Τζιβάρας Βασίλης [vtzivaras@gmail.com]

Πανεπιστήμιο Ιωαννίνων
Τμήμα Πληροφορικής

ΠΕΡΙΕΧΟΜΕΝΑ

1. Εισαγωγή

Το Makefile είναι ένα εργαλείο που μπορείτε να χρησιμοποιήσετε όταν έχετε να κάνετε με ογκώδη προγράμματα. Εκεί προφανώς θα υπάρχουν πολλά αρχεία και ίσως κάποιες βιβλιοθήκες. Η μεταγλώττιση όλων των αρχείων και βιβλιοθηκών κάθε φορά εκτός από χρονοβόρα πολλές φορές δεν είναι και αναγκαία. Μπορεί δηλαδή να έχουμε μια μικρή αλλαγή σε ένα αρχείο σε μια γραμμή και χωρίς το Makefile να μεταγλωττίζονται όλα από την αρχή.

2. Makefiles vs Old Style Coding

Ας υποθέσουμε οτι έχουμε τα αρχεία *file1.c*, *file2.c*, *file3.c* και μια βιβλιοθήκη *mylib.h*. Αν θέλουμε να τα μεταγλωττίσουμε και να βγάλουμε ένα εκτελέσιμο θα εκτελούσαμε,

```
$gcc file1.c file2.c file3.c
```

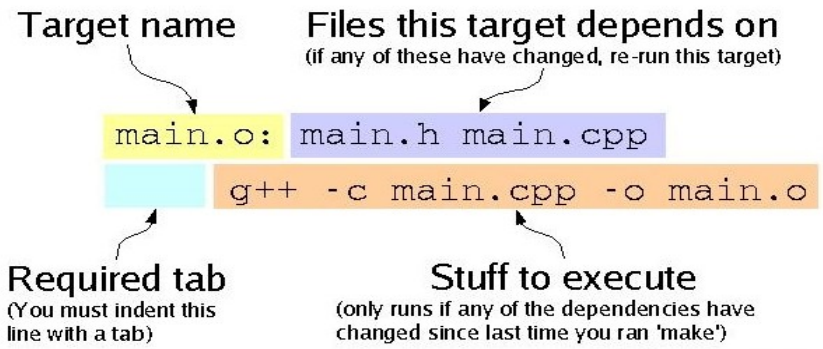
έχοντας κάνει include την βιβλιοθήκη μας σε ένα από αυτά. Σκεφτείτε πως αλλάζουμε μια γραμμή στο file2.c αρχείο και πρέπει να κάνουμε πάλι compile όπως παραπάνω. Σκεφτείτε τώρα ένα πιο μεγάλο project με 20 αρχεία και 10 βιβλιοθήκες ή ακόμη και ένα ολόκληρο λειτουργικό σύστημα όπως το Minix. Εκεί αν δεν υπάρχουν τα Makefiles έχουμε υπερβολικά πολύ αργό compilation και χάσιμο χρόνου. Αυτός είναι και ο κυριότερος λόγος που χρησιμοποιούμε τα Makefiles.

Σε αυτό το σημείο θα πρέπει να έχετε καταλάβει γιατί πρέπει αν τα χρησιμοποιούμε. Άλλα δυο σημαντικά πλεονεκτήματα είναι οτι, τα makefiles έχουν έναν μηχανισμό που καταλαβαίνουν αν έχει αλλάξει η όχι ένα αρχείο απο την τελευταία φορά που έγινε το compilation. Με αυτό τον τρόπο αν έχουμε δέκα αρχεία και αλλάξει το ένα δεν τα ξαναγίνει compile στα υπόλοιπα. Το δεύτερο είναι οτι μπορούμε με δυο μόνο λέξεις “make all” να μεταγλωττίσουμε τον κώδικα μας όσο πολύπλοκος κι αν είναι και όσες βιβλιοθήκες κι αν χρειάζεται.

3. Ανάπτυξη του Makefile

3.1 Δομή του αρχείου Makefile

Ένα Makefile είναι ένα απλό αρχείο άλλα με συγκεκριμένη σύνταξη.



Αποτελείται απο 4 κομμάτια όπως φαίνεται και παραπάνω. Τους *στόχους(target)*, τις *εξαρτήσεις(dependencies)*, που συνήθως είναι τα ονόματα αρχείων απο τα οποία εξαρτάται ο στόχος και μπορεί να είναι και ονόματα στόχων απο άλλους κανόνες. Τις ενέργειες(actions), που εκτελούνται με τη σειρά που ορίζονται όταν κάποιο απο τα αρχεία που αναφέρονται στις εξαρτήσεις έχει νεότερη ημερομηνία από το στόχο. Μπορεί να είναι ότι εντολές θέλετε και πρέπει υποχρεωτικά να ξεκινούν με TAB. *Η παράλειψη του αρχικού TAB σε κάθε γραμμή ενέργειας είναι και το συνηθέστερο λάθος στην*

κατασκευή του Makefile.

Η λογική πίσω από το Makefile είναι όταν θα το εκτελέσουμε να ψάξει το target(εξού και το όνομα :P), θα δει ότι για να ολοκληρωθεί ο στόχος χρειάζονται κάποια depedencies οπότε ψάχνει να βρει απο τι εξαρτάται και ορίζει κάθε φορά ως τρέχων στόχο ένα depedency. Αφού ολοκληρωθεί πηγαίνει στο επόμενο μέχρι να ολοκληρωθούν όλα τα dependencies και ταυτόχρονα ο αρχικός μας στόχος. Όταν υπάρχει ένα τουλάχιστον dependency θα υπάρχει και μια action για αυτό(συνήθως κάποια εντολή που θα εκτελεί το πρόγραμμα πχ. gcc file1.c).

Στο παρακάτω παράδειγμα το all είναι ο αρχικός μας στόχος. Για να ολοκληρωθεί αυτός πρέπει αν ολοκληρωθούν οι στόχοι main και hello που είναι εξαρτήσεις για το all. Για να ολοκληρωθεί το main πρέπει να γίνει ένα g++ και αντίστοιχα και για το hello. Οπότε αφού τελειώσει η μεταγλώττιση έχει ολοκληρωθεί ο αρχικός στόχος all.

```
all: main hello
main:
	g++ main.cpp
hello:
	g++ hello.cpp
```

3.2 Εξαρτήσεις(dependencies)

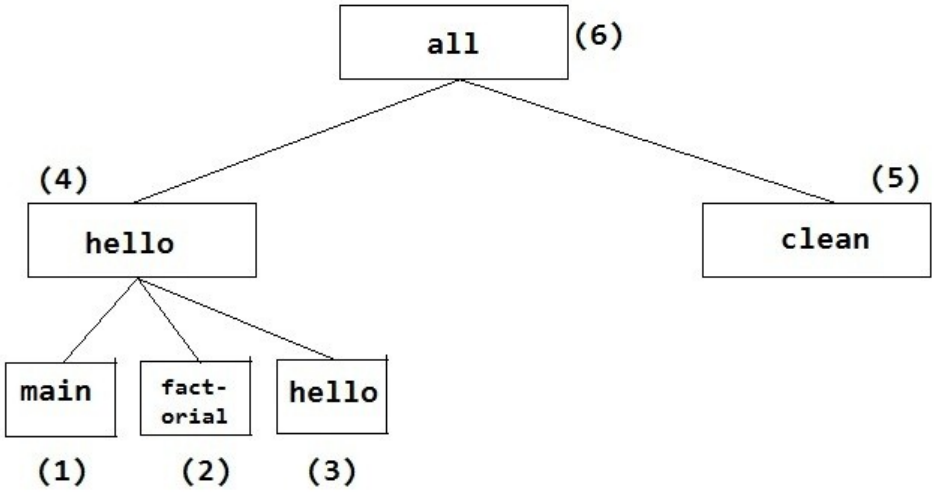
Τα dependencies δεν είναι απαραίτητο να υπάρχουν σε ένα makefile αλλά η ύπαρξή τους βοηθά ώστε να μην κάνουμε compile πολλά αρχεία συγχρόνως! Έτσι το compilation θα γίνει μόνο στο αρχείο στο οποίο έγινε και η αλλαγή.

Το παρακάτω αποτελεί ένα παράδειγμα makefile με depedencies. Πάλι η δομή αποτελείται απο στόχους, εξαρτήσεις και τέλος ενέργειες μόνο που τώρα ως ενέργειες αντί να έχουμε κάποιο gcc/g++ και να τελειώνει η ιστορία εδώ έχουμε κάποιον άλλο στόχο. Ο νέος στόχος λέγεται εξάρτηση(depedencies).

```
all: hello clean

hello: main.o factorial.o hello.o
	g++ main.o factorial.o hello.o -o hello

main.o: main.cpp
	g++ -c main.cpp

factorial.o: factorial.cpp
	g++ -c factorial.cpp

hello.o: hello.cpp
	g++ -c hello.cpp

clean:
	rm -rf *.o
```

Μπορείτε να φανταστείτε την δομή του Makefile ως ένα δέντρο όπως φαίνεται και παρακάτω, όπου να

ολοκληρωθεί ο πατέρας πρέπει να ολοκληρωθούν τα παιδιά του.



Ας αναλύσουμε τώρα το παραπάνω παράδειγμα. Ο αρχικός στόχος είναι ο all, για να γίνει αυτός θα πρέπει αν γίνουν οι στόχοι hello και clean. Άρα μπορούμε να πούμε οτι ο all εξαρτάται απο τους hello και clean. Για να γίνει ο hello χρειάζονται άλλοι 3 στόχοι σε καθένα απο τους οποίους χρειάζεται να γίνει ένα αρχείο compile(έτσι θα γίνεται compile ένα αρχείο κάθε φορά που γίνεται το make). Για να γίνει ο clean στόχος δεν χρειάζεται κάτι οπότε απλα εκτελείται η εντολή που διαγράφει όλα τα .o αρχεία.

3.3 Μεταβλητές(variables)

Στην αρχη καθε Makefile μπορουμε να ορισουμε μεταβλητες για να αποφευγουμε περιττες επαναληψεις και να το κανουμε πιο ευαναγνωστο. Οι ορισμοι των μεταβλητων ειναι της μορφης

```
<ονομα_μεταβλητης> = <κειμενο>
```

Αν υπαρχουν κενα στην αρχη του κειμενου αγνοουνται. Οι μεταβλητες μπορουν να χρησιμοποιηθουν οπουδηποτε (ακομη και για τον ορισμο αλλων μεταβλητων). Στην τιμη μιας μεταβλητης αναφερομαστε με τη συνταξη

```
$(<ονομα_μεταβλητης>) η ${<ονομα_μεταβλητης>}
```

Εστω για παραδειγμα οτι θελουμε να καλεσουμε τον μεταγλωττιστη με την επιλογη -g (για debugging). Μπορουμε στην αρχη του Makefile να ορισουμε :

```
CXXFLAGS = -g
```

και να τροποποιησουμε τους κανονες ως εξης

```
test_data_structures : list.o main.o
	g++ -o $(CXXFLAGS) test_data_structures list.o main.o

list.o : list.C list.h
	g++ -c ${CXXFLAGS} list.C

main.o : main.C
	g++ -c $(CXXFLAGS) main.C
```

3.4 Συντομεύσεις(macros)

Τα macros είναι παρόμοια με μεταβλητές και χρησιμεύουν στην αποθήκευση των ονομάτων των αρχείων. Η make μπορεί να χρησιμοποιήσει την τιμή ενός macro μέσω της συμβολοσειράς $(OBJECTS)

Για παράδειγμα ένα Makefile γραμμένο με χρήση macro:

```
OBJECTS = data.o main.o io.o
project1: $(OBJECTS)
	cc $(OBJECTS) -o project1
data.o: data.c data.h
	cc -c data.c
main.o: data.h io.h main.c
	cc -c main.c
io.o: io.h io.c
	cc -c io.c
```

Η εντολή make έχει κάποια προκαθορισμένα macros

- CC : Περιέχει τον τρέχον μεταγλωττιστή της C (εξ' ορισμού τιμή = cc)
- CFLAGS: Ειδικές επιλογές που προστίθενται στους κανόνες μεταγλώττισης
- $@ : Πλήρες όνομα του τρέχοντος αρχείου target
- $? : Λίστα των αρχείων της τρέχουσας εξάρτησης που πρέπει να ενημερωθούν
- (είναι νεότερα από το target)
- $^ : Τα πηγαία αρχεία της τρέχουσας εξάρτησης
- $< : Το πρώτο πηγαίο αρχείο της τρέχουσας εξάρτησης

Τα παραπάνω macro μπορούν να απλοποιήσουν το αρχείο Makefile

```
CC = gcc
CFLAGS= -Wall -O2
OBJECTS = data.o main.o io.o
project1: $(OBJECTS)
$(CC) $(CFLAGS) $(OBJECTS) -o project1
data.o: data.h
main.o: data.h io.h
io.o: io.h
```

3.5 Αποθήκευση & Εκτέλεση

Όταν δημιουργήσουμε το αρχείο το αποθηκεύουμε συνήθως ως "Makefile" ή "makefile" και για να το τρέξουμε εκτελούμε

```
$make all
```

με την προϋπόθεση ότι ο αρχικός(ριζικός) στόχος μας είναι ο 'all'.

Κατά την εκτέλεση της εντολής gcc/g++ filename.c/cpp απαιτούνται 3 βήματα:
1. **Μεταγλωττιστής:** Ο κώδικας του αρχείου .c/cpp μετατρέπεται σε κώδικα assembly με κατάληξη .s
2. **Assembler:** Ο κώδικας assembly μετατρέπεται σε object code δημιουργώντας αρχεία .o
3. **Linker:** Ο object code συνδέεται με τις αντίστοιχες βιβλιοθήκες, που περιέχουν διάφορες συναρτήσεις

4. Παράδειγμα

Το παρακάτω παράδειγμα είναι ένα ολοκληρωμένο Makefile. Αλλάξτε τα ονόματα των μεταβλητών με τα δικά σας η προσθέστε κώδικα και καλό compilation!

```
CC = gcc
CFLAGS = -Wall
OBJECTS = main.o ship.o ui.o ai.o

main: $(OBJECTS)
$(CC) $(CFLAGS) $(OBJECTS) -o main.exe

%.o : %.c
$(CC) $(CFLAGS) -c $<

clean:
	rm -f main.exe main.o ship.o ui.o ai.o
```

6. Σύνδεσμοι

Μπορείτε να βρείτε περισσότερες πληροφορίες για τα makefiles

- http://www.gnu.org/software/make/manual/html_node/index.html
- http://www.literateprogramming.com/indhill-cstyle.pdf

Το παραπάνω άρθρο γράφτηκε σύμφωνα με αρκετά tutorials που αναφέρονται παρακάτω

- http://www.cs.uoi.gr/~cs442/labs/misc/make-tool.html#rules
- http://mrbook.org/tutorials/make/
- https://docs.cs.byu.edu/wiki/Makefiles