Univerzitet u Novom Sadu

Fakultet tehničkih nauka

Odsek za računarsku tehniku i računarske komunikacije

Upoznavanje sa MAKE alatom i MAKEFILE datotekama

Proces prevođenja (1/2)

- Prevođenje programa koji se sastoji od velikog broja izvornih datoteka i zaglavlja može da bude komplikovano i da oduzme mnogo vremena
- Kucanje dugačkih komandi za prevođenje koda je sklono greškama

Proces prevođenja (2/2)

- Primer: program se sastoji od izvornih datoteka, zaglavlja, biblioteka i njihovih zaglavlja kao na slici

```
app
|
|-- inc
|   |-- app.h
|   |-- factorial.h
|   |-- math.h
|
|-- src
|   |-- app.c
|   |-- factorial.c
|   |-- math.c
|
|-- lib
|    |-- libwrite.so
|
|-- libinc
|    |-- libwrite.h
|
|-- build
```

```
gcc -O2 -c -I../inc -I../libinc ../src/app.c -o app.o
gcc -O2 -c -I../inc ../src/factorial.c -o factorial.o
gcc -O2 -c -I../inc ../src/math.c -o math.o

gcc -L../lib -lwrite app.o factorial.o math.o -o
```

- Ovo se može zameniti jednom složenijom komandom
  gcc -O2 -I../inc -I../libinc -L../lib ../src/app.c -lwrite ../src/factorial.c ../src/math.c -o app
- Drugi način je lakši za korišćenje ali nije efikasan kada se samo nekoliko datoteka promenilo
- Prvi metod je brži kada se promenilo samo nekoliko datoteka, ali postoji rizik da se ne prevede ponovo neka datoteka u kojoj je došlo do promene

make alat (1/2)

- Obično su ulazi projekta izvorne datoteke, a izlaz je jedna ili više izvršnih datoteka ili biblioteka
- `make` je alat koji se koristi za efikasno prevođenje programa
- Automatski detektuje delove programa koji treba da se prevedu i izdaje komande da se prevođenje izvrši
  - Pokreće prevođenje ukoliko izlaz već ne postoji ili ako je došlo do promene u nekoj datoteci od koje izlaz zavisi
- make čita instrukcije iz tekstualne datoteke
- Podrazumevano ime datoteke je `makefile` ili `Makefile`

make alat (2/2)

- Opcijom `-f` možemo iskoristiti bilo koju datoteku za ulazne instrukcije
  - `make -f InstrukcijeZaPrevođenje`
- Opšti oblik sintakse `make`-a:
  - `make [-f makefile] [opcije] ... [ciljevi] ...`
- Razni debag ispisi u toku prevođenja se mogu uključiti opcijom `-d`
- Više informacija o opcijama se može naći u priručniku
  - `man make`

makefile

- Sadrži instrukcije uz pomoć kojih se od ulaza generišu izlazi
- Tekstualna datoteka
- Sastoji se od:
  - eksplicitnih pravila - opisuju kada i kako da se prevede jedna ili više datoteka
  - implicitnih pravila - opisuje kada i kako da se preveda grupa datoteka na osnovu njihovog imena
  - definicija promenljivih - definiše tekstualne promenljive koje će kasnije biti zamenjene
  - direktiva - naređuju izvršavanje akcija dok se analizira makefile
    - uključivanje drugog makefile-a
    - uslovno izvršavanje određenog dela
    - definicija promenljive u više linija
  - komentara - počinju sa # i traju do kraja linije

Eksplicitna pravila

- Pravilo definiše kada i kako da se prevedu datoteke da se dobije odredište pravila
- Sintaksa:
  - `Odredište: zavisnosti`
    `[tab] komanda`
- Uglavnom odredište predstavlja izlaz, a zavisnosti ulaz u pravilo
- `make` će izvršiti komandu ako
  - odredište ne postoji
  - datoteka navedena kao zavisnosti se promenila od poslednjeg prevođenja
  - komanda se promenila od poslednjeg prevođenja
- Eksplicitna pravila - eksplicitno data u makefile-u

Pravljenje objektne datoteke

- Mora da postoji pravilo gde je objektna datoteka odredište
- Zavisnosti su izvorne datoteke (i zaglavlja) potrebne za pravljenje objektne datoteke
- Komanda mora da obuhvata poziv prevodioca kako bi se prevele izvorne datoteke

```
app.o: app.c app.h math.h factorial.h libwrite.h
        gcc -O2 -c -I../inc -I../libinc ../src/app.c -o app.o
```

- Zavisnosti obuhvataju sva zaglavlja korišćena u `app.c` jer se u slučaju promene u nekom od zaglavlja prevođenje mora opet izvršiti

Pravljenje izvršne datoteke

- Mora da postoji pravilo gde je izvršna datoteka odredište
- Zavisnosti su sve objektne datoteke potrebne za pravljenje izvršne datoteke
- Komanda mora da obuhvata poziv povezivača koji pravi izvršnu datoteku

```
app: app.o factorial.o math.o
        gcc -L../lib -lwrite app.o factorial.o math.o -o app
```

- Ukoliko odredište ne postoji (`app`) ili se neka od zavisnosti promenila (bilo koja ulazna datoteka) komanda će se izvršiti i izvršna datoteka će biti napravljena

Promenljive

- Još jedna bitna komponenta makefile-ova
- Identifikator definisan u okviru makefile-a koji predstavlja tekstualni string (vrednost promenljive)
- Prilikom izvršenja pravila promenljive se zamenjuju sa njihovim vrednostima
- Čine kod čitljivijim i lakšim za konfigurisanje

```
CC = gcc
app.o: app.c app.h math.h factorial.h libwrite.h
        ${CC} -O2 -c -I../inc -I../libinc ../src/app.c -o app.o
```

- Promena prevodioca = promena vrednosti promenljive
- Vrednost promenljive se može postaviti i:
  - iz komandne linije - make all CC=/usr/bin/gcc
  - preko promenljive okruženja - export CC=/usr/bin/gcc

Korišćenje specijalnih karaktera (vajld-karti)

- Vajld-karte se koriste za referenciranje više datoteka jednim imenom
- Vajld-karte su:
  - * : menja bilo koji string od 0 ili više karaktera
  - ? : menja jedan karakter
  - [lista_karaktera] : menja jedan karakter sa nekim iz liste
- clean:
  rm -f *.o (briše sve objektne datoteke)
- print: print?.o (sve datoteke sa nazivom print i još
  komanda tačno jedan karakter će biti zavisnosti)
- print: print[0-9].o (sve datoteke sa nazivom print i još
  komanda jednim numeričkim karakterom će biti zavisnosti)
- Ukoliko je potrebno da se u imenu iskoristi karakter koji je i vajld-karta pre njega se piše \
  - print: print*.o (samo print*.o će biti zavisnost)

wildcard funkcija

- Razvijanje vajld-karti se automatski obavlja u okviru definicija mete i zavisnosti
- Šel skripta je zadužena za razvijanje u delu sa komandama (uključujući zavisnosti)
- Razvijanje se ne izvršava u ostalim delovima - npr. definicija promenljivih
- Za razvijanje vajld-karti u ovim delovima koristi se wildcard funkcija
- Sintaksa:
  - $(wildcard šablon)
- Primer
  - sources=$(wildcard *.c *.h) - sve datoteke sa ekstenzijama .c i .h

Zamene

- Zamene se koriste da izvrše određene promene nad vrednošću varijabla
- Sintaksa:
  - $(var:a=b)
  - ${var:a=b}
    - U vrednosti `var` svako `a` na kraju reči se menja sa `b`
    - `a` mora biti na kraju vrednosti ili praćeno razmakom
- Primer:
  - var1:=f1.o f2.o f3.o
    var2:=$(var1:.o=.c)
  - Posle izvršenja zamene `var2` ima vrednost `f1.c f2.c f3.c`

Implicitna pravila (1/3)

- Nije uvek neophodno definisati pravilo za svako odredište
- Za često korišćene standardne procedure postoje implicitna pravila
  - Npr. postoji pravilo za pravljenje .o od .c korišćenjem CC
- Moguće je sekvencijalno primeniti implicitna pravila
  - Npr. od .c napraviti .o, pa iskoristiti povezivač za pravljenje izvršne datoteke
- Da bi se koristilo implicitno pravilo potrebno je napisati pravilo bez komande ili ne napisati pravilo uopšte
  - app: app.o math.o factorial.o
    $(CC) -o app app.o math.o factorial.o
- Pošto se .o koristi kao zavisnost, a ne postoji eksplicitno pravilo u kom je odredište, traži se odgovarajuće implicitno pravilo

Implicitna pravila (2/3)

- Lista implicitnih pravila zavisi od operativnog sistema
- Uvek dostupna pravila su:
  - Prevođenje C programa - koristi CC
  - Prevođenje asemblerskih programa - koristi AS
  - Povezivanje objektnih datoteka - koristi LD
- Lista svih implicitnih pravila se može videti pokretanjem komande `make -p`
- Zabrana korišćenja implicitnih pravila se može postići na dva načina:
  - `-r parametar`
  - `-no-builtin-rules`

Implicitna pravila (3/3)

- Implicitna pravila koriste predefinisane promenljive
  - Menjanjem promenljivih utiče se na izvršenje pravila

- Imena programa: Argumenti programa:
  - AS – asembler — ASFLAGS – argumenti asemblera
  - AR – arhiver — ARFLAGS – argumenti arhivera
  - CC – C prevodilac — CFLAGS – argumenti C prevodioca
  - CPP – C preprocesor — CFLAGS – argumenti C preprocesora
  - CXX – C++ prevodilac — CXXFLAGS – argumenti C++ prevodioca

Definicija i redefinicija implicitnih pravila (1/2)

- Implicitno pravilo se definiše definisanjem šablona pravila
- Šablon pravila se razlikuje od običnog pravila po tome što sadrži znak % u okviru odredišta
- % odgovara podstringu koji ne može biti prazan
  - %.c - odgovara svakoj .c datoteci
  - %.o - odgovara svakoj .o datoteci
- % u zavisnostima odgovara istom podstringu kao i u odredištu
  - %.o:%.c
    komanda

    (definiše kako se bilo koja .o datoteka pravi od .c datoteke)

Definicija i redefinicija implicitnih pravila (2/2)

- Šablon pravila može da ima više od jednog odredišta
  - `make` smatra da će izvršavanje komande napraviti sva odredišta
  - Primer:
    - %.tab.c %.tab.h:%.y
      bison -d $<
- Implicitno pravilo se poništava definisanjem pravila sa istim odredištem i zavisnostima, ali bez komandi
  - Primer:
    - %.o : %.c

Automatske promenljive

- `make` dodeljuje vrednosti automatskim promenljivama kada se pronađe pravilo i počne izvršavanje
- Vidljive su samo iz komandi
- Ne mogu se koristiti u definiciji pravila

$@	Naziv datoteke iz odredišta
$%	Naziv člana odredišta, kada je odredište član arhive
$<	Naziv prve zavisnosti
$?	Naziv svih zavisnosti novijih od odredišta
$^	Nazivi svih zavisnosti razdvojeni razmacima bez duplikata
$+	Slično kao $^ s tim što se duplikati pojavljuju u redosledu u kom su navedeni

Primeri upotrebe automatskih promenljivih

- Napravi izvršnu datoteku povezivanjem svih zavisnosti
  - app: app.o math.o factorial.o
    $(CC) $^ -o app
- Napravi izvršnu datoteku, koja se zove isto kao odredište, povezivanjem svih zavisnosti
  - app: app.o math.o factorial.o
    $(CC) $^ -o $@
- Napravi izvršnu datoteku, koja se zove isto kao odredište, prevođenjem prve zavisnosti
  - math.o : math.c math.h
    $(CC) $(CFLAGS) $< -o $@

Lažno odredište (1/2)

- Korisno je imati komande koje ne prave direktno datoteke, već logički predstavljaju deo procesa prevođenja
  - clean:
    rm *.o app
- Postoje dva problema u vezi sa pravilom napisanim iznad:
  - konflikt sa postojećim imenima datoteka (ukoliko neko napravi datoteku clean u istom direktorijumu, pravilo se nikad neće izvršiti)
  - make će tražiti implicitno pravilo
- Rešenje - korišćenje specijalnog `.PHONY` odredišta
  - .PHONY clean:
    clean:
    rm *.o app

Lažno odredište (2/2)

- Moguće je dodati zavisnosti lažnim (`.Phony`) odredištima
- Primer - kada `makefile` pravi više od jedne izvršne datoteke zgodno je napraviti lažno pravilo koje pravi sve izlaze
  - all: app1 app2 app3
    .PHONY : all
    app1: app1.o math.o
    cc -o app1.o math.o
    app2: app2.o factorial.o
    cc -o app2.o factorial.o
    app3: app3.o factorial.o math.o
    cc -o app2.o factorial.o math.o
- Česta lažna odredišta su:
  - all - pravi sve izlaze
  - clean - briše sve izvršne i privremene datoteke
  - install - kopira sve izlaze na predefinisanu lokaciju

Primer makefile-a

```
SHELL = /bin/sh

OBJS = app.o factorial.o math.o
CFLAGS = -O2
CC = gcc
INCLUDES = -I ../inc -I ../libinc
LIBS = -lwrite
LDFLAGS = -L ../lib

hello:${OBJS}
	${CC} ${CFLAGS} ${LDFLAGS} ${INCLUDES} -o $@ ${OBJS} ${LIBS}

clean:
	rm -f *.o app

%.o:%.c
	${CC} ${CFLAGS} ${INCLUDES} -c $< -o $@
```