SEGÉDLET

A "PROGRAMFEJLESZTŐ ESZKÖZÖK" c.

MÉRÉSHEZ

Negyedik kiadás

Készült:

A Távközlési és Médiainformatikai Tanszék
Távközlési laboratóriumában.
2022. február

Összeállította:

Horváth György, tanszéki mérnök
Skopkó Tamás, adjunktus

Tartalomjegyzék

A segédlet használatáról

Az alábbi segédlet a Villamosmérnöki szak, Infokommunikációs Hálózatok szakirány laboratóriumainak "Programfejlesztő eszközök" című méréséhez készült. Elsődleges feladata hogy bevezesse a hallgatókat a csoportos programozói munka módszereibe, és több platformon (Windows, Linux, Java) is, példa programokon keresztül demonstrálja azt.

Anyagunk a Távközlési Laboratórium eszközeinek felhasználásával bemutat néhány, a villamosmérnöki gyakorlatban tipikus technológiai láncot, melyek felhasználásával mind általános célú felhasználói, mind pedig hardver specifikus, vagy több processzorból álló heterogén program rendszerek állíthatók elő.

Feltételezzük az olvasóról, hogy rendelkezik az IBM-PC típusú, hálózatba kapcsolt személyi számítógépek használatához szükséges általános, valamint alapszintű „C” nyelvi ismeretekkel, és ajánlott valamilyen processzor (ix86, TMS, PIC) assembly nyelvű programozói ismerete.

Az anyag rövid tartalma:

- A technológiai lánc elemei
- Szövegszerkesztők (the, vi) és forráskód
- Fordító programok (gcc, cc, javac) és object
- A linker, és library manager (link, ld, lib, ar, jar)
- Programtöltés, hibakeresés (gdb, jdb)
- A program maintenance utility (make), és Makefile
- Archiválás (zip,tar,gz) és verziószámozás

A segédlet ONLINE verziója, és a méréshez tartozó feladatlap a

https://smartcomlab.tmit.bme.hu/alpha/meresek/ttmer99.htm

alatt találhatók.

Bevezető

Programok... Szinte észrevétlenül kúszik be életünk minden területére az elektronika, és a számítástechnika. Ez pedig egyre inkább (mikro-)számítógépeket jelent az egyszerű mosógéptől a mobiltelefonon keresztül a videó-játékokig, beleértve természetesen magát a (személyi-)számítógépet is.

Ezek a gépek perifériáikon keresztül jelzéseket, információkat fogadnak-küldenek, és működésük módját memóriájukba égetett vagy letöltött instrukciók szabják meg. Az utasításkódokat, az alapvető kiindulási adatokat, valahol-valamikor programozók alkották. Elképzeléseiket forrásnyelven megírt programsorokba öntve, amely nem 'közönséges halandó", hanem fordítóprogram (vagy másik programozó) számára készült olvasmány, bírták működésre hétköznapi dolgaink gépesített világát.

A sarokban lévő mosógép processzorának néhány-soros program szabja meg a különféle mosási, öblítési fázisok vezérlését. Az ősöreg irodai fax processzorának már "beszélgetni" kellett tudnia a vonal túlsó végén lévő készülékkel, melynek protokollja néhány száz sorban leírható. Egy digitális tároló oszcilloszkóp működtetése néhány ezer sor volt – mára már tipikusan konzum operációs rendszer grafikus felületébe épül be.

Számítógépünk elé leülve egy néhány perces munka során is programok sokaságát használjuk, kezdve az operációs rendszer parancsértelmezőjétől az egyszerű szövegszerkesztőn át a színes grafikus felhasználói felületet nyújtó szolgáltatásokig. Internetes létünket ezernyi apró programocska segíti/keseríti tabletünkön vagy a kiszolgálón futva.

Hány sor lehet egy operációs rendszer (pl. egy Linux base system, vagy a Microsoft Windows)? Egy ember írta, vagy egy egész csapat? Hány munkaóra fekszik bennük? Hány "névtelen" programozó munkájának eredményét keltjük életre ha rákattintunk az egérrel néhány ikonra a grafikus felületen?

A választ a programozói munka gyakorlata adja meg. Az alábbi anyag korántsem törekszik a teljességre. Felvillant néhányat azok közül az alapvető módszerek közül, melyeknek a felhasználásával az olvasó átlépheti az "egyszemélyes" programfejlesztés kereteit.

Kellemes programozást és hibavadászatot kíván,

A szerző

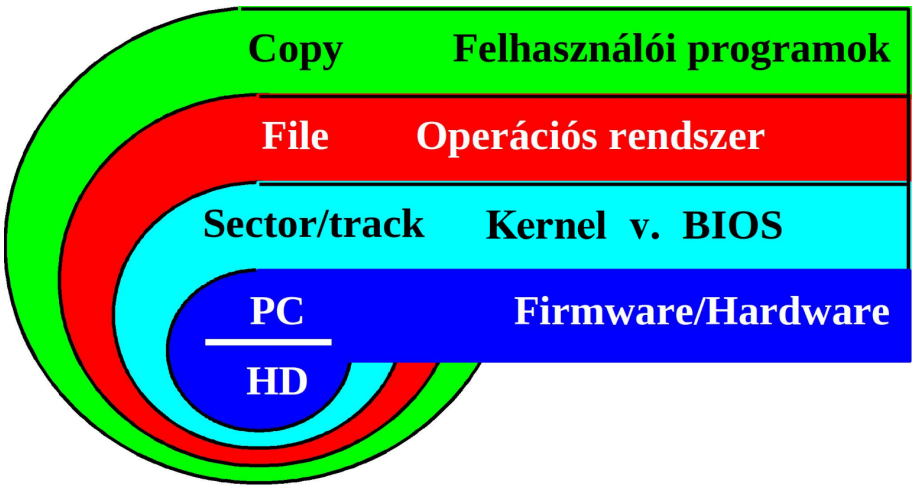
A technológiai lánc

Ahogy egy csúzli előállításához is fel kell építenünk egy technológiai láncot (tervezés; nyersanyag = faág + gumiszalag + cukorspárga + bőrdarab; szerszámok = fűrész + kés; gyártás = méretre szabás + illesztés + rögzítés; tesztelés = konzervdoboz 20 méterről; eladás; stb..), úgy ezt a programok esetén is meg kell tennünk.
A **rendszertervezés** a felhasználó, vagy egy nagyobb technológiai lánchoz kapcsolódva, más rendszertervező kívánalmaihoz igazodó **specifikáció** alapján történik.

A specifikációban rögzítjük hogy **mit kell "tudnia"** a programunknak, és a rendszertervben kell kiötlenünk, hogy milyen **eszközök** és **erőforrások** felhasználásával, **hogyan** állítsuk elő azt. Gondoskodni kell róla, hogy a munka jól legyen **dokumentálva**, hogy később az esetleges hibák forrása könnyen visszakereshető és javítható legyen (minőségbiztosítás).

Esetünkben a lánc végén egy vagy több program illetve programmodul állhat. A villamosmérnöki gyakorlatban ez a végtermék igen változatos formákat ölthet; a lemezen tárolt felhasználói programcsomagtól (Applications) egészen az egy- vagy több célprocesszorba égetett kódig (Firmware).

A hierarchikus programozástechnikához szükséges, hogy tisztázzuk mit is programozunk tulajdonképpen. Érdemes tanulmányozni egy PC programozói szemmel felépített **hierarchikus** modelljét, a COPY parancson keresztül:



A "tetőről" indulva, a Windows parancs sorába kiadott COPY parancs az operációs rendszer **parancsértelmező**jének szól, ami az **applikációs** (felhasználói programok) rétegen helyezkedik el, ahol általában kezelői felülettel rendelkező programok vannak. (nem szervizdémon)

A parancs paramétereként megadott file-specifikációkat elemezve, a másolás végrehajtásához rendszerhívásokat kezdeményez az **operációs rendszer** felé, melynek feladata a gép erőforrásainak kezelése, elosztása és egységes programozói felület biztosítása.

Miután az operációs rendszer "tudja" hogy a merevlemez milyen logikai drive-okra, illetve partíciókra van bontva (a file-rendszer kezelése az egyik legfontosabb feladata), és ezeken belül hol van szabad hely, illetve hol helyezkedik el a forrásként megadott file. Régen **BIOS** (Basic Input Output System) hívások segítségével már rendelkezhett a másolás végrehajtásáról. Manapság ezt valamely rendszermag (**kernel**), vagy virtuális gép (**VM**, pl).végződteti.

A kernel szintű meghajtóknak (**driver**) nincs más feladata, mint az operációs rendszertől paraméterként kapott track (egy sorszámozott adathordozó sáv a merevlemez felületén), és szektor (a sávon belüli, sorszámozott és adott méretű adatblokk) információkkal összhangban vezérli a hard disk fejpozícionálási, és adatátviteli (olvasás, írás) tevékenységét. Ez a HD (hard disk) controller-nek, mint perifériának szóló I/O utasítások sorozataként realizálódik.

Szigorúan véve a dolognak itt még nincs vége, hiszen napjainkban egyre gyakoribbak a rendkívül intelligens perifériák. Tipikus, hogy a rendszert mintegy "becsapva", a winchester saját, sebességre optimalizált szektor transzlációval, és belső gyorsító tárral (cache) rendelkezik. Ezeket a funkciókat a winchester elektronikában elhelyezkedő mikrokontroller beégetett kódja (**Firmware**) vezérli.

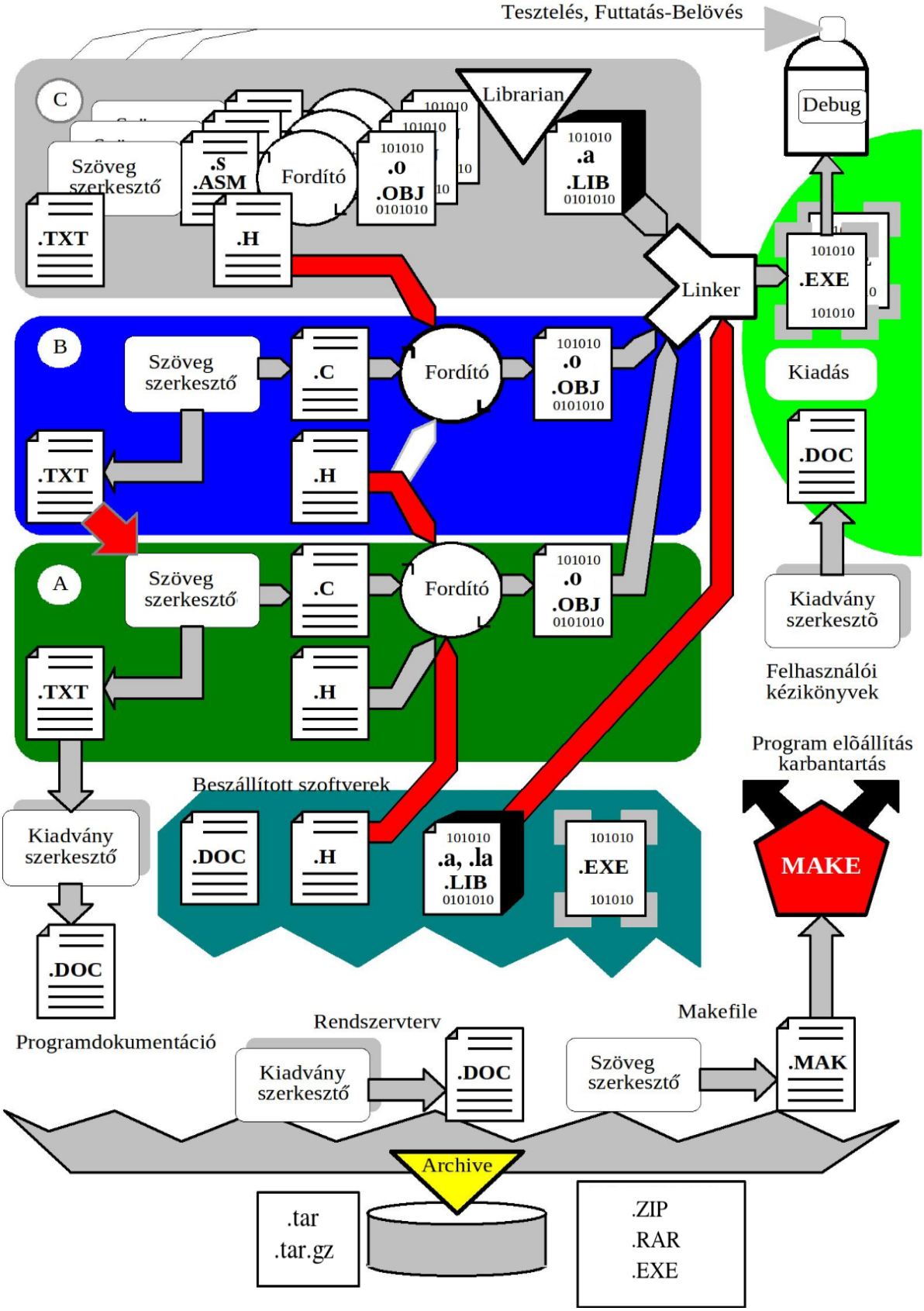
Fenti modell rétegei persze újabb, részletekbe menő hierarchia szintekre bonthatók, mint például az operációs rendszer vezérlő és kiszolgáló szintjei.

A moduláris programozástechnika biztosítja, hogy az egyes hierarchiaszintek logikailag egybekapcsolható tevékenységeit megvalósító kódokat, illetve a hozzájuk tartozó adatszerkezeteket különálló programmodulokba szervezve, áttekinthető és jól nyomon követhető rendszert kapjunk.

A módszert alkalmazva az egyes részfeladatok megoldása az ahhoz legjobban értő programozóra bízható. Irreális követelmény lenne hogy a programozótól elvárjuk, ugyanolyan mély ismeretekkel rendelkezzen a soros vonali kommunikációs eszközök programozásáról mint az absztrakt adatszerkezetek alkalmazásáról egy magasszintű kommunikációs protokoll létrehozásánál.

Egy programmodul kialakítása már lehet testreszabott, egyéni feladat. A rendszerprogramozóval, vagy a környező hierarchiaszintek programozóival egyeztetett kapcsolódási felületek, mint peremfeltétel mellett a programozó a saját alapobjektumára koncentrálva létrehozhatja saját adatszerkezeteit

és megírhatja az objektumok (pl. egy egyszerű soros vonali periféria illesztő) kezelését végző eljárásokat.


Tesztelés, Futtatás-Belövés
C
Szöveg szerkesztő
.S
.ASM
Fordító
101010
.0
.OBJ
0101010
Librarian
101010
.a
.LIB
0101010
.TXT
.H
Debug
Linker
101010
.EXE
101010
B
Szöveg szerkesztő
.C
Fordító
101010
.0
.OBJ
0101010
.TXT
.H
Kiadás
.DOC
A
Szöveg szerkesztő
.C
Fordító
101010
.0
.OBJ
0101010
.TXT
.H
Kiadvány szerkesztõ
Felhasználói kézikönyvek
Program elõállítás karbantartás
Beszállított szoftverek
Kiadvány szerkesztő
.DOC
.H
101010
.a, .la
.LIB
0101010
101010
.EXE
101010
MAKE
.DOC
Programdokumentáció
Rendszervterv
Kiadvány szerkesztő
.DOC
Makefile
Szöveg szerkesztő
.MAK
Archive
.tar
.tar.gz
.ZIP
.RAR
.EXE

Az előző oldal ábrája egy lehetséges technológiai láncra ad példát, amely támogatja a hierarchikus és moduláris programozástechnikát.

Példánkban három szintet különböztetünk meg, az "A" programozó magát az alkalmazást (keretprogram, **main** függvény) programozza a beszállított, és az alsóbb szintek programozói által létrehozott modulokra támaszkodva. A "B" csoport (vagy programozó) egy közbenső szinten a kiszolgálói, és vezérlési feladatkör összefogásáért felelős, míg a "C" csoport az alacsony szintű, eszközközeli eljárások gyűjteményét hozza létre.

A "C" csoport több programozói **szövegszerkesztő** program segítségével létrehozzák az assembly forráskódot (**.ASM**), a modul eljárásainak és adatszerkezeteinek pontos dokumentációját (**.TXT**), és a magasabb hierarchiaszinten dolgozó kollégáik (A, B) számára a C nyelvű illesztéshez szükséges header (**.H**) file-okat. Ez utóbbi tartalmazza a csomag eljárásainak (függvényeinek) prototípusait, az ott alkalmazott adatszerkezetek leírását (typedef), konstans definíciókat (#define direktíva), illetve elérhetővé teszik az esetleges globális adatterületeket és változókat (extern direktíva). A "C" csoporton belül, a modulok között az illeszkedés (az ábrán nincs feltüntetve) az assembly nyelvnél megszokott include file-okon (**.INC**) keresztül történik.

A fordítóprogram (itt assembler) segítségével létrehozzák a tárgykódú (object, **.OBJ**) modulokat, melyeknek formátumát valamely szabvány (pl. ELF – Executable Linkable Format, COFF - Common Object File Format) határozza meg. Az ebben szereplő objektumok leírási módja már független az őket forrásban leíró programozási nyelvtől, így a technológiai lánc támogatja a kevert nyelvű programozást is. Tartalmazzák a modul által közzétett (public) objektumok nevét, a rájuk vonatkozó elhelyezési információkat (milyen relatív címen, mely programszekcióba kerüljenek, pl. .text=kód, .data=adat, stb..), az inicializált adatterület adatait, és a relokálatlan, processzornak szóló bináris kódokat.

A **library manager** (librarian, könyvtárszerkesztő) segítségével modulkönyvtárba (**.LIB**) szervezik a csoport által létrehozott object file-okat, ami elősegíti a könnyebb kezelést, szállítást, arhíválást és publikálást. Nem véletlen hogy a Linux rendszer esetén az **ar** (archive) program segítségével hozhatunk létre ilyen könyvtárakat. Mint látni fogjuk, a C programozási nyelv fejlesztő környezetéhez hozzá tartoznak az ú.n. standard C library-k, melyek a szabványos C alapeljárásait tartalmazzák. Gyakori, hogy az egyes **beszállított programcsomagok** (megvásárolt és felhasznált) esetén is library és header file-ok formájában (pl. MS-Windows SDK) kapjuk meg a programozói felületet (**API**, Application Programming Interface).

A "B" csoport feladata összetett, hiszen egyrészt alkalmaznia kell az alsóbb szint eljáráskönyvtárát, másrészt neki is létre kell hozni egy, a főprogram alapobjektumaira épülő programozói felületet a rendszerprogramozó ("A") számára. Eszközei csak a fordítóprogram esetén térnek el a "C" csoportétól. Itt - példaként - magas szintű C programozási nyelvet alkalmazunk (lehet más, PASCAL, FORTRAN, PL-1, COBOL, stb... is, de a villamosmérnöki gyakorlatban a legelterjedtebb nyelv a C illetve assembly), és ennek megfelelően az illeszkedés a környező szintekhez header file-ok segítségével történik. A gyakorlatban ezen a szinten a legnépesebb a programozói tábor egy projekten belül, és itt jöhet létre a legtöbb programmodul.

Az "A" csoport többnyire egyetlen személyt, a rendszerprogramozót jelenti, aki az egész projektet összefogja, bár rendes körülmények között a főprogram programozójának, és a rendszerprogramozónak a személye szétválhat.

A **rendszertervet** egy alkalmasan választott kiadványszerkesztő program (szerényebb követelmények esetén szövegszerkesztő) segítségével lehet összeállítani (**.DOC**). Ugyancsak ezzel készül a programdokumentáció, amely a programozóktól begyűjtött (**.TXT**) modul dokumentációkra támaszkodik és a felhasználói leírás törzspéldánya. Természetesen ez utóbbit nyomdai eljárással célszerű végleges formába önteni illetve sokszorosítani.

Ügyesebb megoldás a W3 konzorcium által rögzített, időtállónak tűnő specifikáció alapján előállított **.HTML** alapú dokumentáció, amiből megfelelő eszközökkel sokféle dokumentum típus állítható elő (RTF, ODT, PDF)

A projekt anyagait célszerű periodikusan, esetleg a verziószám változása során **archiválni**. (A munkamentések egy jól szervezett hálózati környezetben a rendszergazda által szabályozott módon mindig megtörténnek) Erre a célra minden operációs rendszer nyújt egy alapszolgáltatást, (pl. DOS esetén backup, Linux-nál tar és compress) de alkalmazhatók egyéb gyártótól származó eszközök is (arj, pkzip, gzip).

Azt, hogy a modulrendszerben minden komponens **naprakész** legyen, (az egyes modulok is és a technológiai lánc elemei is függnek egymástól) a **MAKE** (program maintenance utility) biztosítja, az őt vezérlő Makefile felhasználásával (ebben vannak leírva a technológiai lánc eljárásai és modulhierarchia elemeinek függőségi listái).

Végül nézzük magát a végterméket, annak be- vagy letöltését, **futtatását**, belövését, illetve **tesztelését**. Ahogy a bevezetőnkben említésre került, a programok megjelenési formái igen változatosak lehetnek, annak a függvényében hogy mit programoztunk.

Az egyes modulokat (.OBJ, és .LIB) a **linker** fűzi össze olyan programmá, amely alkalmas a betöltésre és futtatásra (itt pl. **.EXE**).

Speciális eset amikor egy objektum nincsen sztatikusan hozzáadva a végtermékhez, hanem csak hivatkozunk rá – **dinamikusan** linkeljük. Az ilyen ”Shared Object” (**.so**) vagy ”Dynamic Link Library” (**.DLL**) eleve rendelkezésre kell hogy álljon a célplatfromon – méghozzá olyan verzióban, amellyel a szoftverünk készült. Ez időnként kínos meglepetésekkel szolgál...

Egy "mosógép" processzor (micro controller) programját töltés helyett magába a chip-be, vagy a környékén elhelyezett PROM-ba (Programmable, Read-only Memory) kell beégetni. A linker kimenete ekkor olyan formátumú kell hogy legyen, amit az "égető megeszik", pl. Intel-hex, JEDEC, bináris, stb... A hibakeresés itt oszcilloszkóp, **szimulátor**, vagy ú.n. **in-circuit emulátor** (a chip helyébe illesztett hibakereső rendszer) segítségével történhet.

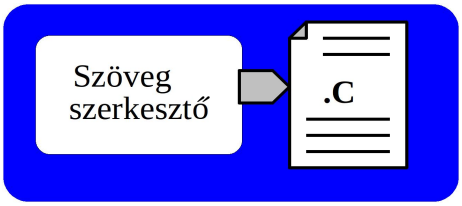
Egy számítógép felhasználói programját az operációs rendszer tölti be az operatív tárba, így a linker kimenete egy, az operációs rendszer által elfogadott formátumú, azaz végrehajtható (**executable**, Windows esetén .EXE, .COM, VMS esetén .EXE, Linux-nál a file-mód x) file kell hogy legyen. A hibakeresés a fejlesztői környezethez mellékelt **szimbolikus debugger** segítségével történhet, mely lehetővé teszi a forrássoronkénti léptetést, adatszerkezetek vizsgálatát, töréspontok és vizsgálati helyek megadását, stb...

A fenti eszközöket a hibásan működő vagy működésképtelen program hibáinak feltárására kell bevetni. Azt hogy a program megfelel-e a specifikációnak avagy sem, különálló **konformancia** vagy **interop tesztek** során kell eldönteni.

Ugyancsak célszerű az egyes modulok programozóinak saját **teszt programmal** meggyőződni a modul működőképességéről. Különösen igaz ez akkor, ha a végtermék nem program, hanem **eljáráskönyvtár**, **eszközmeghajtó** (device driver), futásidejű osztott eljáráskönyvtár (pl. **runtime library**, vagy **Windows DLL**).

Segédletünk nem tárgyalja a különféle integrált fejlesztői környezeteket (IDE), illetve konkurens fejlesztői és verziószámozó rendszereket (CVS,SVN, stb. .), mivel ezeket időről időre lecserélik valami újabbra (e dokumentum második kiadásának szerkesztésekor az Eclipse, illetve a csoportos online munka automatizálására a git/github, illetve Jenkins a ”menő”).

Szövegszerkesztők és forráskód

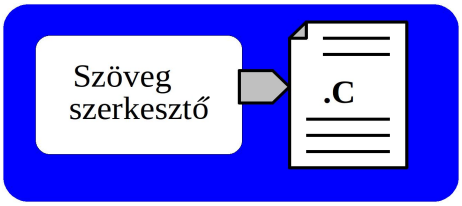


A forráskód előállításához célszerű olyan szövegszerkesztőt használni, amely rendelkezik a programfejlesztés szempontjából nélkülözhetetlen alapszolgáltatásokkal. Habár az integrált fejlesztői környezetek többnyire tartalmaznak beépített szövegszerkesztőt, indokolt lehet az olyan eszközök használata, melyek támogatják az igen nagy méretű file-ok kezelését, tartalmaznak billentyűzet-makrózást, és minimális táblázatkezelő funkciókat. Ezeket két alkalmasan választott eszköz, a ”The Hessling Editor” (**THE**), és a **vi/vim** (Linux) referenciakártyáin (következő oldalak) keresztül mutatjuk be.

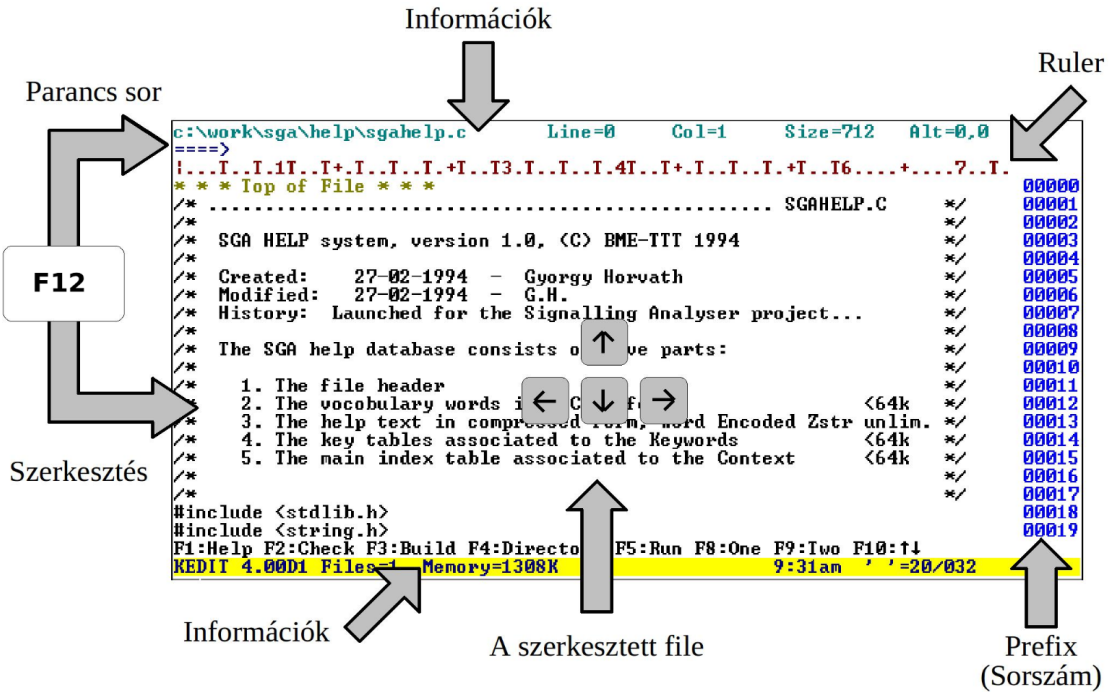
Természetesen a forráskód kialakítása az adott programozási nyelv szintaxisát kell hogy kövesse. Ezen túlmenően azonban célszerű bizonyos formai és tartalmi követelményeket is kialakítani a csoportmunka során:

- Minden file elején megjegyzésként szerepeljen a file vagy modul neve, verziószáma, a programozó neve, a copyright, a modul használatának rövid leírása, az esetleges feltételes fordításra vonatkozó kulcsszavak, a létrehozás dátuma, a módosítások dátuma, jellege, programozója, és végül a file végén újból fel kell tüntetni a modul nevét. A programozás, így a kommentezés nyelve is az **angol**.
- A **forrás tagolása** során célszerű tartani az alábbi sorrendet: - include v. header file-ok befűzése (include)- globális (public) változók deklarációja - inicializált globális változók deklarációja - belső szimbolikus konstansok definíciói - belső adatszerkezet típusdefiníciói - belső változók deklarációi - inicializált belső változók deklarációi - belső makro definíciók - belső függvények, eljárások - globális függvények, eljárások
- Header, vagy include file-ban zárjuk ki az újra-belépést (#ifndef mymodule, #define mymodule, ... #endif), és csak akkor generáljunk benne kódot (függvény deklaráció, inicializált adat, stb...) ha tényleg muszáj. Ide célszerű csak include-okat, szimbolikus konstans definíciókat, típus definíciókat (adatszerkezet), makrókat, külső hivatkozásokat (extern int ...), és függvény prototípusokat tenni.
- Lehetőleg egy szerkesztett sor csak egy utasítássort tartalmazzon, de a **sor** ne legyen hosszabb **72** karakternél, használjunk inkább folytatósort. (A régi FORTRAN nyelv esetén ennél sokkal szigorúbbak a követelmények.) Ez alól csak az az eset kivétel, ha több utasítás is logikailag egybefogható, és el is férnek egy sorban.
- A típusok, konstansok, függvények, és változók nevénél a project direktíváit kell követni (pl. az MS-Windows esetén az ú.n. Hungarian Notation-t használják), de programunkat próbáljuk meg olvashatóvá tenni a funkcióra utaló angol nyelvű szavak felhasználásával.
- Minden **lényeges** részletet lássunk el megjegyzésekkel. A függvények esetén írjuk le az alapfunkciót, a paraméterezést, és a globális erőforrásokkal kapcsolatos hatásait. Az esetleges feltételes fordítások IF-

ENDIF párjait, és a többszörösen beágyazott ciklusok utasítás zárójeleit párosítsuk a mellékjük helyezett megjegyzésekkel.



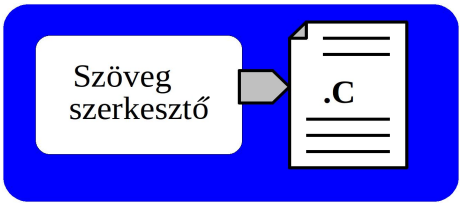
THE referenciakártya

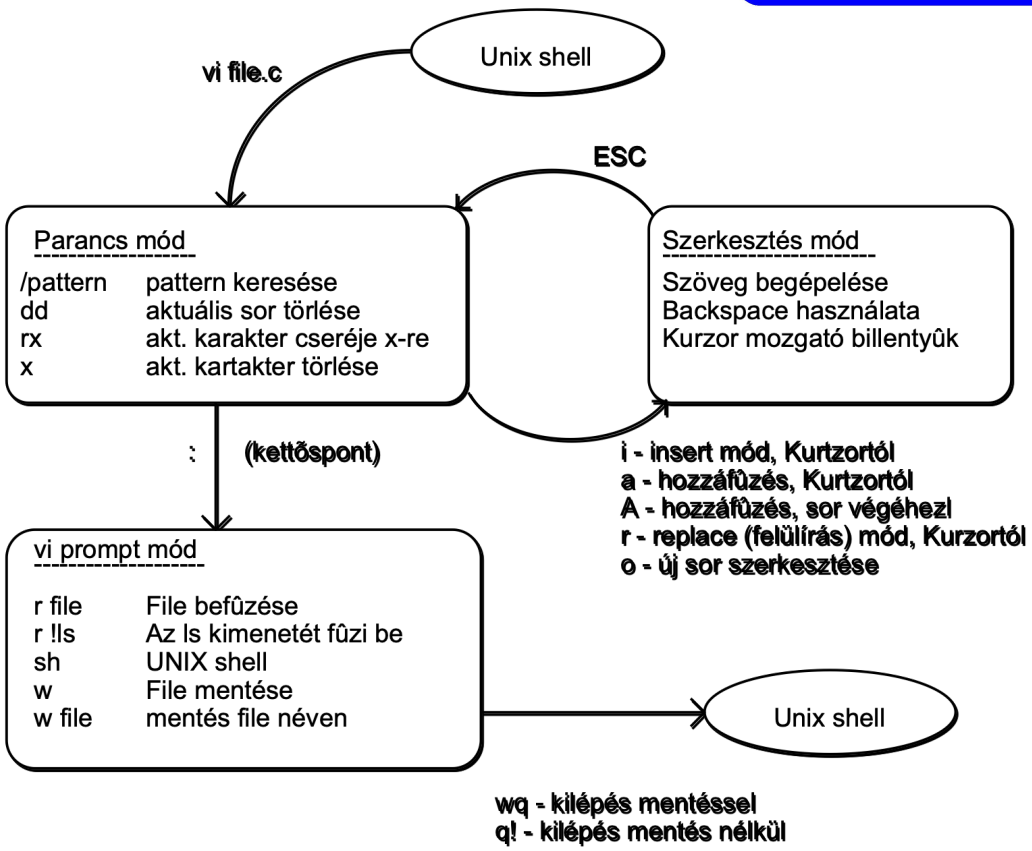


Indítás	**the** <filespec>
Szöveg bevitel, input mód	===>**i**
Szerkesztés	Szöveg begépelése, **cursor** mozgatók
Beszúrás/felülírás	**Ins**
Új sor beszúrása	cursor a beszúrás elé, **CTRL+A**
Sor törlése	cursor a törlendő soron, **CTRL+D**
Sor kijelölése	kijelölendő soron, **CTRL+L**
Sor másolása, kijelölés feloldása	beszúrás elé **CTRL+C**
Sor másolása, kijelölés marad	beszúrás elé, **CTRL+K**
Több sor kijelölése	első sor, és utolsó soron **CTRL+L**
Blokk kijelölése	Blokk sarkainál **CTRL+B**
Blokk másolása, felülírás (lásd még sorok másolása, ALT+C)	cursor a célterület bal felső sarkára **CTRL+W**
Kijelölés feloldása	**CTRL+U**
File befűzése	===>**get** <filespec>
Mintázat keresése	===>/a keresett mintázat/
Mintázat cseréje n sorra, soronként m előfordulást (összes előfordulás: *)	===>**c**/src/trg/**n m** például **c**/printf(/fprinf(outfile,/* *
File mentése	===>**ff**
File mentése más néven	===>**ss** <newfilespec>
Két ablak nyitása	===>**scr 2**
Váltás az ablakok között	**F10**

File váltása az ablakban újabb file szerkesztése	===>**x** **===>x** <plusfilespec>
Kilépés	===>**qq**

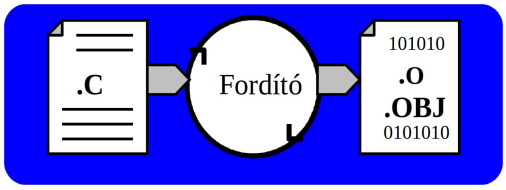
vi referenciakártya





gyakoribb, haladó szintű műveletek összefoglalása

Művelet	THE =>	vi, vim :
Szöveg keresése	`/minta/`	`/minta`
Kis/NAGY betű érdektelen	`set case mixed ignore`	`set ic`
Kis/NAGY betű érdekes	`set case mixed respect`	`set noic`
Csere innen max 5 soron keresztül	`c/minta/mire/5`	`s/minta/mire/5`
Csere innen az összeset	`c/minta/mire/*`	`.,$s/minta/mire/g`
Rendezés a 19.-id karakter szerint	`sort * 19 27`	`sort /.*\%19v/`
Mentés formátuma DOS	`set EOLOUT CRLF`	`set ff dos`
Linux fileformátum	`set EOLOUT LF`	`set ff Linux`
Csak a keresett sorokat mutassa	`all/minta/`	`g/minta/`
Osztott ablakok, vízszintes osztás	`scr 2`	`sp`
Függőlegesen osztott ablakok	`scr 2 v`	`vsp`
Mozgás az ablakok között	`F10`	**`CTRL+w w`**



Fordítóprogramok és tárgykód

A fordítóprogram feladata a forráskód **előfeldolgozása** (preprocesszor, a makrók, include-ok kifejtése), **szintaktikai elemzése** (a hibák kijelzése mellett), és a **fordítás** során a linker által elfogadott tárgykódú modul (**object**) előállítása.

Segédszolgáltatásként formázott fordítási, assembly lista és keresztreferencia kérhető, melyek többek között összefoglalva tálalják a modul adat- és vezérlési szerkezetére vonatkozó információkat. A kész object vizsgálatára is léteznek szoftver eszközök, melyek a file rekordjait jeleníti meg többé-kevésbé emberileg emészthető formában.

A legtöbb fordító egyszerre több forrást is képes fordítani, és külön kapcsoló hiányában, rögtön megpróbálja előállítani a végrehajtható kódot is a linker automatikus hívásával.

A segédlet alábbi táblázatai néhány, az interneten szabadon elérhető fordító használatát és legfontosabb opcióit mutatja be.

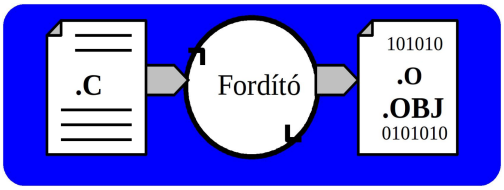
Host/CPU	Cél OS/CPU	Fordító	Használata
/	*/*	GNU C compiler	**gcc** [opciók] file1 file2 ....
Lin,Win/*	Win32/x86	MINGW (GNU C)	**gcc** [opciók] file1 file2 ....
*/x86	*/x86	Watcom C	**owcc** [opciók] file1 file2 ....
/	/PIC,Z80.8051	sdcc	**sdcc** [opciók] file1 file2 ...
Win/x86	/PIC	mcc18	**mcc18** [opciók] file1 file2 ...
/	*/*	JAVA compiler	**javac** [opc.] src.java

A fontosabb fordítási opciók:

Opció	gcc	sdcc	owcc	mcc18	javac
debug info	**-g**	**--debug**	**-g2**		**-g**
out file	**-o** xxx.o	**-o** xxx.o	**-o** xxx.o	**-fo** xxx.o	**-d** path
Warning off	**-W**	**--disable-warning**	**-Wlevel** 0	**-W=1**	**-nowarn**
Csak fordít	**-c**	**-c**	**-c**	(csak ford.)	
definíció	**-D**name	**-D**name	**-D**name=xx	**-D**name=xx	
include dir.	**-I**dir	**-I**directory	**-I** dir	**-I** dir	**-cp** classpath
listafile	**-E**	(mindig)		(mp2cod)	
asm lista	**-S**	(mindig)		(mp2cod)	

A fenti táblázat korántsem teljes, az alkalmazható kapcsolók köre is erősen függ a fordító gyártójától, az operációs rendszertől, és a cél processzor típusától. Ha a célrendszer egy öreg DOS-os PC, a Watcom C külön kapcsolókészlete szabályozza a memória modellt, amely a processzor sajátos címzési módszeréből következik.

A kapcsolókról részletesebb listát kaphatunk vagy úgy hogy a fordítót paraméterek nélkül indítjuk, vagy az online felhasználói kézikönyvekből (Linux esetén pl. a **man cc** parancs használható).

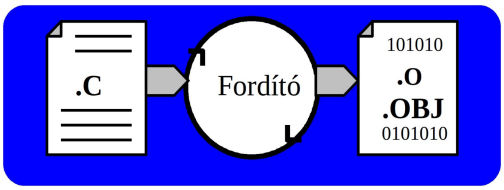


Példa:

Még egy olyan egyszerű feladat megoldása - mint a "Hello" üzenet kiíratása - is tanulságos folyamat. Az előző fejezetben leírt, vagy az általunk megszokott szövegszerkesztővel létrehoztunk egy - a feladatot a „C” tankönyvekben leírtak szerint megoldani kívánt - STDHELLO.C forrásfile-t:

```
/*.............................................. STDHELLO.C       */
/*..                                                              */
/*..   An example to print a Hello message to the standard output  */
/*..   ========================================================   */
/*..                                                              */
/*..   (C) Technical University of Budapest                       */
/*..       Department of Telecommunication and Telematics         */
/*..       Telecommunications Laboratory                          */
/*..                                                              */
/*  Created:    26-09-2013  -  Gyorgy Horvath                     */
/*  Modified:   26-09-2013  -  Gyorgy Horvath                     */
/*..                                                              */
/*  History:    Programming Practice for Module Laboratory        */
/*              Demonstrate how to compile and link a             */
/*              simple program and show all the components        */
/*              of the created EXE file                           */
/*                                                                */
/*  Header file that we need to use printf                        */
/*                                                                */
#include <stdio.h>
/*                                                                */
/*  Our static (used by us only) data                             */
/*                                                                */
static char *MsgHello = "\nHello\n";
/*                                                                */
/*  The only function constructed is the main() as follows:       */
/*                                                                */
/*  Print "Hello" to the standard output                          */
/*  ----------------------------------                            */
/*  input:   none        (ignore argc, and argv parameters )      */
/*  return:  none        (don't care the return value)            */
/*  affects: the standard output                                  */
/*  calls:   printf      in stdio                                 */
/*  notes:   Avoid the printf("\nHello\n") style programming      */
/*           since there may be several hello's in our module.    */
/*           Use static data, or a separate global message module */
/*           containing the program messages.                     */
/*                                                                */
/*          =====                                                 */
void        main( void )
/*          =====                                                 */
{
printf( MsgHello );
}
/*..                                                              */
/*...................................... END OF STDHELLO.C       */
```

Az hogy a fordítóprogram környezetének beállítása korrekt-e vagy sem, az első fordítási kísérletnél kiderül. Ha a fordítónk nem indul (a fordítási parancs hatása hibaüzenet az operációs rendszertől), vagy hiányolja a #include <> direktíva által specifikált header file-okat, többnyire magunknak kell gondoskodni a környezet helyes beállításáról.



A technológiai láncnak ezen a poi
beállítása lehet kritikus:

PATH	A keresési út programindításhoz
TMP vagy **TEMP**,	A fordító átmeneti munkaterületét adja meg
INCLUDE	A header file-ok elérési útvonala, annak az operációs rendszernek a szintaxisával, amelyen a fejlesztőeszközünk fut (nem a célrendszer)
LIB	Eljáráskönyvtárak elérési útvonala

Gyakori megoldás hogy az eszközkészlet gyártója által biztosított külön batch file, vagy shell script indításával gondoskodhatunk a megfelelő beállításokról.

Álljon itt példaként az Open Watcom két állománya – Win32, és Linux hoszt részére:

```
C:\WATCOM>type owsetenv.bat
@echo off
echo Open Watcom Build Environment
SET PATH=C:\WATCOM\BINW;%PATH%
SET PATH=C:\WATCOM\BINNT;%PATH%
SET INCLUDE=C:\WATCOM\H\NT;%INCLUDE%
SET INCLUDE=C:\WATCOM\H\NT;%INCLUDE%
SET INCLUDE=%INCLUDE%;C:\WATCOM\H\NT\DIRECTX
SET INCLUDE=%INCLUDE%;C:\WATCOM\H\NT\DDK
SET INCLUDE=C:\WATCOM\H;%INCLUDE%
SET WATCOM=C:\WATCOM
SET EDPATH=C:\WATCOM\EDDAT
SET WHTMLHELP=C:\WATCOM\BINNT\HELP
SET WIPFC=C:\WATCOM\WIPFC

C:\WATCOM>
```

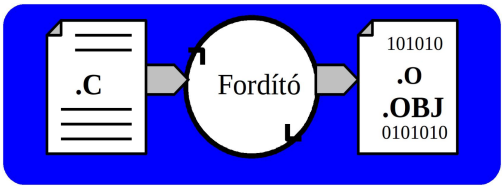
illetve

```
yoda Watcom # cat owsetenv.sh
#!/bin/sh
echo Open Watcom Build Environment
export PATH=/opt/Watcom/binl:$PATH
export INCLUDE=/opt/Watcom/lh:$INCLUDE
export EDPATH=/opt/Watcom/eddat
export WIPFC=/opt/Watcom/wipfc
export WATCOM=/opt/Watcom
yoda Watcom #
```

Ha feltételezett beállítások jónak tűnnek, pl. Linux alatt a GNU C fordító segítségével előállíthatjuk a mintaprogramhoz szükséges object file-t az alábbiak szerint:

```
meres5@alpha:~/tmp$ gcc -c stdhello.c
meres5@alpha:~/tmp$ ls -la stdhello*
-rw-r--r-- 1 meres5 meres 2945 Apr  8 15:28 stdhello.c
-rw-r--r-- 1 meres5 meres 1080 Apr  8 15:29 stdhello.o
meres5@alpha:~/tmp$
```

Az újonnan keletkezett stdhello.o ta
vizsgálva azonosíthatjuk a forrásban l



```
meres5@alpha:~/tmp$ objdump -hs stdhello.o

stdhello.o:     file format elf32-powerpc

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000044  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000004  00000000  00000000  00000078  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, DATA
  2 .bss          00000000  00000000  00000000  0000007c  2**0
                  ALLOC
  3 .rodata       00000008  00000000  00000000  0000007c  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .comment      0000001d  00000000  00000000  00000084  2**0
                  CONTENTS, READONLY
  5 .note.GNU-stack 00000000  00000000  00000000  000000a1  2**0
                  CONTENTS, READONLY
  6 .gnu.attributes 00000014  00000000  00000000  000000a1  2**0
                  CONTENTS, READONLY
Contents of section .text:
 0000 9421fff0 7c0802a6 90010014 93e1000c  .!..|...........
 0010 7c3f0b78 3c000000 7c090378 80090000  |?.x<...|..x....
 0020 7c030378 4cc63182 48000001 397f0010  |..xL.1.H...9...
 0030 800b0004 7c0803a6 83ebfffc 7d615b78  ....|.......}a[x
 0040 4e800020                             N..
Contents of section .data:
 0000 00000000                             ....
Contents of section .rodata:
 0000 0a48656c 6c6f0a00                    .Hello..
Contents of section .comment:
 0000 00474343 3a202844 65626961 6e20342e  .GCC: (Debian 4.
 0010 342e352d 38292034 2e342e35 00        4.5-8) 4.4.5.
Contents of section .gnu.attributes:
 0000 41000000 13676e75 00010000 000b0401  A....gnu........
 0010 08010c02                             ....
```

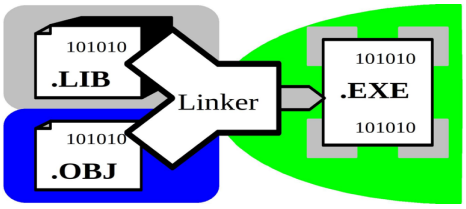
Ez a hasznos eszköz fellelhető több gyártó eszközkészletében is – persze lehet hogy más más néven, pl **tdump**, **dmpobj**, stb. . .

Sokféle szempont szerint jeleníthető meg a tartalom – akár assembly lista is kérhető a kódról.

Az egyes szekciók neve változhat a platformtól, és eszközkészlettől függően, de az alábbi táblázat segít az eligazodásban:

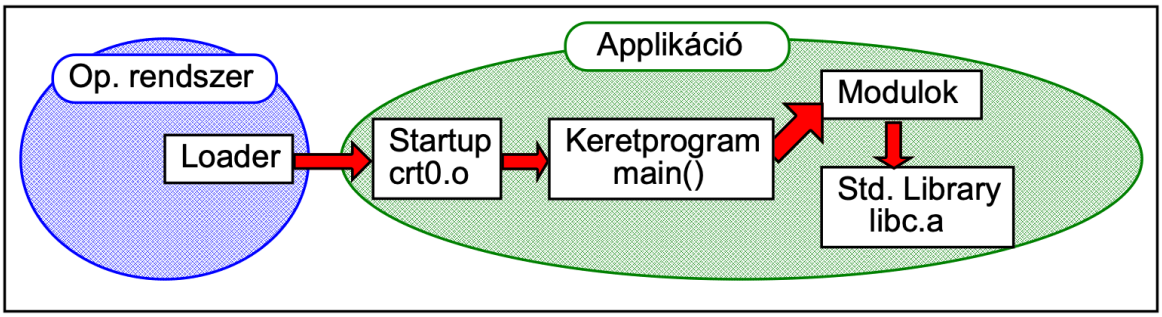
Szekció	Leírás
.text, .code	program kód.
.data	írható, olvasható adatterület – lehet inicializálva
.bss	nem inicializált adatterület
.rodata, .const, .rdata, .init	csak olvasható, inicializált adatterület
.vector	Interrupt vector table
.stack	Stack

A linker



A linker feladata a tárgykódú modulok és könyvtárak összefűzése a célrendszer által betölthető (DOS/Windows esetén EXE file-ok, Linux esetén ELF (Executable and Linkable Format) formátumú, vagy az abszolút módon elhelyezett objektumok esetén közvetlenül égethető/futtatható (pl. Intel-hex formátumú file) kóddá oly módon, hogy az egyes modulok objektumai a megfelelő programszekciókba kerüljenek, és a kereszthivatkozások a megfelelő elhelyezési információk (relocation) birtokában feloldódjanak. A végtermék persze lehet akár speciális formájú, dinamikusan linkelhető (DLL, Dynamically Linked Library, vagy Shared Object) futásidejű eljáráskönyvtár is.

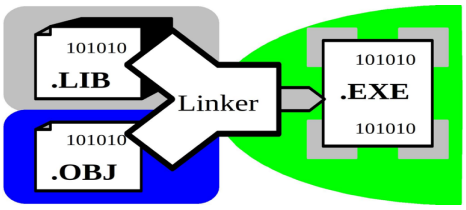
Ahhoz hogy megértsük milyen komponenseket kell összefűzni egy operációs rendszer alatt futtatható program előállításánál, célszerű áttekinteni egy tipikus vezérlés-átadási hierarchiát az alábbi ábra segítségével:



Az operációs rendszer a file töltése és esetleges relokálása után az ú.n. Startup code belépési (**__start** szimbólum) pontjára adja át a vezérlést. Ez többnyire a fejlesztő eszköz valamely LIB könyvtárában elhelyezkedő **crt*** (crtnull) nevű tárgykódú (Object, .o, .obj) modulban van, amely C nyelvű keretprogram esetén a program indításakor megadott parancssorbeli paraméterek átadása mellet meghívja az általunk írt **main** függvényt.

A keretprogram, illetve az egyes programmodulok többnyire használják a nyelvi környezet standard szolgáltatásait, amelyek a fejlesztő rendszerrel szállított tárgykódú eljáráskönyvtár (**libc**) befűzésével érhetők el.

Multitask operációs rendszer esetén a linkelés során beállítható, hogy ez utóbbit kódként hozzáfűzve statikus (**static** library) módon kívánjuk elhelyezni a file-ba, vagy az operációs rendszer osztott eljáráskönyvtárát (**shared** object library) használjuk. A dinamikus vagy osztott objektumok előnye hogy a kód csak egy példányban van jelen, és az exe file mérete is kicsi lesz; hátránya hogy a programot más rendszerbe beszállítva bizonyos objektumok hiányozhatnak a futáshoz. A statikusan linkelt program mérete igen nagy lehet viszont "önjáró".



Az alábbi táblázatok a fordítóknál bemutatott fejlesztőrendszerek linkereinek használatát, a standard elemek lelőhelyét, illetve a fontosabb kapcsolókat ismertetik.

Az egyes linkerek indítása:

Host/CPU	Cél OS/CPU	Linker	Használata
/	*/*	GNU C compiler	**ld** [opciók] obj1 obj2 **-l**lib1 ....
Lin,Win/*	Win32/x86	MINGW (GNU C)	**ld** [opciók] obj1 obj2 **-l**lib1 ....
*/x86	*/x86	Watcom C	**wlink** direktívák v. @lnkfile
/	/PIC,Z80.8051	sdcc	**aslink** -f lnkfile [opciók] ...
Win/x86	/PIC	mcc18	**mplink** [opciók] lnkfile obj lib...
/	*/*	JAVA archive*	**jar** [opciók] jarfile [classfiles]..

A korszerű eszközök esetén a linker indítása a fordítás után automatikusan történik. A fenti táblázatban feltüntettük azt a megoldást, amikor az eszközt csak linkelésre használjuk. Az előbbi megoldás persze előnyösnek látszik, hiszen a linker paraméterezése - beleértve a standard objektumok megadását is - automatikusan megtörténik. Ekkor viszont fogalmunk sincs róla miből is áll össze a végtermék.

A gyakrabban használt linker opciók:

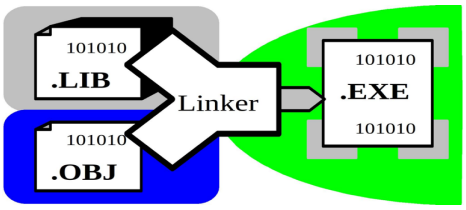
Opciók	ld (GNU)	aslink (sdcc)	Wlink (Watcom)	mplink (Microchip)
karcsúsítás	**-s**		(wstrip)	
out file	**-o** xxx	**-i** (intelhex)	**N** xxx.o	**/o** file
map file	**-M**	**-m**	**M**	**/m** file
lib path	**-L** path	**-k** path	**LIBP** path	**/l** path
lib valami	**-l**valami	**-l** name.lib	**L** file.lib	(control)
control file	@ file	**-f** file	@ file	(arg1)

Példa:

Visszatérve a "Hello" üzenet kiíratására, most már csak össze kell fűznünk az előző fejezet példáinál előállított tárgykódú állományt a standard objektumokkal, és már kész is a futtatható program. Azaz erősen leegyszerűsítve:

Futtatható_Programfile = Startup_module + STDHELLO + Standard_Library

A technológiai lépés az egyik választott fejlesztőeszköz esetén az alábbiak szerint alakulhat:



LD A Linux vagy UNIX szerű operációs rendszerek linkereinek kezelése meglehetősen egységes. A következő példát egy IBM RS6000 típusú gépre telepített Linux operáció rendszere alatt dolgoztuk ki. Előállítjuk az STDHELLO program kétféle verzióját. Az első az osztott eljáráskönyvtáron alapul (default), míg a második statikusan magába foglalja a futáshoz szükséges valamennyi alapmodult, le egészen a rendszerhívásokig.

Ha a GNU fordítót a legegyszerűbb módon (gcc stdhello.c -o stdhello) alkalmazzuk a programunk előállítására, valami ilyesmi zajlik a háttérben a tárgykódú modulok összefűzésekor:

```
ttmer99@ttmer99:~$ ld -s -dynamic-linker /lib/ld-linux.so.2 \
      /usr/lib/i386-linux-gnu/crt1.o /usr/lib/i386-linux-gnu/crti.o \
      stdhello.o -lc /usr/lib/i386-linux-gnu/crtn.o \
      -o stdhello_dynamic

ttmer99@ttmer99:~$ ./stdhello_dynamic

Hello
ttmer99@ttmer99:~$ ls -l stdhello_dynamic
-rwxr-xr-x 1 ttmer99 ttmer99 2688 Apr 12 09:11 stdhello_dynamic
```

Látható hogy a régi idők modelljéhez képest itt nem csak a startup code (immár crt1) hanem a dinamikus linker, prológus, és kód epilógus is befűzésre kerül a tárgykódú modulunk, és a C library importja mellé. Miután csak hivatkozások vannak a szabvány C függvényekre, a keletkezett program elég kicsi lesz.

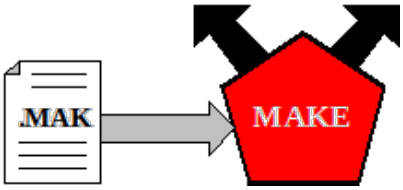
Az ”önjáró” verziót a MUSL (http://www.musl-libc.org/) projekt beágyazott rendszerek számára kidolgozott C könyvtárával, és a fodítói könyvtár rutinjaival fűzzük össze:

```
ttmer99@ttmer99:~$ ld crt1.o stdhello.o -L. -lc -lgcc -o stdhello_static
ttmer99@ttmer99:~$
ttmer99@ttmer99:~$ $ ./stdhello_static

Hello
ttmer99@ttmer99:~$ ls -l stdhello_static
-rwxr-xr-x 1 ttmer99 ttmer99 25402 Apr 15 10:08 stdhello_static
```

Ez utóbbi végtermék mintegy tízszer akkora mint az előző verzió, viszont nem igényel jól kiépített futtatási környezetet.

Érdemes megfigyelni hogy a -l kapcsoló után a könyvtár rövidített nevét kell megadni – tehát a **-lgcc** feltételezi a lib**gcc**.a meglétét a szabványos (/lib, /usr/lib . . .) vagy az általunk megadott (-L .) helyeken.

A Make



A Program Maintenance Utility (Make) - feladata egy programrendszer elemeinek naprakészen tartása, a teljes technológiai lánc függőségi viszonyainak figyelembevételével. Vezérléséhez létre kell hozni egy - a teljes technológiai folyamatot leíró - file-t (neve többnyire Makefile, vagy *.mak) amely három részre tagolható.

Az első rész többnyire olyan **definíciókat** tartalmaz, mely belső használatú szimbólumokat hoz létre egyrészt a parancssorban megadható előállítási opciók feltételezett értékeire, az erőforrások elérésére, a speciális paraméterezésekre, és végül a fejlesztőeszközök programjainak szinonimáira vonatkozóan.

A második rész tartalmazza az **általános technológiai utasításokat**, azaz azt hogy az egyes fázisokban milyen file-okból, mely fejlesztő eszköz segítségével milyen terméket kívánunk létrehozni.

A harmadik rész az előállítani kívánt rész- illetve végtermékek **függőségi listáit** tartalmazza az adott lépcső bemenő elemeitől, kiegészítve az esetleges - átlagostól eltérő - fordítási/előállítási szabállyal.

Bár a gyakorlatban előforduló Makefile-ok elég bonyolultnak tűnnek, a működés módja mégis roppant egyszerű.

Ha a függőségi listában akár csak egy elem is frissebb mint a termék, akkor ez utóbbit újra elő kell állítani, vagy az általános, vagy pedig a lista alatt megadott speciális előállítási szabálynak megfelelően, és e szabályt rekurzívan újra alkalmazzuk a láncon.

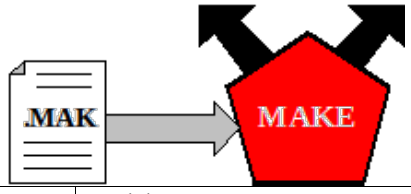
Az egyes listákból persze igen komoly méretű függőségi fa építhető fel, ahogy azt a későbbi példában is látni fogjuk.

Mivel az eszköz jóval általánosabban használható mint egy (pl. C) fordító, igen elterjedten alkalmazzák, és majd minden fejlesztő rendszerhez mellékelnek egyet. A használat módja is elég egységes (mióta a Microsoft régebbi, inkompatibilis darabja "kihalt"), paraméterek nélkül indítva az aktuális alkönyvtárban levő **Makefile** nevű file-t használja (ha van) vezérlőként, és előállítja az első függőségi listához tartozó végterméket (target). Paraméterként megadható egy, vagy több target is amit létre kell hoznia.

make [options ...] [target[s]]

Makefile nevének megadása	**-f** nomakefil.ext
Könyvtárváltás és make	**-C** path
definíció átadása	**-D**symbol=value
definíció megszüntetése	**-U**symbol

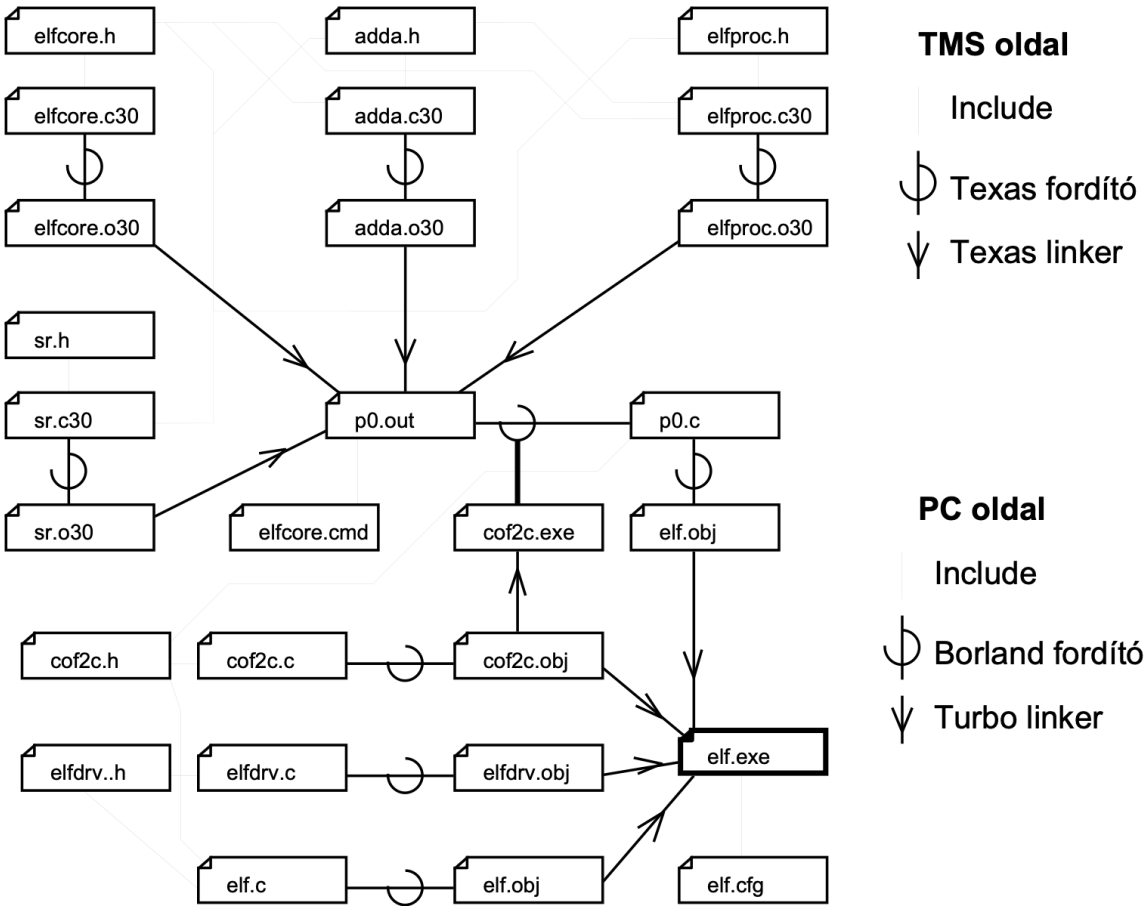
A vezérőfile szintaxisa:

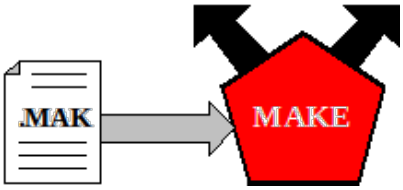


Szintax elem	Használata	Példa
Megjegyzések	# comment string	# Version 1.0B
Definíció	Symbol=Value	CC=gcc
Szimb. használat	**$(**Symbol)	$(CC) qqcs.c
Általános szabály	**.**Source_ext**.**Target_ext**:** How_to_create_target	.c.obj: $(CC) -c $*.c
Speciális szabály	Target_file**:** How_to_create_target	p0.c: cof2c p0.out p0.c

Példa:

A Make bemutatására egy régi beszédfelismerő szoftver-rendszer teszt programja lehet a legalkalmasabb. Itt egy kétprocesszoros rendszer programozásáról volt szó, ahol az egyik oldal egy IBM-PC/AT kompatibilis számítógép DOS operációs rendszerrel, a másik oldal pedig a PC-be illesztett jelfeldolgozó (DSP) kártya. A végtermék egy teszt program (elf.exe) amely letölti a magába ágyazott (p0.out, p0.c...) TMS kódot a DSP kártyába, elindítja azt, és adatcsere regisztereken keresztül kommunikál vele. A program előállításához szükséges technológiai vázlat az alábbi ábrán látható.

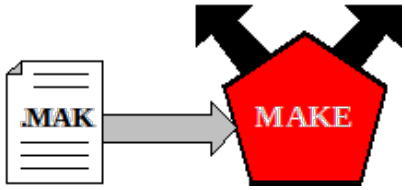




Itt a BORLAND Make eszközét használtuk fel a program előállításához, az alábbi, elf. vezérlőfile felhasználásával. A file a már ismertetett eszközöket használja az egyes lépések végrehajtásához, és csak egyetlen különleges eleme van a technológiai láncnak (cof2c) amely a TMS oldal p0.out COFF formátumú tölthető kódját konvertálja C nyelven megfogalmazott inicializált adatterületté a kódbeágyazáshoz. Miután ez a konverter is fejlesztés alatt áll, ugyancsak a rendszer frissítendő részei közé tartozik.

A Makefile (elf.) tartalma:

```
#..................................................... ELF. .....
#...    ELF,     BME-TTT            1993, 2013                  ..
#...    ================            Gyorgy Horvath 06-12-1993   ..
#...                                                            ..
#...   Make sample programs for ELF DSP using ELFCORE           ..
#...                                                            ..
#...   The existence of the following Programmer's Development  ..
#...   Tools are assumed:                                       ..
#...                                                            ..
#...   - Borland C++ 3.1                                        ..
#...   - Texas TMS320C4x Development system ( C compiler )      ..
#...                                                            ..
.AUTODEPEND

.PATH.obj = .

#               *Translator Definitions*
CC = bcc +ELF.CFG
TASM = TASM
TLIB = tlib
TLINK = tlink
LIBPATH = I:\BORLANDC\LIB
INCLUDEPATH = I:\BORLANDC\INCLUDE

#               *Implicit Rules*
.c.obj:
  $(CC) -c {$< }

.c30.o30:
  cl30 -c -k -mb -eo.o30 -ea.a30 -fc{$< } -ii:\tms\c30\bin -i.

.out.c:
  cof2c {$< } > $*.c

.cpp.obj:
  $(CC) -c {$< }

#               *List Macros*

EXE_dependencies =  \0.85
 elf.obj \
 elfdrv.obj \
 p0.obj

#               *Explicit Rules*
#                                       - Create the primary target
.\elf.exe: elf.cfg $(EXE_dependencies)
  $(TLINK) /v/x/c/P-/L$(LIBPATH) @&&|
c0l.obj+
.\elf.obj+
.\elfdrv.obj+
.\p0.obj
.\elf
               # no map file
emu.lib+
```

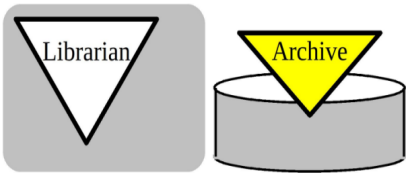



```
mathl.lib+
cl.lib
|

#               *Individual File Dep. *
#                                       -
Create the host side targets
elf.obj: elf.c elfdrv.h elfproc.h adda.h sr.h cof2c.h elf.cfg

elfdrv.obj: elf.cfg elfdrv.c elfdrv.h

p0.obj: p0.c cof2c.h elf.cfg

#                                       - Code embedding, out to C data
p0.c: p0.out cof2c.h cof2c.exe

#                                       - TMS side targets
sr.o30: sr.c30  elfcore.h elfproc.h adda.h sr.h

adda.o30: adda.c30  adda.h elfcore.h elfproc.h

elfproc.o30: elfproc.c30 elfcore.h elfproc.h

elfcore.o30: elfcore.c30 elfcore.h elfproc.h

p0.out: elfcore.o30 elfproc.o30 adda.o30 sr.o30  elfcore.cmd
  cl30 -mb -eo.o30 -fo elfcore.o30 -fo elfproc.o30 -fo adda.o30 -fo sr.o30
-z  -a  -o p0.out -m p0.map elfcore.cmd

#                                       - The code translator COF2C
cof2c.exe: elf.cfg cof2c.obj
  $(TLINK) /v/x/c/P-/L$(LIBPATH) @&&|
c0l.obj+
cof2c.obj
cof2c
               # no map file
emu.lib+
mathl.lib+
cl.lib
|


#               *Individual File Dependencies*
cof2c.obj: elf.cfg cof2c.c cof2c.h

#               *Compiler Configuration File*
elf.cfg: elf.
  copy &&|
-ml
-2
-C
-v
-vi-
-Ff=2000
-w-
-n.
-I$(INCLUDEPATH)
-L$(LIBPATH)
| elf.cfg
```

A rendszer naprakészen tartását a **make -f elf** parancs kiadásával biztosítottuk. Ekkor a módosításokkal összhangban csak a szükséges elemek előállítására került sor a függőségi vonalak mentén.
Ha mondjuk csak a cof2c konvertert szeretnénk frissíteni a többi elem érintetlenül hagyásával, akkor meg kell adni az egyedi célt is, azaz:

make -f elf cof2c.exe

Archiválás és Library



Az alábbiakban néhány, rendkívül hasonló funkciójú eszközt mutatunk be, amelyek akár teljes projekt, vagy csak a legfontosabb objektumaink archívumokban történő rendezett tárolását végzik. Az eszköz szolgáltatásainak szerves részét képezi az archívum tartalmának listázása, egyes elemek frissítése - cseréje, új elemek bevitele, elemek eltávolítása, illetve kiszedése.

Míg a legtöbb fejlesztőrendszer szerves részét képező **Library manager** (vagy rövidebben Librarian) kifejezetten a tárgykódú (object) állományok eljáráskönyvtárba történő szervezését végzi (az osztott eljáráskönyvtárakat a linkerrel szokták létrehozni), addig egy alkalmasan választott (az operációs rendszereknek többnyire van ilyen) **tömörítő** és/vagy **backup** program a teljes projekt rendszeres munka-mentését, vagy verziószámonkénti megőrzését végzi. Mint látni fogjuk a Linux rendszer alatt mindkét feladatra beépített eszköz használható.

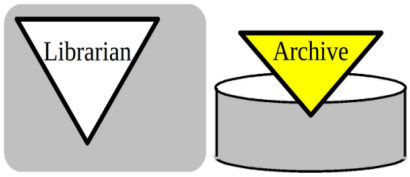
E két funkció furcsa egyesítésére szolgál a JAVA **jar** eszköze ahol az osztályokon alapuló objektumkönyvtár, illetve az archívum egy és ugyanazon **ZIP**-tömörített, de **jar** kiterjesztésű produktum lesz.

Visszatekintve a bevezető modell C programozói csoportjára, a sok apró, alacsony szintű eljárást már célszerű könyvtárba csomagolni. Könnyen meglehet hogy egy adott technológiai lánc kimenete is "csak" egy ilyen eljáráskönyvtár lesz, melyet a hozzá tartozó header file-okkal, illetve dokumentációval kiegészítve (mindezeket egy tömörítő-archív programmal összecsomagolva) a világ túlsó felén dolgozó programozók mint beszállított szoftvert fogják felhasználni. Az egyes eszközök indítása:

OS/CPU	Eszköz	Parancssor felépítése
Linux/*	GNU Librarian Archive	**ar** oper [opciók] arfile obj1 obj2 ... **tar** oper [opciók] **-f** tarfile file1 file2 ...
*/x86	Watcom	**wlib** [opciók] libfile oper ...
/	JAVA	**jar** oper [opciók] jarfile class1 class2 ...

A gyakrabban használt műveletek:

Opció	wlib	jar	GNU ar	GNU tar
Listázás		**t**	**t**	**-t**
Berakás	+objfile	**c** vagy **u**	**c** vagy **q**	**-c** vagy **-r**
Kiszedés	**:**modul **-x** (mindet)	**x**	**x**	**-x**
Frissítés	-+objfile	**u**	**ru**	**-u**
Törlés	-modul		**d**	



Linux rendszer esetén az archívum többfélkeképpen is tömöríthető, például **-z** kapcsoló megadása esetén **tar.gz** kiterjesztésű GZIP tömörített arhvívum, míg **-j** esetén **tar.bz2** lesz az eredmény. A Windows környezetet külön nem tárgyaljuk, hiszen ott számtalan jó hatásfokú tömörítő eszköz (pl. winrar, winzip, stb...) létezik

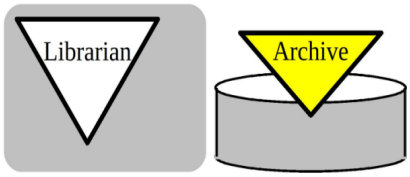
Példa:
Az alábbiakban a programozói munka során néha felbukkanó esetet mutatunk be a tárgykódú könyvtárak, és az általános célú archívumok kezelésére.

---

ar Vizsgáljuk meg a Linux rendszerben (ssh alpha ...) az alfanumerikus terminálokon történő ablakkezelés támogatására szolgáló **ncurses** eljáráskönyvtár tartalmát. Miután bemásoltuk a file-t a munkakönyvtárunkba, listázzuk tartalmát, és szedjük ki a safe_sprintf objektumot.

Az objdump segítségével listázzuk ki a szimbólum tábláját.

```
meres5@alpha:~/tmp$ cp /usr/lib/libncurses.a .
meres5@alpha:~/tmp$ ar tv libncurses.a
rw-r--r-- 0/0   1752 Jan  4 12:17 2011 hardscroll.o
rw-r--r-- 0/0   5528 Jan  4 12:17 2011 hashmap.o
 . . .
rw-r--r-- 0/0    868 Jan  4 12:17 2011 nc_panel.o
rw-r--r-- 0/0   1320 Jan  4 12:17 2011 safe_sprintf.o
rw-r--r-- 0/0  29520 Jan  4 12:17 2011 tty_update.o
 . . .
meres5@alpha:~/tmp$ ar x libncurses.a safe_sprintf.o
meres5@alpha:~/tmp$ objdump -t safe_sprintf.o

safe_sprintf.o:     file format elf32-powerpc

SYMBOL TABLE:
00000000 l    d  .text      00000000 .text
00000000 l    d  .data      00000000 .data
00000000 l    d  .bss       00000000 .bss
00000000 l    d  .comment   00000000 .comment
00000000 l    d  .note.GNU-stack 00000000 .note.GNU-stack
00000000 l    d  .gnu.attributes 00000000 .gnu.attributes
00000000 g     F .text      00000128 _nc_printf_string
00000000         *UND*      00000000 SP
00000000         *UND*      00000000 _nc_globals
00000000         *UND*      00000000 vsnprintf
00000000         *UND*      00000000 _nc_doalloc
00000000         *UND*      00000000 free
```



tar A tar file-ok illetve komplett könyvtári szerkezetek mentésére, visszaállítására szolgál, ahol a backup eszköz lehet szalag (a nevét innen kapta a parancs), floppy, vagy file. Az stdhello minden állományát archiváljuk egy stdhello.tar nevű file-ba, majd a tömörítés után vizsgáljuk annak hatékonyságát:

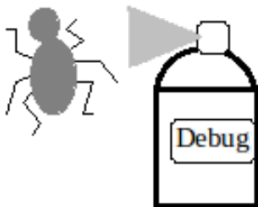
```
meres5@alpha:~$ tar -cv -f stdhello.tar ttmer99
ttmer99/
ttmer99/crt1.o
ttmer99/libc.a
ttmer99/libgcc.a
ttmer99/stdhello.c
ttmer99/stdhello.o
ttmer99/stdhello_dynamic
ttmer99/stdhello_static
meres5@alpha:~$ ls -l stdhello.tar
-rw-r--r-- 1 meres5 meres 2334720 Apr 15 13:21 stdhello.tar
meres5@alpha:~$ gzip stdhello.tar
meres5@alpha:~$ ls -l stdhello.tar.gz
-rw-r--r-- 1 meres5 meres 595948 Apr 15 13:21 stdhello.tar.gz
```

Miután a hálózaton lévő számtalan szoftver archívum (anonymous ftp sites) a Linux-os anyagokat ilyen tar.gz formában tárolják, célszerű megnézni a kicsomagolás módját is. Lépjünk bea ~/tmp könyvtárba, és ott csomagoljuk ki az imént összerakott, tömörített file-t:

```
meres5@alpha:~$ cd ~/tmp/
meres5@alpha:~/tmp$ tar -xvzf ../stdhello.tar.gz
ttmer99/
ttmer99/crt1.o
ttmer99/libc.a
ttmer99/libgcc.a
ttmer99/stdhello.c
ttmer99/stdhello.o
ttmer99/stdhello_dynamic
ttmer99/stdhello_static
meres5@alpha:~/tmp$ ls -la
total 12
drwxr-xr-x  3 meres5 meres 4096 Apr 15 13:24 .
drwx------ 20 meres5 meres 4096 Apr 15 13:21 ..
drwxr-xr-x  2 meres5 meres 4096 Apr 15 13:20 ttmer99
meres5@alpha:~/tmp$ find .
.
./ttmer99
./ttmer99/crt1.o
./ttmer99/libc.a
./ttmer99/libgcc.a
./ttmer99/stdhello.c
./ttmer99/stdhello.o
./ttmer99/stdhello_dynamic
./ttmer99/stdhello_static
```

jar A JAVA archívumokat jdk eszközkészlet nélkül is manipulálhatjuk – a jar állományt zip-re történő átnevezés után már egyszerű tömörített archívumként kezelhetjük – fájlokat

adhatunk hozzá, törölhetünk, cserélhetünk, majd visszanevezhetjük jar-rá.



A Debugger

A debugger feladata a fejlesztés alatt álló programok hibáinak felderítése. A felderítés eszköze lehet a töréspontok elhelyezése, lépésenkénti nyomon követés, a processzor regisztereinek és a program adatszerkezetének vizsgálata, valamint az úgynevezett watch-point-ok elhelyezése, amikor a program futása megakad ha egy változó értéke eleget tesz bizonyos, általunk megadott feltételeknek.

A felderítést nagyban megkönnyíti ha a szimbolikus, és a forráskóddal kapcsolatos információk is rendelkezésre állnak. Ez a szolgáltatás a fordítónak és linker-nek megadott parancssorbeli kapcsolókkal többnyire beállítható, ahogy az az előző fejezetek táblázataiban is szerepel.

Példa:

Az alábbiakban a Linux elterjedt eszközét villantjuk fel röviden.

---

gdb Vizsgáljuk meg a Linux rendszer alatt (ssh alpha ...) elkészített stdhello programot a gdb segítségével. Listázzuk ki a forrásnyelvű sorokat, helyezzünk el töréspontot a kiíratás sorára, futtassuk le a programot a töréspontig, vizsgáljuk meg az MsgHello nevű adatszerkezetet és a processzor regisztereit a töréspontban, léptessünk egyet a kiíratáson keresztül, és fejezzük be a munkát. Azaz:

```
meres5@alpha:~/tmp$ gcc -g -o stdhello stdhello.c
meres5@alpha:~/tmp$ gdb stdhello
.......
Reading symbols from /home/meres5/tmp/stdhello...done.
(gdb) list
43      /*          =====                                          */
44      void        main( void )
45      /*          =====                                          */
46      {
47      printf( MsgHello );
(gdb) break 47
Breakpoint 1 at 0x100004b0: file stdhello.c, line 47.
(gdb) r un
Starting program: /home/meres5/tmp/stdhello

Breakpoint 1, main () at stdhello.c:47
47      printf( MsgHello );
(gdb) print MsgHello
$1 = 0x100006fc "\nHello\n"
(gdb) print MsgHello[0]
$2 = 10 '\n'
(gdb) print &MsgHello[0]
$3 = 0x100006fc "\nHello\n"
(gdb) i reg
r0             0xfe8d69c     266917532
r1             0xffffea50    4294961744
r2             0xf7fd2720    4160562976
r3             0x1     1
r4             0xffffecc4    4294962372
r5             0xffffeccc    4294962380
r6             0xffffed18    4294962456
r7             0x0     0
r8             0x0     0
```

```
r9             0xf7fcb280    4160533120
r10            0x0    0
```

(folytatás)

```
r11            0xffec934     268355892
r12            0x1    1
r13            0x10018864    268535908
r14            0x0    0
. . . . .
r26            0x0    0
r27            0xf7ffd534    4160738612
r28            0xf7ffe018    4160741400
r29            0x0    0
r30            0xffebff4     268353524
r31            0xffffea50    4294961744
pc             0x100004b0    0x100004b0 <main+20>
msr            0x2d032       184370
cr             0x22000482    570426498
lr             0xfe8d69c     0xfe8d69c
ctr            0x1000049c    268436636
xer            0x0    0
orig_r3        0xf7fcc000    -134430720
trap           0x700  1792
(gdb) step
Hello
48      }
(gdb) cont
Continuing.

Program exited with code 07.
(gdb) quit
meres5@alpha:~/tmp$
```

Hasznos linkek

http://hessling-editor.sourceforge.net/ ..... The Hessling Editor

http://vim.sourceforge.net/download.php ..... Vim (text editor)

http://www.mingw.org/ .....Minimalist GNU for Windows

http://gcc.gnu.org/ .....GCC, the GNU Compiler Collection

http://www.openwatcom.org ..... Open Watcom