Praktikum iz operativnih sistema 1

Vežbe 5

Pregled gradiva

- `gcc i gdb`
- `GNU make`

Richard Stallman , izvor: Wikipedia

Primer

- Napisati program na C jeziku koji određuje da li je zadata godina prestupna ili ne.
  Godinu zadati kao parametar main funkcije.

Primer
- rešenje -

```
// hello.c
#include <stdio.h>
#include <string.h>
int main(int argc, char* argv[]){
	if(argc < 2) {
		printf(“Enter year parametar\n");
		return 1;
	}
	else {
		int year;
		char isLeapYear[4] = {0};

		sscanf(argv[1], "%d", &year);

		if(!(year % 4 == 0 &&  year % 100 != 0 || year % 400 == 0) )
			strcpy(isLeapYear, "not");

		printf("%d is %s leap year\n", year, isLeapYear);
	}
	return 0;
}
```

Prevodjenje i pokretanje `hello` programa -`gcc`-

- `gcc` kompajler prevodi `c` programe
- Ako nije instaliran, skinuti ga sa `apt-get`

```
$ sudo apt-get install gcc
```

- Provođenje i pokretanje `hello.c`

```
$ gcc -o hello hello.c
$ ./hello 2020
```

gcc
- opcije pri prevođenju -

- `$ gcc -o output hello.c`
  - Zadavanje naziva izlaznom fajlu
- `$ gcc -S hello.c`
  - Stvaranje asm fajla (sa podrazumevanim imenom)
- `$ gcc -c hello.c`
  - Stvaranje objektnog fajla (sa podrazumevanim imenom)
- `$ gcc -O3 hello.c`
  - Prevođenje sa 3. stepenom optimizacije.
  - Uvećava se veličina i vreme prevođenja programa, a smanjuje se izvršno vreme
  - Podrazumevano je isključena
- `$ gcc -Wall hello.c`
  - Ispisuju se sva upozorenja
- `$ gcc -w hello.c`
  - Uklanjaju se sva upozorenja
- `$ gcc -Ifolder_include hello.c`
  - Relativna putanja do zaglavlja (`.h`) u folderu `folder_include`
- `$ gcc -g hello.c`
  - U program se uključuje tabela simbola koja je neophodna tokom procesa debagovanja

Debagovanje programa
- gdb -

- Moramo da opet prevedemo kod, ovog puta sa `-g` opcijom
  ```
  $ gcc -g -o hello hello.c
  ```
- Pokretanje debugger-a
  ```
  $ gdb ./hello
  ```
- Komande za debagovanje
  - **`run`** : Pokreće `hello` program
  - **`run arg1 arg2`** : Pokreće `hello` program sa parametrima za `main`
  - **`break hello.c:5`** : Postavlja breakpoint na liniji 5 u fajlu `hello.c`
  - **`break hello.c:main`** : Postavlja breakpoint na početak funkcije `main`
  - **`info breakpoint`** : Lista sve breakpoint tačke
  - **`clear hello.c:main`** : Uklanja breakpoint sa početka funkcije `main`
  - **`s`** : Step into
  - **`n`** : Step over
  - **`c`** : Continue
  - **`print year`** : Ispis vrednosti promenljive `year`
  - **`quit`** : Izlazak iz debagovanja

GNU make
- uvod -

- Komanda `make` - automatsko ažuriranje zavisnih fajlova
- Česta primena kod `c/c++` jezika
- Rekompilacija delova programa usled promene `.c, .cpp, .h` ili `.o` fajlova

GNU make
- način rada (1/2) -

- Poziv komande

```
$ make [-f makefile] [options] … [target]
```

- **makefile** sadrži listu pravila:

```
# ovo je makefile
fajl: fajl_1 fajl_2 ...
      komanda_1
      komanda_2
```

- `# komentar`
- **fajl** predstavlja ime zavisnog fajla i definiše početak pravila za njegovo generisanje/ažuriranje (**target** u gornjoj komandi)
  - Primeri zavisnih fajlova su izvršni ili objektni fajlovi
- **fajl_1 fajl_2...** je lista fajlova odvojenih razmakom od kojih zavisi **fajl**
  - Ukoliko neki od fajlova iz liste zavisnosti ne postoji, mora postojati pravilo koje generiše taj fajl
- **komanda_1, komanda_2 ...** su niz shell komandi koje se izvršavaju da bi se ažurirao ili kreirao **fajl**
  - **Svaku komandu pišemo kao za bash shell**
  - **Svaka komanda mora da bude uvučena tab znakom!**

GNU make
- način rada (2/2) -

```
# ovo je makefile
fajl_a: fajl_1 fajl_2 ...
    komanda_1
    komanda_2
    ...
fajl_b: fajl_3 fajl_4 ...
    komanda_3
    komanda_4
    ...
```

Poziv make komande za generisanje/ažuriranje **fajl_b** čije se pravilo nalazi u fajlu **makefile**

```
$ make -f makefile fajl_b
```

- Ukoliko se ne navede pravilo (**fajl_b** u gornjoj komandi), podrazumeva se prvo u zadatom makefile-u (**fajl_a**)
- Ukoliko se ne navede naziv fajla opcijom `-f` (**makefile** u gornjoj komandi), podrazumevano se traže oni koji se zovu **GNUmakefile**, **makefile** ili **Makefile**, tim redosledom
- Izvršavanje gornje **make** komande:
  - Pokrenuće se redom `komanda_3, komanda_4 …` za **fajl_b** ako je:
    - neki fajl iz liste zavisnosti, `fajl_3 fajl_4 ...`, izmenjen ili
    - neki fajl iz liste zavisnosti, `fajl_3 fajl_4 ...`, ne postoji ili
    - **fajl_b**, kao zavisni fajl, nije još uvek napravljen.
    - Inače, **make** komanda nema efekta.

GNU make
- motivacija -

main.c	hello.c	hello.h

`#include "hello.h"`
 	
`int main(){`
`  // extern function`
`  hello();`
`  return 0;`
`}`
	`#include <stdio.h>`
	
	`int getYear() { return 2020; }`
	
	`void hello(){`
	`  printf("Hello! It's %dth!\n", getYear());`
	`}`
		`void hello();`
		`int  getYear();`

- Kod dat gore se nalazi u folderu `makefiles1.x`
- Kompajliranje gornjeg koda:

  `$ gcc -o prog main.c hello.c -I.`
- Problemi:
  - #1 Posle svake izmene `.c` i `.h` fajlova morali bismo da ukucavamo istu komandu ispočetka
  - #2 Uvek bismo rekompajlirali sve fajlove, iako za tim možda nema potrebe
- Rešenje: make fajl

GNU make
- rešenje 1.0 -

- U prilogu je najjednostavniji make fajl, `makefile1.0`

```
# makefile1.0
prog: main.c hello.c hello.h
	gcc -o prog main.c hello.c -I.
```

tab!

- Ovim je rešen problem #1, ali ne i problem #2
  - Svaki put kada se promeni `main.c`, praviće se i novi `main.o` i `hello.o`

GNU make
- rešenje 1.1 -

- Poboljšana verzija

```
# makefile1.1
CC = gcc
CFLAGS = -Wall -I.
prog: main.o hello.o
	$(CC) -o prog main.o hello.o
hello.o: hello.c
	$(CC) -o hello.o -c hello.c $(CFLAGS)
main.o: main.c hello.h
	$(CC) -o main.o -c main.c $(CFLAGS)
```

- Sada će `.o` fajlovi da se prave samo ako su menjani odgovarajući `.c` fajlovi
- Kako da smanjimo mogućnost slovnih grešaka prilikom pisanja pravila?

GNU make
- rešenje 1.2 -

```
# makefile1.2
CC = gcc
CFLAGS = -Wall -I.

prog: main.o hello.o
   $(CC) -o $@ $^
hello.o: hello.c
   $(CC) -o $@ -c $^ $(CFLAGS)
main.o: main.c hello.h
   $(CC) -o $@ -c $< $(CFLAGS)
#  $(CC) -o $@ -c $(word 1, $^) $(CFLAGS)
```

- `$@` naziv zavisnog fajla iz pravila
- `$<` naziv prvog fajla iz liste zavisnosti
- `$^` cela lista zavisnosti
- `$(word 1, $^)` prvi fajl unutar cele liste zavisnosti
- Problem skalabilnosti: Dodavanje novih `.c` fajlova iziskuje dodavanje novih `.o` pravila

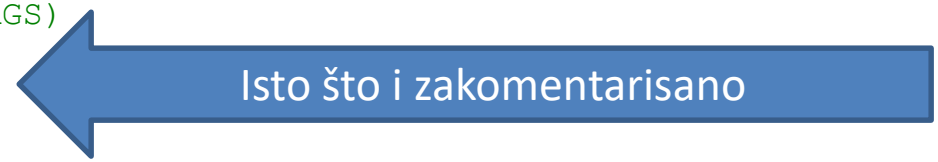
GNU make
- rešenje 1.3 -

```
# makefile1.3

CC = gcc
CFLAGS = -Wall -I.

prog: main.o hello.o
    $(CC) -o $@ $^
#hello.o: hello.c
#   $(CC) -o $@ -c $^
#main.o: main.c hello.h
#   $(CC) -o $@ -c $< $(CFLAGS)
#   $(CC) -o $@ -c $(word 1, $<) $(CFLAGS)
%.o: %.c
    $(CC) -o $@ -c $< $(CFLAGS)

main.o: hello.h
```



- Ubačena su generička pravila za `.o` fajlove
- Pošto generička lista zavisnosti nije dovoljna za main.o, ona je dopunjena dodatnom listom zavisnosti
- Obratiti pažnju da pravilo ne mora da ima niz komandi i da može da se piše iz više delova, kao što je slučaj sa `main.o`
- Da li je moguće pokupiti zavisnosti objektnih fajlova prilikom njihovog kreiranja, tako da ne mora ručno da se piše poslednja linija?

GNU make
- rešenje 1.4 -

```
# makefile1.4

CC = gcc
CFLAGS = -w -I.

prog: main.o hello.o
    $(CC) -o $@ $^

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

#main.o: hello.h

include main.d hello.d

#main.d
#main.o: main.c hello.h

#hello.d
#hello.o: hello.c
```

Rezultat komande se upisuje u fajlove .d

Sada nema potrebe za ovom linijom

Uključuju se generisani main.d i hello.d fajlovi čiji je sadržaj dat dole

- `-MM` opcija gcc kompajlera generiše listu zavisnosti objektnog fajla koji se kreira
- Pravi se fajl sa ekstenzijom .d koji ima isti naziv kao i kreirani objektni fajl

GNU make
- rešenje 1.5 -

```
# makefile1.5

CC = gcc
CFLAGS = -w -I.
OBJ = main.o hello.o

prog: $(OBJ)
	$(CC) -o $@ $^
%.o: %.c
	$(CC) -o $@ -c $< $(CFLAGS)
%.d: %.c
	$(CC) -MM  $< > $@

include main.d hello.d

clean:
	rm -f *.o
	rm -f prog
	rm -f *.d
	rm -f *~
```

- Dodat je makro `OBJ` koji sadrži listu objektnih fajlova od kojih zavisi izvršni fajl
- Dodato je `clean` pravilo bez liste zavisnosti
  - Takvo pravilo se naziva akcijom
  - Akcije se mogu uvek izvršavati, za razliku od pravila sa listom zavisnosti koja se izvršavaju samo ako nešto nije ažurirano

```
$ make -f makefile1.5 clean
```

GNU make
- rešenje 1.6 -

```
# makefile1.6

CC = gcc
CFLAGS = -w -I.
#OBJ = main.o hello.o
OBJ = $(wildcard *.o)

prog: $(OBJ)
    $(CC) -o $@ $^

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

include main.d hello.d

clean:
    rm -f *.o
    rm -f prog
    rm -f *.d
    rm -f *~

.PHONY: clean
```

- Ukoliko postoji fajl koji se zove isto kao i akcija, akcija ne može da se izvrši
  - Rešenje: dodati naziv akcije u listu zavisnosti pravila koje se zove `.PHONY`
- Makro `OBJ` je izmenjen, tako da se koristi `wildcard` make funkcija za automatsko generisanje liste objektnih fajlova na osnovu šablona
  - Šta je ovde problem?

GNU make
- rešenje 1.7 -

```
# makefile1.7
CC = gcc
CFLAGS = -w -I.
SRC = $(wildcard *.c)
#SRC = main.c hello.c
OBJ = $(patsubst %.c, %.o, $(SRC))
#OBJ = main.o hello.o

prog: $(OBJ)
    $(CC) -o $@ $^
%.o: %.c
     $(CC) -o $@ -c $< $(CFLAGS)
%.d: %.c
     $(CC) -MM  $< > $@
include main.d hello.d

clean:
    rm -f *.o
    rm -f prog
    rm -f *.d
    rm -f *~

.PHONY: clean
```

- Generisanje liste .o fajlova na prethodni način nije bio dobar, jer .o fajlovi možda ne postoje, pa je OBJ makro prazan
- Rešenje je da se OBJ generiše na osnovu liste .c fajlova, jer se .o i .c fajlovi zovu isto
- Korišćenjem patsubst make funkcije se radi konverzija liste .c fajlova u istu listu .o fajlova

GNU make
- primer 2.0 -

```
# makefile2.0
# Voditi racuna da nema blanko znakova nakon definisanja putanja
IDIR = ../include
OBJDIR = obj
SRCDIR = .
CC = gcc
CFLAGS = -w -I$(IDIR)
PROGRAM = prog
SRC = $(wildcard $(SRCDIR)/*.c)
OBJ = $(patsubst $(SRCDIR)/%.c,$(OBJDIR)/%.o,$(SRC))
DEP = $(patsubst $(SRCDIR)/%.c, $(SRCDIR)/%.d, $(SRC))
$(PROGRAM): $(OBJ)
    $(CC) -o $@ $^
$(OBJDIR)/%.o: $(SRCDIR)/%.c
    $(CC) -o $@ -c $< $(CFLAGS)
%.d: %.c
     $(CC) -MM -MT $(OBJDIR)/$*.o $< $(CFLAGS) > $@
include $(DEP)

clean:
    rm -f $(OBJDIR)/*.o
    rm -f *.d
    rm -f $(PROGRAM)
    rm -f *~

.PHONY: clean
```

- Dati makefile odgovara novom kodu koji se nalazi u folderu `makefiles2.x`
- Kod se sada ne nalazi u jednom istom folderu, pa moramo da pazimo na tačne putanje
- Komanda **–MM** kaže gcc kompajleru da generiše pravilo sa zavisnostima
- opcija **–MT** služi za postavljanje naziva pravila (sada to pravilo neće biti recimo main.o, nego obj/main.o)

Literatura

- https://gcc.gnu.org/onlinedocs/
  - Kompletna dokumentacija za `gcc` kompajler
- https://www.gnu.org/software/gdb/documentation/
  - Kompletna dokumentacija za `gdb` debager
- http://www.gnu.org/software/make/manual/make.html
  - Kompletna dokumentacija za `make` komandu