Platformă de e-learning și curriculă e-content pentru învățământul superior tehnic

Utilizarea Sistemelor de Operare

28. Utilizarea Make pentru automatizarea compilării

Automatizarea sarcinilor - Makefile

- Sarcinile de compilare/linking/etc. (și nu numai) sunt de multe ori repetitive
- Folosirea de utilitare de automatizare
  - make, Apache Ant, Scons (dependențe)
  - shell scripting (fără dependențe)
- **make** folosește un fișier **Makefile**
- Structura tipică **Makefile**

```
# Comments use the hash symbol
target: dependencies
command 1
command 2
        .
        .
command n
```

Makefile simplu

```
$ cat Makefile
all: test

test: test.o
        gcc -o test test.o

test.o: test.c
        gcc -Wall -c -o test.o test.c

$ make
gcc -Wall -c -o test.o test.c
gcc -o test test.o

$ ./test
Hello, World!
```

Makefile simplu (2)

- La rularea **make** se caută prima regulă
  - se obișnuiește să fie all

1. all depinde de test
2. test depinde de test.o
3. test.o depinde de test.c
   - test.c există
   - se rulează comanda de compilare
   - se obține test.o
4. test.o a fost obținut
   - se rulează comanda de linking
   - se obține test
5. test este obținut; all este obținut

Makefile upgrade

```
$ cat Makefile1
CC = gcc
CFLAGS = -Wall

all: test

test: test.o
        $(CC) -o $@ $^

test.o: test.c
        $(CC) $(CFLAGS) -c -o $@ $<

clean:
        -rm -f *~ *.o test

$ make -f Makefile1 clean
rm -f *~ *.o test

$ make -f Makefile1
gcc -Wall -c -o test.o test.c
gcc -o test test.o
```

Makefile upgrade (2)

- Se definesc variabile
  - CC, CFLAGS
  - se referă cu $(nume_variabila) -> $(CC)

- Variabile automate
  - $@ -> ținta (target-ul)
  - $^ -> toate dependențele
  - $< -> prima dependență

- Regulă de ștergere: clean
  - se șterg fișiere obiect, temporare și executabilul

- Opțiunea -f specifică un fișier Makefile altul decât cel implicit (Makefile sau GNUMakefile)

UNIUNEA EUROPEANĂ GUVERNUL ROMÂNIEI Instrumente Structurale 2007-2013

Surse multiple

- Un program este constituit, de obicei, din mai multe fișiere sursă (module)
- Fiecare modul implementează o componentă a aplicației

- Exemplu: aplicație chat
  - un modul pentru funcții utile -> util.c
  - un header pentru antetele funcțiilor utile -> util.h
  - un modul pentru interfața cu utilizatorul -> ui.c
  - un modul pentru antetele funcțiilor de interfațare -> ui.h
  - un modul care asigură comunicarea în rețea -> net.c
  - un header pentru antete -> net.h
  - un modul de jurnalizare -> log.c
  - un header petnru jurnalizare -> log.h
  - un modul care integrează celelalte module -> app.c

Cum se scrie un header

```c
#include <stdlib.h>

#include "util.h"

int copy_to_list (char *msg, size_t
   msg_len, struct list *list)
{
   [...]
}

int find_in_list (int id, struct list
   *list)
{
   [...]
}
```

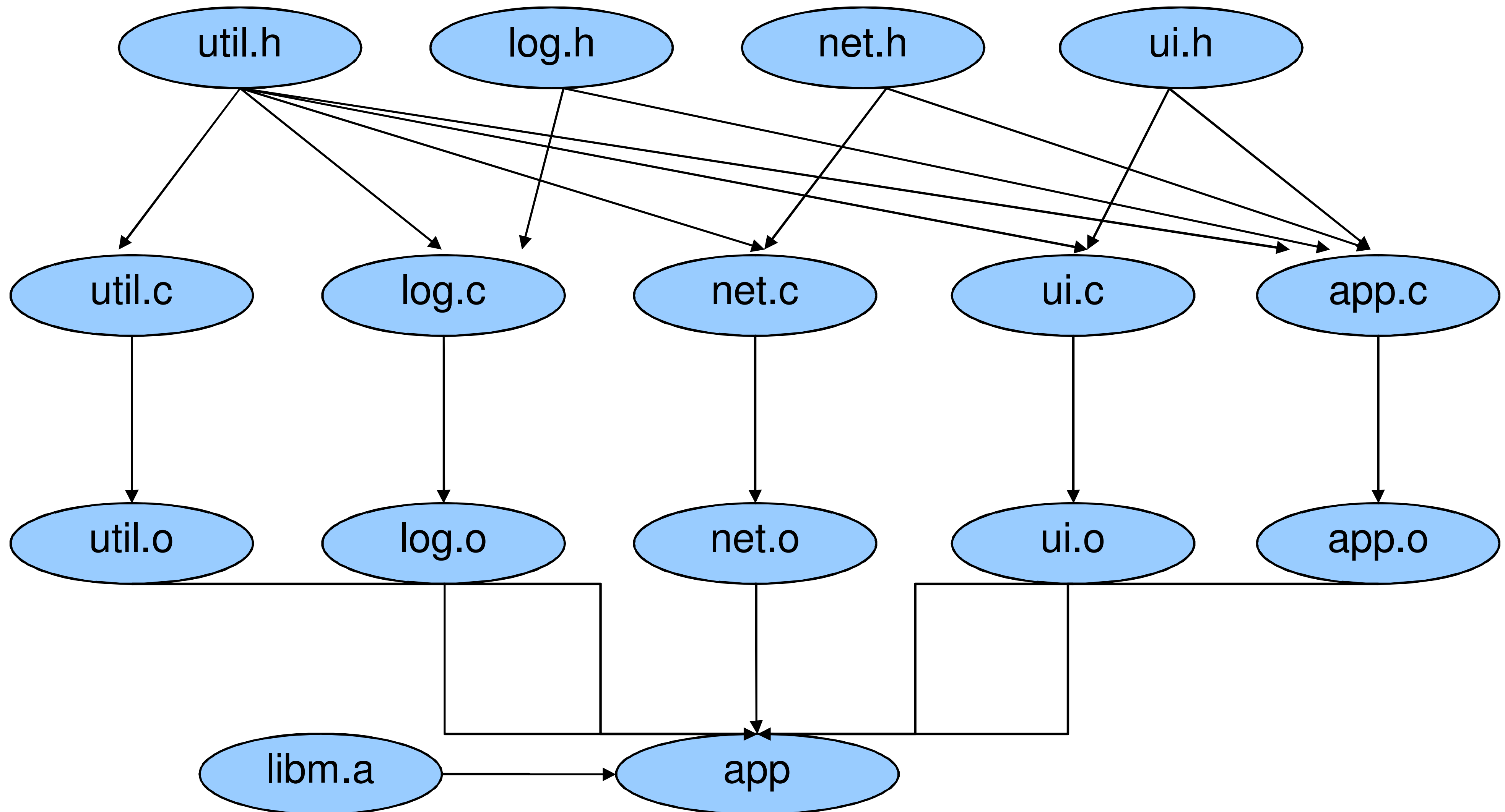
```c
#ifndef _UTIL_H
#define _UTIL_H         1

struct list {
        int id;
        void *generic_data;
        struct list *next;
        struct list *prev;
};

int copy_to_list (char *msg, size_t
   msg_len, struct list *list);
int find_in_list (int id, struct
   list *list);

#endif
```

Dependențe



Makefile final

```
$ cat Makefile.prj
# -g -> compilare cu simboluri de debug
CFLAGS = -Wall -g
LDLIBS = -lm

all: app

app: app.o ui.o log.o util.o net.o

app.o: app.c util.h log.h ui.h net.h

ui.o: ui.c ui.h util.h

log.o: log.c log.h util.h

util.o: util.c util.h

net.o: net.c util.h

clean:
        -rm -f *~ *.o app
```

```
$ make -f Makefile.prj
cc -Wall -g         -c -o app.o app.c
cc -Wall -g         -c -o ui.o ui.c
cc -Wall -g         -c -o log.o log.c
cc -Wall -g         -c -o util.o util.c
cc -Wall -g         -c -o net.o net.c
cc   app.o ui.o log.o util.o net.o  -lm -o
app
```