Politechnika Wrocławska

Programowanie Systemowe

LABORATORIUM

Temat: **Kompilator gcc, pliki Makefile i zależności**

Hanna Kieszek – 283797

03.11.2025

1. Cel ćwiczenia

Poznanie narzędzia gcc do kompilowania programów napisanych w c lub c++. Wykorzystywanie komend gcc w pliku Makefile.

2. Wykorzystane narzędzia

1. Debian
2. Terminal

3. Przebieg laboratorium

3.1. Stworzenie pliku helloworld w języku c i pliku Makefile

Pierwsza część zadania polegała na tym aby utworzyć plik w języku c który miał wypisywać w terminalu Hello World. Aby utworzyć ten plik została wykonana komenda touch plik.c.

Zawartość pliku hello.c została zamieszczona poniżej:

```
#include <studio.h>

Int main() {
	printf(“Hello World”);
}
```

Następnie został utworzony plik Makefile który zawierał dwie reguły i jedną zmienną globalną. Pierwsza reguła all tworzyła katalog o nazwie podanej w zmiennej globalnej OUTDIR a następnie za pomocą kompilatora gcc kompilowała plik plik.c. Skompilowany plik był zapisywany w folderze OUTDIR pod nazwą hello.

```
OUTDIR=build

all:
	mkdir -p $(OUTDIR)
	gcc -o $(OUTDIR)/hello plik.c
```

W komendzie mkdir została zastosowana opcja -p która odpowiada za to, że jeśli katalog o tej nazwie już istnieje to nie zostanie on ponownie utworzony ani nie zostanie wyświetlony błąd.

Natomiast w gcc została użyta opcja -o która pozwala ustalić do jakiego pliku zostanie zapisany skompilowany plik.c

Została stworzona reguła clean która po wywołaniu usuwała katalog OUTDIR:

```
clean:
	rm -rf $(OUTDIR)
```

Opcja -rf oznacza, że katalog będzie usunięty z całą jego zawartością oraz wymusza usunięcie bez pytania o potwierdzenie oraz ignoruje błąd jeżeli nie znajdzie określonego w zapytaniu folderu.

3.2. Modyfikacja pliku makefile

W drugiej części zadania w pliku makefile zostały utworzone kolejne zmienne globalne w których znajdują się plik wejściowy, używany kompilator oraz folder źródłowy. Zmienna przechowująca kompilator jest użyteczna w momencie gdy zostałby on zmieniony na inny. Wtedy zamiast zmieniać go w każdym miejscu, należałoby zmienić tylko zawartość zmiennej.

Reguła all została zmodyfikowana tak aby była w niej zawarta nowa zmienna dla kompilatora oraz folder źródłowy, natomiast jej działanie nie zostało zmienione.

```
OUTDIR=build
PLIK=plik.c		#zmienna z plikiem
SRCDIR=src		#zmienna z folderem źródłowym
CC=gcc			#zmienna z kompilatorem

all:
	mkdir -p $(OUTDIR)
	$(CC) -o $(OUTDIR)/hello $(SRCDIR)/$(PLIK)
```

Reguła clean nie została zmieniona.

3.3. Zastosowanie zależności w pliku Makefile

Zależności są używane do definiowania co jest potrzebne do wykonania danej reguły i jeżeli te instrukcje nie zostały wcześniej wykonane, to program wykona je automatycznie. Zależności nadają kolejność z jaką reguły mają być wykonane aby cały program poprawnie działał.

W trzeciej części zadania do zmiennych została dodana zmienna TARGET która zawiera nazwę pliku który będzie wynikiem wykonania make.

```
OUTDIR=build
PLIK=plik.c
SCRDIR=scr
CC= gcc
TARGET=hello
```

Reguła all zawiera zależność do reguły $(TARGET) więc po wywołaniu make zostanie sprawdzone czy instrukcje zawarte w $(TARGET) zostały wykonane. Jeśli nie, zostaną one wykonane automatycznie.
Działanie reguły clean pozostaje niezmienne. Ma ona usuwać folder OUTDIR.

```
all: $(TARGET)

clean:
	rm -rf $(OUTDIR)
```

Została utworzona reguła $(TARGET). Ma ona zależność do reguły plik.o i do reguły $(OUTDIR). Zawiera ona instrukcje do jednoczesnego kompilowania i linkowania pliku plik.o i zapisywania go do pliku $(TARGET). Gdy reguła zostanie wykonana zostanie utworzony plik hello w folderze build.

Plik.o to wynik działania reguły o tej samej nazwie dlatego też jest utwożona do niej zależność, ponieważ reguła $(TARGET) nie mogłaby zostać wykonana gdyby ten plik nie istniał. Jeżeli program nie znajdzie pliku plik.o lub folderu build, wywoła regułę plik.o lub $(OUTDIR).

```
$(TARGET):  plik.o $(OUTDIR)
	$(CC) $(OUTDIR)/plik.o -o $(OUTDIR)/$(TARGET)
```

Reguła plik.o jest zależna od reguły $(OUTDIR) gdyż w komendzie jest podane, że plik.o ma być zapisywany w folderze $(OUTDIR) za którego utworzenie odpowiada reguła o tej samej nazwie. Więc zanim zostanie ona wykonana, program sprawdzi czy katalog build już istnieje. Jeżeli nie, wykona on regułę $(OUTDIR).

Komenda zawarta w regule plik.o odpowiada za wyłącznie skompilowanie pliku źródłowego zawartego w zmiennej $(PLIK) o czym świadczy opcja -c. Kiedy ta opcja występuje, plik jest tylko kompilowany a nie kompilowany i linkowany. Po wykonaniu tej reguły zostanie utworzony plik.o w katalogu build na podstawie pliku plik.c z katalogu src.

```
plik.o:   $(OUTDIR)
	$(CC) -c $(SRCDIR)/$(PLIK) -o $(OUTDIR)/plik.o
```

Reguła $(OUTDIR) tworzy katalog o nazwie podanej w zmiennej o tej samej nazwie.

```
$(OUTDIR):
	mkdir $(OUTDIR)
```

4. Podsumowanie

Kompilator gcc opcje:
-o - określa pod jaką nazwą ma być zapisany skompilowany plik
-c - opcja dzięki której plik może zostać wyłącznie skompilowany

Zależności między regułami informują o tym która jest potrzebna do wykonania której oraz poniekąd określają kolejność ich wykonywania.