Instrukcja do zajęć laboratoryjnych

Język ANSI C (w systemie LINUX)

wersja: 1.0

Nr ćwiczenia:	dodatkowe 2	
Temat:	**Automatyczne tworzenie skryptu konfiguracyjnego (plik *configure*) z wykorzystaniem programów automake oraz autoconf. Omówienie programu install.**	
Cel ćwiczenia:		
Wymagane przygotowanie teoretyczne:	Informacje podane na wykładzie. Podstawy pracy z systemem LINUX.	
Sposób zaliczenia:	Sprawozdanie w formie pisemnej.	[ ]
	Pozytywna ocena ćwiczenia przez prowadzącego pod koniec zajęć.	[ ]

1. Konwencje przyjęte w instrukcji

`Czcionka o stałej szerokości`

Nazwy programów, poleceń, katalogów, wyniki działania wydawanych poleceń.

`Czcionka o stałej szerokości pogrubiona`

Podaje tekst, który należy dosłownie przepisać. W przypadku plików źródłowych wyróżnia istotniejsze fragmenty.

`Czcionka o stałej szerokości kursywą`

Tekst komentarza w przykładowych sesjach przy terminalu.

`Czcionka o stałej szerokości kursywą pogrubiona`

Wyróżnia istotniejsze fragmenty wyników działania wydawanych poleceń.

2. Uwagi wstępne

Po napisaniu i przetestowaniu jakiegokolwiek użytkowego programu, należy go w wygodny sposób dostarczyć użytkownikowi. Gdy jest on rozprowadzany w postaci plików źródłowych (a środowiska UNIX-owe bardzo „lubią” taką formę dystrybucji), trzeba to zrobić w taki sposób, aby odbiorca

tych plików był w stanie samodzielnie poradzić sobie z kompilacją, często przecież bardzo złożonego, programu. A kompilowanie plików źródłowych złożonych programów nie jest z reguły sprawą banalną! Gdyby nie istniały narzędzia wspomagające ten proces, byłoby to zadanie przerastające możliwości niejednego użytkownika.

W przypadku prostszych aplikacji w zupełności wystarczy przygotowanie odpowiedniego pliku `Makefile`. Bardziej złożone projekty wymagają natomiast umieszczania w plikach `Makefile` różnego rodzaju opcji konfiguracyjnych, ścieżek dostępu do bibliotek i innych elementów, bardzo ściśle zależnych od konfiguracji konkretnego komputera. Pliki `Makefile` muszą więc być tworzone indywidualnie dla każdego użytkownika. Innymi słowy muszą one być w dużym stopniu „spersonalizowane". Osiągamy to generując je za każdym razem, gdy rozpoczynamy kompilowanie projektu. Dzięki tworzeniu ich „w locie", są one w wymaganym stopniu dostosowane do konkretnej konfiguracji danego komputera. W praktyce sprowadza się to do uruchomienia specjalnego skryptu, zwykle o nazwie `configure`. Tworzenie tego skryptu wydatnie wspomagają dwa programy: `automake` oraz `autoconf`, które są tematem niniejszego ćwiczenia.

Kompilowanie oraz instalowanie programów zazwyczaj odbywa się poprzez wydanie następującej sekwencji trzech poleceń (tak, najczęściej po skompilowaniu projektu trzeba go zainstalować, czyli wgrać w odpowiednie miejsca utworzone w czasie kompilacji pliki):

```
./configure (to polecenie często wydawane jest z różnymi opcjami)
make
make install
```

Wydanie tych trzech z pozoru prostych poleceń uruchamia całą wielką machinę konfiguracyjną. Zadaniem skryptu `configure` jest właśnie utworzenie wszystkich „spersonalizowanych", wymaganych do późniejszej kompilacji, plików `Makefile`. Polecenie `make`, na bazie tych właśnie plików `Makefile`, skompiluje wszystkie wymagane pliki źródłowe, natomiast polecenie `make install` umieści w odpowiednich katalogach utworzone w czasie komplikacji pliki (czyli *de facto* zainstaluje program w systemie operacyjnym).

Skrypt `configure` tworzony jest automatycznie za pomocą programu `autoconf`. Autor umieszcza w pliku o nazwie `configure.in` odpowiednie makropolecenia, które następnie są czytane przez `autoconf` i na ich podstawie generowany jest skrypt `configure` (informacje o ew. błędach są umieszczane w pliku `config.log`).

Skrypt `configure` może przyjmować różne opcje pozwalające na dostosowanie procesu kompilacji do własnych potrzeb. Najczęściej używane to:

- `./configure --prefix=/drzewo/katalogów` – pozwala na określenie drzewa katalogów „pod" którym instalowane będą pliki po wydaniu polecenia `make install`. Najczęściej programy instalowane są w drzewie `/usr/local`. Ta opcja pozwala zmienić domyślne położenie plików i tym samym czuwać nad zachowaniem porządku w systemie,

- `./configure --enable-funkcja` lub `--with-pakiet` – pozwala na włączenie standardowo wyłączonych funkcji lub przystosowanie programu do współpracy z określonym pakietem.

Listę dostępnych opcji można zwykle wyświetlić wydając polecenie `./configure --help`.

Oprócz standardowo używanych opcji (takich jak np. te opisane powyżej) istnieje oczywiście możliwość tworzenia własnych, specyficznych tylko dla naszego programu opcji. Wymaga to jednak dość wnikliwego zapoznania się z programem `autoconf`.

3. Przykład użycia programów automake i autoconf

Aby prawidłowo wykorzystać siłę programów `automake` oraz `autoconf` musimy utworzyć, oprócz oczywiście plików źródłowych naszego programu, pliki o nazwach `Makefile.am` oraz `configure.in`. Niech zawartość tych plików oraz plików źródłowych programu będzie następująca:

```
/* hello.h */

#define HELLO_NAME "Nazwa aplikacji: HELLO"
#define HELLO_VERSION "Wersja: 1.0"
#define HELLO_AUTHOR "Autor: Artur Gramacki"
```

```
/* hello.c */

#include<stdio.h>
#include "hello.h"

int main(void)
{
  printf("Hello, world !\n\n");
  printf("%s\n%s\n%s\n\n", HELLO_NAME, HELLO_VERSION, HELLO_AUTHOR);
  return(0);
}
```

```
# Makefile.am

bin_PROGRAMS = hello
hello_SOURCES = hello.c hello.h
```

```
# configure.in

AC_INIT(hello.c)
AM_INIT_AUTOMAKE(hello, 1.0)
AC_PROG_CC
AC_PROG_INSTALL
AC_OUTPUT(Makefile)
```

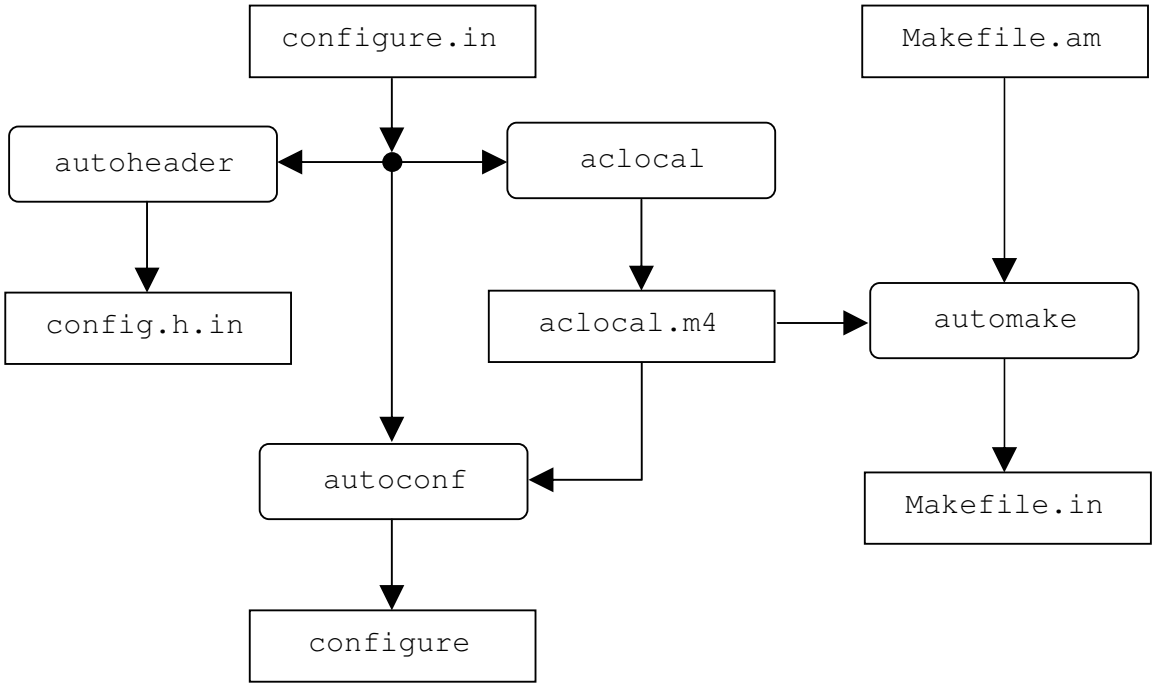
Na bazie pliku `configure.in`, za pomocą polecenia `alclocal`, zostanie utworzony plik `aclocal.m4`. Następnie polecenie `autoconf` utworzy skrypt `configure`. Na końcu polecenie `automake` (wydane z parametrem `-a` o czym poniżej), na bazie pliku `Makefile.am`, otworzy plik `Makefile.in`. Teraz można już wykonać skrypt `configure`, który utworzy wymagany plik (lub pliki) `Makefile`. W końcu możemy wydać polecenie `make` oraz `make install`, które odpowiednio skompilują i zainstalują nasz program.

Utworzony plik `Makefile` jest dość obszerny. Zawiera on wiele automatycznie wygenerowanych reguł, takich jak np. `make all`, `make install`, `make uninstall`. Pamiętajmy, że gdy polecenie `make install` wydamy bez żadnych dodatkowych parametrów, aplikacja zostanie zainstalowana w drzewie katalogów `/usr/local` (oczywiście instalacja powiedzie się tylko wówczas, gdy posiadamy odpowiednie uprawnienia do tego katalogu).

Wydanie polecenia `automake` z opcją `-a` powoduje, że `automake` sprawdza istnienie niektórych standardowych plików, które są wymagane przez standard GNU i powinny być obecne w katalogu bieżącym. Część z nich zostanie utworzona automatycznie, część trzeba utworzyć ręcznie, np.

wydając polecenie: `touch NEWS README AUTHORS ChangeLog`. Gdy polecenie `automake -a` nie wygeneruje żadnych komunikatów o błędach, oznacza to, że szczęśliwie dobrnęliśmy do końca.

Opisane powyżej etapy zobrazowano na poniższym rysunku. W prostokątach o ostrych rogach umieszczona nazwy plików a w prostokątach o zaokrąglonych rogach nazwy programów (poleceń). Polecenie `autoheader` oraz plik `config.h.in` zostaną omówione w dalszej części instrukcji.



Poniżej pokazano przykładową sesję przy terminalu:

```
Na początku na dysku mamy 4 pliki – 2 źródłowe i 2 dla programów automake
i autoconf. Przełącznik -F w graficzny sposób obrazuje typ pliku –
gwiazdka po nazwie oznacza, że jest to plik wykonywalny, znak @ oznacz, że
jest to dowiązanie symboliczne itd.(patrz dokumentacja).
$ ls -F
configure.in  hello.c  hello.h  Makefile.am
$

Po wykonaniu polecenia aclocal pojawia się plik aclocal.m4
$ aclocal
$ ls -F
aclocal.m4  configure.in  hello.c  hello.h  Makefile.am
$

Po wykonaniu polecenia autoconf pojawia się skrypt configure
$ autoconf
$ ls -F
aclocal.m4  configure*  configure.in  hello.c  hello.h  Makefile.am
$

Po wykonaniu polecenia automake pojawia się plik Makefile.in. Polecenie
generuje ostrzegawcze komunikaty o nieistnieniu pewnych wymaganych
standardem GNU plikach.
$ automake
automake: configure.in: required file `./install-sh' not found
automake: configure.in: required file `./mkinstalldirs' not found
```

```
automake: configure.in: required file `./missing' not found
automake: Makefile.am: required file `./INSTALL' not found
automake: Makefile.am: required file `./NEWS' not found
automake: Makefile.am: required file `./README' not found
automake: Makefile.am: required file `./COPYING' not found
automake: Makefile.am: required file `./AUTHORS' not found
automake: Makefile.am: required file `./ChangeLog' not found
automake: configure.in: required file `./depcomp' not found
$

$ ls -F
aclocal.m4  configure.in  hello.h      Makefile.in
configure*  hello.c       Makefile.am
$

Polecenia automake wydane z parametrem -a powoduje, że część wymaganych
plików jest tworzona automatycznie a część nie.
$ automake -a
automake: configure.in: installing `./install-sh'
automake: configure.in: installing `./mkinstalldirs'
automake: configure.in: installing `./missing'
automake: Makefile.am: installing `./INSTALL'
automake: Makefile.am: required file `./NEWS' not found
automake: Makefile.am: required file `./README' not found
automake: Makefile.am: installing `./COPYING'
automake: Makefile.am: required file `./AUTHORS' not found
automake: Makefile.am: required file `./ChangeLog' not found
automake: configure.in: installing `./depcomp'
$

$ ls -F
aclocal.m4    COPYING@  hello.h      Makefile.am  mkinstalldirs@
configure*    depcomp@  INSTALL@     Makefile.in
configure.in  hello.c   install-sh@  missing@
$

Tworzymy brakujące pliki
$ touch NEWS README AUTHORS ChangeLog
$

Ponownie uruchamiamy program automake. Tym razem kończy on swoje działanie
bez żadnych komunikatów ostrzegawczych.
$ automake -a
$ ls -F
aclocal.m4  configure*    depcomp@  INSTALL@     Makefile.in     NEWS
AUTHORS     configure.in  hello.c   install-sh@  missing@        README
ChangeLog   COPYING@      hello.h   Makefile.am  mkinstalldirs@
$

Dokładniej zapoznajmy się z finalną zawartością katalogu. Zwróćmy uwagę,
że niektóre pliki są dowiązaniami symbolicznymi.
$ ls -lF
total 104
-rw-r--r--    1 artur    students    16626 Mar 18 12:59 aclocal.m4
-rw-r--r--    1 artur    students        0 Mar 18 13:02 AUTHORS
-rw-r--r--    1 artur    students        0 Mar 18 13:02 ChangeLog
-rwxr-xr-x    1 artur    students    50236 Mar 18 13:00 configure*
-rw-r--r--    1 artur    students      110 Mar 18 10:41 configure.in
lrwxrwxrwx    1 artur    students       27 Mar 18 13:02 COPYING ->
/usr/share/automake/COPYING
lrwxrwxrwx    1 artur    students       27 Mar 18 13:02 depcomp ->
/usr/share/automake/depcomp*
-rw-r--r--    1 artur    students      162 Mar 18 10:35 hello.c
```

```
-rw-r--r--    1 artur    students      137 Mar 18 09:41 hello.h
lrwxrwxrwx    1 artur    students       27 Mar 18 13:02 INSTALL ->
/usr/share/automake/INSTALL
lrwxrwxrwx    1 artur    students       30 Mar 18 13:02 install-sh ->
/usr/share/automake/install-sh*
-rw-r--r--    1 artur    students       68 Mar 18 10:41 Makefile.am
-rw-r--r--    1 artur    students    10878 Mar 18 13:03 Makefile.in
lrwxrwxrwx    1 artur    students       27 Mar 18 13:02 missing ->
/usr/share/automake/missing*
lrwxrwxrwx    1 artur    students       33 Mar 18 13:02 mkinstalldirs ->
/usr/share/automake/mkinstalldirs*
-rw-r--r--    1 artur    students        0 Mar 18 13:02 NEWS
-rw-r--r--    1 artur    students        0 Mar 18 13:02 README
```

Gdy pliki konfiguracyjne zostały już wygenerowane pora przejść do kompilacji programu:

```
Pora więc uruchomić skrypt konfiguracyjny. Dodatkowo specyfikujemy opcję –
prefix, aby wskazać katalog, gdzie zostanie zainstalowany program. Po
wykonaniu testów diagnostycznych tworzony jest spersonalizowany plik
Makefile.
$ ./configure --prefix=$PWD
loading cache ./config.cache
checking for a BSD compatible install... (cached) /usr/bin/install -c
checking whether build environment is sane... yes
checking for mawk... (cached) mawk
checking whether make sets ${MAKE}... (cached) yes
checking for gcc... (cached) gcc
checking whether the C compiler (gcc  ) works... yes
checking whether the C compiler (gcc  ) is a cross-compiler... no
checking whether we are using GNU C... (cached) yes
checking whether gcc accepts -g... (cached) yes
checking for style of include used by make... GNU
checking dependency style of gcc... (cached) gcc
checking for a BSD compatible install... /usr/bin/install -c
creating ./config.status
creating Makefile
$

Wykonujemy kompilację. Widzimy, że podczas kompilacji do gcc dodawanych
jest wiele opcji –D definiujących różne makra języka C. Przy większych
projektach liczba takich parametrów może być tak duża, że nie będą się
mieściły w dopuszczalnym rozmiarze ciągu parametrów powłoki lub
kompilatora. Rozwiązanie tego problemu pokażemy poniżej.
$ make
source='hello.c' object='hello.o' libtool=no \
depfile='.deps/hello.Po' tmpdepfile='.deps/hello.TPo' \
depmode=gcc /bin/sh ./depcomp \
gcc -DPACKAGE=\"hello\" -DVERSION=\"1.0\"  -I. -I.     -g -O2 -c `test -f
hello.c || echo './'`hello.c
gcc  -g -O2   -o hello  hello.o
$

Instalujemy naszą aplikację. Ponieważ wydaliśmy wcześniej polecenie
./configure --prefix=$PWD, aplikacja (u nas jeden plik wykonywalny o
nazwie 'hello') zostanie zainstalowana w katalogu bieżącym, w podkatalogu
bin.
$ make install
make[1]: Entering directory `/home/artur/c/auto_make_conf'
/bin/sh ./mkinstalldirs /home/artur/c/auto_make_conf/bin
mkdir /home/artur/c/auto_make_conf/bin
  /usr/bin/install -c hello /home/artur/c/auto_make_conf/bin/hello
```

```
make[1]: Nothing to be done for `install-data-am'.
make[1]: Leaving directory `/home/artur/c/auto_make_conf'
$

$ cd bin
$ ls -F
hello*
$
```

Powyżej wspomniano o pewnej uciążliwości w czasie kompilowania programów (duża liczba opcji `-D`). Problem ten rozwiązujemy w taki sposób, że do pliku `configure.in` dodajemy wywołanie jeszcze jednego makra:

```
AM_CONFIG_HEADER(config.h)
```

które powoduje, że zamiast dodawania dużej ilości opcji `-D` w czasie kompilacji `gcc`, zostanie utworzony plik `config.h` zawierający odpowiednie definicje makr i nazw symbolicznych. Wydanie wówczas polecenie `autoheader` utworzy ten plik. Do kompilatora dodana zostanie jedynie opcja `DHAVE_CONFIG_H`, którą należy wykorzystać przy dołączaniu pliku `config.h` w plikach źródłowych. W pliku c należy więc dodać następujący fragment:

```
#ifdef HAVE_CONFIG_H
#include <config.h>
#endif
```

W dyrektywie `#include` zostały użyte znaki `< >` a nie `""`, ze względu na to, że pliki źródłowe niekoniecznie muszą się znajdować w tym samym katalogu co plik `config.h`. Całość sprowadza się teraz do wydania czterech poleceń:

```
aclocal
autoconf
autoheader
automake -a
```

Zobaczy jak powyższe działa w praktyce:

```
Pierwsze dwa polecenia nie wymagają już komentarza.
$ aclocal
$ autoconf
$

Polecenie autoheader tworzy, na bazie makra w pliku configure.in, plik
config.h.in.
$ autoheader
$ ls -F
Makefile.am  config.h.in  configure.in  hello.h
aclocal.m4   configure*   hello.c
$

Oglądamy zawartość pliku.
$ cat config.h.in
/* config.h.in.  Generated automatically from configure.in by autoheader
2.13.  */

/* Name of package */
#undef PACKAGE

/* Version number of package */
#undef VERSION

$ automake -a
automake: configure.in: installing `./install-sh'
```

```
automake: configure.in: installing `./mkinstalldirs'
automake: configure.in: installing `./missing'
automake: Makefile.am: installing `./INSTALL'
automake: Makefile.am: installing `./COPYING'
automake: configure.in: installing `./depcomp'
$
```

W porównaniu do poprzedniego przypadku tworzony jest dodatkowo plik config.in

```
$ ./configure
creating cache ./config.cache
checking for a BSD compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for mawk... mawk
checking whether make sets ${MAKE}... yes
checking for gcc... gcc
checking whether the C compiler (gcc  ) works... yes
checking whether the C compiler (gcc  ) is a cross-compiler... no
checking whether we are using GNU C... yes
checking whether gcc accepts -g... yes
checking for style of include used by make... GNU
checking dependency style of gcc... gcc
checking for a BSD compatible install... /usr/bin/install -c
updating cache ./config.cache
creating ./config.status
creating Makefile
creating config.h
$
```

```
$ ls -F
AUTHORS    Makefile.am  config.cache    configure*    install-sh@
stamp-h1
COPYING@   Makefile.in  config.h        configure.in  missing@
ChangeLog  NEWS         config.h.in     depcomp@      mkinstalldirs@
INSTALL@   README       config.log      hello.c       stamp-h
Makefile   aclocal.m4   config.status*  hello.h       stamp-h.in
$
```

Oglądamy zawartość pliku config.h

```
$ cat config.h
/* config.h.  Generated automatically by configure.  */
/* config.h.in.  Generated automatically from configure.in by autoheader
2.13.  */

/* Name of package */
#define PACKAGE "hello"

/* Version number of package */
#define VERSION "1.0"
```

Tym razem widzimy, że zamiast wielu opcji -D dodane jest tylko -DHAVE_CONFIG_H

```
$ make
make  all-am
make[1]: Entering directory `/home/artur/c/auto_make_conf'
source='hello.c' object='hello.o' libtool=no \
depfile='.deps/hello.Po' tmpdepfile='.deps/hello.TPo' \
depmode=gcc /bin/sh ./depcomp \
gcc -DHAVE_CONFIG_H -I. -I. -I.     -g -O2 -c `test -f hello.c || echo
'./'`hello.c
gcc  -g -O2   -o hello  hello.o
make[1]: Leaving directory `/home/artur/c/auto_make_conf'
$
```

4. Przykład użycia programu install

Program `install` posiada funkcjonalność programu `cp` oraz dodatkowo umożliwia ustawianie praw dostępu do kopiowanych plików, a jeżeli pozwalają na to uprawnienia użytkownika, umożliwia również ustawianie właściciel i grupy. Z pomocą tego program tego programu możliwe jest również tworzenie katalogów.

Program ten jest bardzo często używany w plikach `Makefile` do automatyzowania różnych czynności związanych z instalowaniem skompilowanych wcześniej programów (najczęściej używamy go w czasie wykonywania polecenia `make install`). Nic nie stoi jednak na przeszkodzie, aby programu tego używać zupełnie niezależnie.

Dokumentacja `man install` podaje szczegóły użycia programu. Istnieją trzy różne formaty, które zostaną poniżej omówione.

```
       install [OPTION]... SOURCE DEST           (1st format)
       install [OPTION]... SOURCE... DIRECTORY   (2nd format)
       install -d [OPTION]... DIRECTORY...       (3rd format)
```

`SOURCE` jest nazwą jednego lub kilku plików do skopiowania a `DEST` jest nazwą pliku docelowego albo, gdy jako `SOURCE` określono kilka plików, katalogiem. Gdy używamy opcji `-d` program `install` umożliwia nam tworzenie katalogów. Zobaczmy jak `install` działa w praktyce:

```
                        ----- PIERWSZY FROMAT -----

Jesteśmy w katalogu bieżącym. Jest w nim plik mbox. Przekopijemy go do
podkatalogu temp.
~$ ls -l mbox
-rw-------    1 artur    students     2971 Mar 11 13:16 mbox
~$

Kopiujemy plik mbox do katalogo temp.
~$ install mbox temp/mbox_kopia
~$

Okazuje się, że przekopiowany plik otrzymał uprawnienia 0755. Gdy jawnie
nie podany uprawnień, takie będą przyjęte domyślnie.
~/$ ls -l temp/mbox_kopia
-rwxr-xr-x    1 artur    students     2971 Mar 16 09:16 mbox_kopia
~/$

~$ rm temp/mbox_kopia
~$ install -m 644 mbox temp/mbox_kopia
~$ ls -l temp/mbox_kopia
Tym razem przekopiowany plik otrzymał jawnie podane uprawnienia.
-rw-r--r--    1 artur    students     2971 Mar 16 09:25 mbox_kopia
~$

Próba skopiowania pliku oraz ustawienia jego właściciela (-o) oraz grupy
(-g). Operacja nie udała się, gdyż nie posiadamy wystarczających
uprawnień.
~$ install -m 644 -o root -g root mbox temp/mbox_kopia2
install: cannot change ownership of `temp/mbox_kopia2': Operation not
permitted
~$

Przełączenie się na użytkownika root (uwaga: tą czynność możesz
przećwiczyć tylko na własnym serwerze, gdzie masz uprawnienia root). Na
laboratorium tylko read-only!
~$ su
```

```
Password:
/home/artur#  install -m 644 -o root -g root mbox temp/mbox_kopia2
/home/artur# ls -l temp/mbox_kopia*
Stwierdzamy, że przekopiowany plik zmienił zmieniła (tak jak chcieliśmy)
właściciela oraz grupę.
-rw-r--r--    1 artur    students     2971 Mar 16 09:29 temp/mbox_kopia
-rw-r--r--    1 root     root         2971 Mar 16 09:34 temp/mbox_kopia2
/home/artur#
```

```
                        ----- DRUGI FROMAT -----

Tworzymy trzy pliki
~$ touch plik1 plik2 plik3
~$ ls -l plik*
-rw-r--r--    1 artur    students        0 Mar 16 10:09 plik1
-rw-r--r--    1 artur    students        0 Mar 16 10:09 plik2
-rw-r--r--    1 artur    students        0 Mar 16 10:09 plik3
~$

Ponieważ jako SOURCE podano więcej niż jeden plik, DEST jest
interpretowane jako katalog i tam kopiowane są wskazane pliki.
~$ install -m 644 plik1 plik2 plik3 temp/
~$ ls -l temp/
total 0
-rw-r--r--    1 artur    students        0 Mar 16 10:13 plik1
-rw-r--r--    1 artur    students        0 Mar 16 10:13 plik2
-rw-r--r--    1 artur    students        0 Mar 16 10:13 plik3
~$
```

```
                        ----- TRZECI FROMAT -----

Jesteśmy w katalogu bieżącym. W podkatalogu temp utworzymy dwa dodatkowe
podkatalogi. Zwróćmy uwagę na ustawione prawa dostępów.
~$ install -d temp/kat1
~$
~$ install -m 700 -d temp/kat2
~$
~$ ls -la temp/
total 16
drwxr-xr-x    4 artur    students     4096 Mar 16 11:42 .
drwxr-xr-x   18 artur    students     4096 Mar 16 10:09 ..
drwxr-xr-x    2 artur    students     4096 Mar 16 11:41 kat1
drwx------    2 artur    students     4096 Mar 16 11:42 kat2
-rw-r--r--    1 artur    students        0 Mar 16 10:13 plik1
-rw-r--r--    1 artur    students        0 Mar 16 10:13 plik2
-rw-r--r--    1 artur    students        0 Mar 16 10:13 plik3
~$
```

5. Podsumowanie

Programy `automake` oraz `autoconf` są to bardzo zaawansowane technologicznie i złożone narzędzia, które wspomagają proces tworzenia złożonych systemów informatycznych. Zaprezentowano je na przykładzie kompilacji bardzo prostego programiku typu „Hello, world". Oczywiście w tym przypadku użycie programów `automake` oraz `autoconf` było całkowicie zbędne i na wyrost, gdyż tak prosty program w zupełności obywa się bez nich (wystarczy utworzyć prosty pliki `Makefile` i ... już). Naszym celem było jednak pokazanie zasad pracy z programami `automake`

oraz `autoconf` i dlatego też posłużenie się tak banalnym programem źródłowym jest w pełni uzasadnione.

Okazuje się, że programiści z całego świata, tworząc złożone programy, powszechnie używają `automake` oraz `autoconf`e. Można wręcz stwierdzić, że stały się one *de facto* standardem wśród programistów i z pewnością przyczyniły się do tego, że aplikacje rozprowadzane w postaci plików źródłowych „trafiły pod strzechy". Kapitalne znaczenie ma tutaj fakt, że proces kompilowania i instalowania programów został bardzo mocno zunifikowany i w praktyce sprowadza się do wydania kilku bardzo prostych poleceń (`./configure`, `make`, `make install` – ew. z odpowiednimi opcjami).

Literatura

1. Dokumentacja systemowa `man` oraz `info`
2. Dokumentacja `autoconf` i `automake`: http://www.gnu.org/manual/
3. http://www.airs.com/ian/configure
4. A. Podstawczyński, *Linux praktyczne rozwiązania*, Helion 2000
5. http://adsystems.com.pl/autotools-tutorial-pl.html