecifikusat írtunk volna a kódba. Ha a **libpng** könyvtár nincs installálva, a **pkg–config** nem ír ki semmit, így fordítási hibához jutunk, hiszen a fordító nem fogja megtalálni a **png.h** headert.