ESD Electronic Systems Design

Synthesis Verification Testing Power Communication

Make (compiling), gdb (debugging), doxygen (documenting)

Dipartimento di Informatica

Università di Verona, Italy

Agenda

- Building
  - GNU Make
- Debugging
  - GNU Debugger
- Documenting
  - Doxygen

GNU Make

Compilazione

Make

- **Make** e' un utility largamente utilizzato in ambiente UNIX (e non solo) per compilare programmi con decine/centinaia di file sorgenti, Rapidamente!
- **Idea:** ricompiliamo solo i file sorgenti del programma che sono stati modificati
- **Vantaggio** (uno dei tanti), abbraviamo il tempo di compilazione dell'intero programma

Makefile

- **Makefile** e' uno script utilizzato per automatizzare la compilazione di un programma attraverso l'utility Make.

Un makefile contine **macro** e **rules**

Makefile - macro

Una **macro** in un makefile funziona come una variabile. E' defitita come: <tab><nome> = <valore>

- Esempi definizione:

```
OBJS = main.o io.o
CC = clang
```

- Esempio modificata:

```
OBJS += support.o
CC = gcc
```

- Esempio lettura:

```
@echo $(OBJS)     (stampa -> main.o io.o support.o)
@echo ${CC}       (stampa -> gcc)
```

Makefile - macro

Ci sono varie macro definite di default:

Nome	Descrizione	valore
AS	Compilatore assembly	as
CC	Compilatore c	cc
CXX	Compilatore c++	g++
RM	Programma per cancellare un file	rm -f
CFLAGS	Flags per compilatore c	Vuoto
CXXFLAG	Flags per compilatore c++	Vuoto
LD	Linker	ld

(make –p per avere la lista completa).

Makefile - rules

Una **rule** (regola) definisce come e *quando* ricompilare l'obiettivo della regola (rule's target). La regola definisce i requisiti del target, e la ricetta da usare per creare o aggiornare il target.

Definizione <target> : <requisiti>

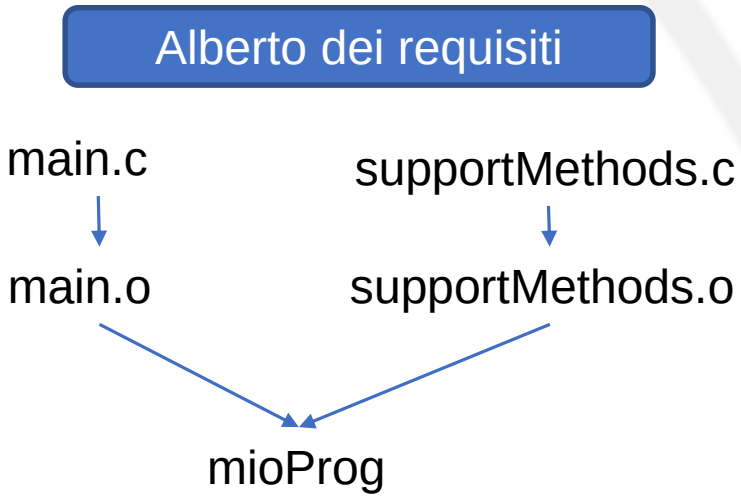
<TAB> ricetta (-> come ricompilare il target)

- Esempio definizione:

```
mioprogr: main.o io.o
        $(LD) $(LDFLAGS) –o mioprogr main.o io.o
```

Makefile

```
NAME=mioProgr
PROGRAM=$(NAME)

#object files
OBJS=main.o supportMethods.o

all: $(PROGRAM)

$(PROGRAM): $(OBJS)
    @echo Linking mioProgram
    $(LD) $(LDFLAGS) -o $(PROGRAM) $(OBJS)

main.o: ./src/main.c
    @echo Compiling main.o
    $(CC) $(CFLAGS) -o main.o ./src/main.c

supportMethods.o: ./src/supportMethods.c
    @echo Compiling supportMethods.o
    $(CC) $(CFLAGS) -o supportMethods.o \
                        ./src/supportMethods.c
```



Se modifico main.c, make ricompila solo main.c per rigenerare main.o e “ri-linkare” mioProg

Quando ricompilare un target

Un target è ricompilato se il data della sua ultima modifica è precendete a quella di almeno una delle sue dipendenze

Makefile – macro (continua)

```
NAME=mioProgr
PROGRAM=$(NAME)

#object files
OBJS=main.o supportMethods.o

all: $(PROGRAM)

$(PROGRAM): $(OBJS)
    @echo Linking $@
    $(LD) $(LDFLAGS) -o $@ $^

main.o: ./src/main.c
    @echo Compiling $@
    $(CC) $(CFLAGS) –o $@ $<

supportMethods.o: ./src/supportMethods.c
    @echo Compiling $@
    $(CC) $(CFLAGS) -o $@ $<
```

- Macro predefinite:
  - $@ il target
  - $< la prima dipendenza
  - $^ tutte le dipendenze
  - $? I file modificati
  - % il file corrente

Makefile – rules (continua)

```
NAME=mioProgr
PROGRAM=$(NAME)

#object files
OBJS=main.o supportMethods.o

all: $(PROGRAM)
...

clean:
    rm *.o

install:
    cp mioProg /usr/bin/

Doc:
```

- Primo target e' il target di default, eseguito quando make e' chiamato senza parametri
- Nomi tipici di target
  - all: compila e linka tutti i target
  - clean: cancella tutti i file generati
  - install: installa tutti i file generati
  - help: stampa una lista di target disponibili
  - doc: genera la documentazione
- .PHONY
  - Colleziona come dipendenze i target che **non** sono nomi di file generati (e.g., all, clean, help, etc..)

Makefile: esecuzione

- Eseguire make con Makefile, usando il target di default

  `make`

- Eseguire make in parallelo (opzionale: spcificare il numero massimo di thread in parallelo)

  `make -jN`

  - Massimo N thread in parallelo

  – E.g. per avere massimo 4 thread: `make -j4`

  – `make -j`

  - Nessun limite massimo di thread in parallelo

- Eseguire make con MyMake come file di configurazione:

  `make -f MyMake`

- Eseguire make per costruire un target specifico (e.g., app.x)

  `make app.x`

GNU Debugger

Debugging

Debugging

- Verifica e debugging
  - Una delle fasi più complesse durante lo sviluppo SW
  - L'attività che richiede più tempo (70-80%)
- Fondamentale usare strumenti per
  - Semplificare il debugging del codice
  - Velocizzare la correzione di bug
  - Evitare comportamenti strani dovuti a metodi bufferizzati nel caso di errari di segmentazione
- Un debugger permette:
  - Sospensione dell'esecuzione di un programma in un punto specifico
  - Eseguire un programma passo dopo passo
  - Ispezione i valori delle variabili a run-time

GNU Debugger

- GDB (The GNU Debugger)
  - Libero e open-source
  - Debugger standard usato con gcc
  - Richiede di arricchire l'eseguibile
    - gcc -g compila con informazioni di debug
    - gcc -ggdb come -g ma esplicitamente per gdb
  - Strumento da linea di comando
    - DDD è la più famosa interfaccia grafica
      - Non più mantenuta
    - cgdb è un'interfaccia testuale basata su *curses*

GDB: Comandi base

- Caricare un eseguibile

```
gdb my_exec.x
gdb --args  my_exec.x <program args>
```

- Chiudere gdb

```
quit / q
```

- Assegnare eventuali argomenti all'eseguibile

```
set args <arguments>
```

- Aprire l'help di un argomento (e.g., di un comando)

```
help <argomento>
```

GDB Breakpoints

- Breakpoint
  - Marcatura di un punto nel codice sorgente
  - L'esecuzione del programma viene sospesa quando viene raggiunto un breakpoint
  - Utile per ispezionare un punto del programma a run-time
  - Sintassi:

```
break / b <file:line>
break / b <methodName>
delete <numero> (rimuove il breakpoint #<numero>
```

GDB Espressioni

- Monitoraggio: Watch point
  - Specifica di un **espressione**
    - Non un punto nel codice
  - Utile per controllare l'evoluzione di una variabile
  - Sintassi:

```
watch <espressione>
```

- Valutazione: Print
  - Stampa il risultato di un espressione
    - Utilizza il valore corrente delle variabili
  - Utile per monitorare variabili
  - Sintassi:

```
print / p <espressione>
```

GDB: Ispezione dello Stack

- Backtrace
  - Stampa la sequenza di chiamate a funzione
  - Permette di capire l'ordine di esecuzione
  - Sintassi:

```
backtrace / bt
```

- Frame
  - Stampa informazioni sullo stack di metodi corrente
  - Sintassi:

```
frame / f                 descrizione corta
info frame / info f       descrizione lunga
```

  - Cambiare frame:

```
frame <numero> / f <numero>
```

GDB: Esecuzione

- Eseguire un programma dall'inizio:
  `run / r [<argomenti>]`
- Continuare l'esecuzione dal punto raggiunto
  `continue / c`
- Eseguire la prossima riga *atomicamente*
  `next / n`
- Eseguire la prossima istruzione
  - Eventualmente entrando in metodi/funzioni
    `step / s`
- Ripeti l'ultimo comando
  `\return`

Doxygen

Documentazione

Documentazione

- Tipi di documentazione
  - Per gli utenti
    - Manuali, guide, tutoria, etc…
    - Documentazione di API (nel caso di librerie)
  - Per gli sviluppatori
    - Documentazione di API
    - Documentazione dell'implementazione
    - Altro
      - Use cases, UML, specifiche, etc…

Documentazione del codice sorgente

- Commenti delle API
  - Generazione automatica della documentazione
  - Richiede l'utilizzo di un formato preciso
    - Linguaggi per la generazione della documentazione disponibili per ogni linguaggio
  - Obiettivo: generare la documentazione «ufficiale» del progetto
- Commenti dei dettagli implementativi
  - Immersi nel codice sorgente
  - Possono utilizzare semplice linguaggio naturale
  - Obiettivo: rendere il codice sorgente comprensibile a terze parti o «riletture» future

Doxygen

- Tool per la generazione automatica della documentazione di API
  - Multi-piattaforma
  - Libero ed open-source
- Supporta diversi linguaggi di programmazione
  - C, C++, Java
- Supporta diversi formati di output
  - ***HTML***, Latex, RTF

Usare Doxygen

- Generare un file di configurazione (Doxyfile)

```
Doxygen -g
```

- Configurare il Doxyfile

```
emacs Doxyfile
```

- Eseguire doxygen con il Doxyfile di default

```
Doxygen
```

- Eseguire doxygen con un file di configurazione specifico

```
doxygen <config file>
```

Configurazione di Doxygen

- La configurazione avviene **settando parametri nel Doxyfile**

- Parametri principali
  - PROJECT_NAME (e.g., MyLib)
    - Nome del progetto
  - OUTPUT_DIRECTORY (e.g. doc)
    - Directory dove salvare la documentazione
  - INPUT / FILE_PATTERNS (e.g. src)
    - Lista dei file (o directory) di ingresso
  - RECURSIVE (e.g. YES)
    - Se YES, allora visita le cartelle ricorsivamente
  - EXCLUDE / EXCLUDE_PATTERNS (*.java)
    - File da escludere
  - GENERATE_* (e.g. GENERATE_HTML)
    - Specifica del formato di uscita

Formato dei commenti

- I commenti in doxygen devono rispettare una forma particolare:

```
/** Commento parserizzato da Doxygen */
/// Commento parserizzato da Doxygen
```

  - Altri formati di commento vengono ignorati

```
/* Commento ignorato da Doxygen */
// Commento ignorato da Doxygen
```

- Informazioni speciali per la documentazione vengono date tramite *tag* speciali nella forma:

```
@tag
\tag
```

Tag principali di Doxygen

`@brief <commento>`

- Un breve commento sulla parte di codice seguente
- Utilizzato prima di blocchi di codice

`@param <nome parametro> <commento>`

- <commento> spiega il significato/utilizzo del parametro <nome parametro>
- Utilizzato per commentare dichiarazioni di metodi

`@return <comment>`

- Commenta il comportamento del valore di ritorno di un metodo
- Utilizzato per commentare dichiarazioni di metodi

`@throw <nome eccezione> <commento>`

- Commenta eccezioni lanciate da metodi (non per C)
- Utilizzato per commentare dichiarazioni di metodi

Tag di documentazione globale

- Informazioni sul contenuto di un file

```
/** @file
 * <comment >
 */
```

- Raggruppare metodi con una connessione logica

```
/** @name <groupName>*/
/*@{ */
<methodsWithTheirDocumentation>
/*@} */
```

Esempio

```
/** @name List accessors. */
/*@{ */

/** @brief Gets the element at given position.
 * Linear complexity.
 * @param l The list.
 * @param pos The position.
 * @return The stored element.
 */
void * getListElement( List * l, int pos );

/*@} */
```