
AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

# Narzędzia binutils (i inne)

**Maciej Siczek**
**Przemysław Plutecki**

**Wydział Fizyki i Informatyki Stosowanej**

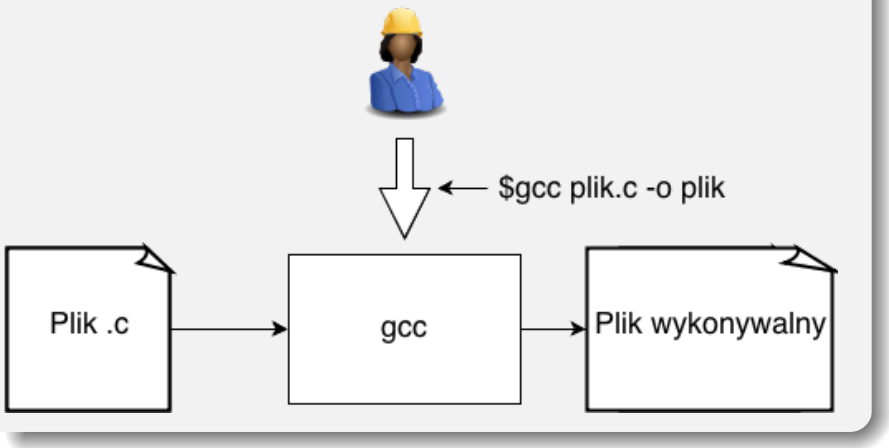
**03.04.2013**

# Plan prezentacji

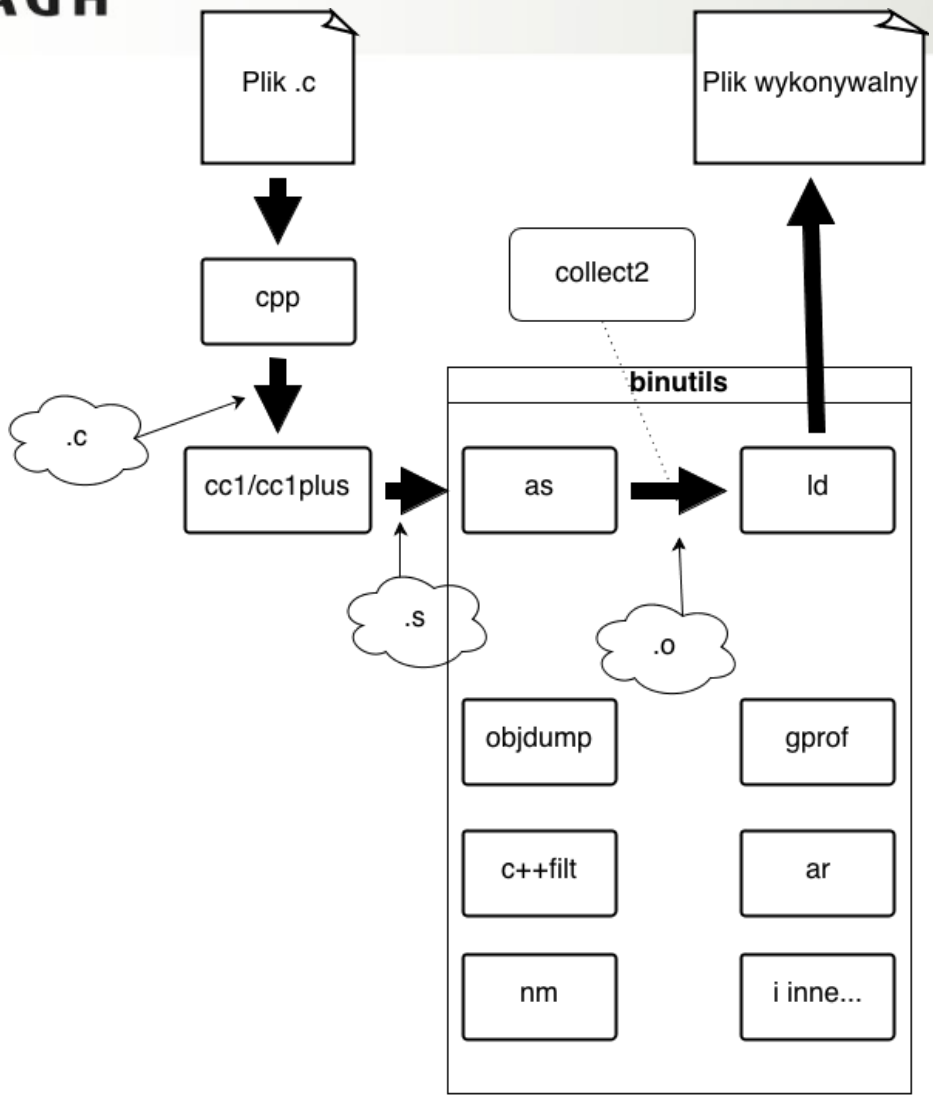
1. Synteza
   - Wstęp
   - ELF
   - cpp
   - gcc
   - as
   - ld
2. Analiza
   - GDB
   - DDD
   - objdump
   - strings
   - gprof
   - gcov

# Wstęp

## Kiedyś




# binutils - gcc backend




*gcc (g++) — umbrella (wrapper)*

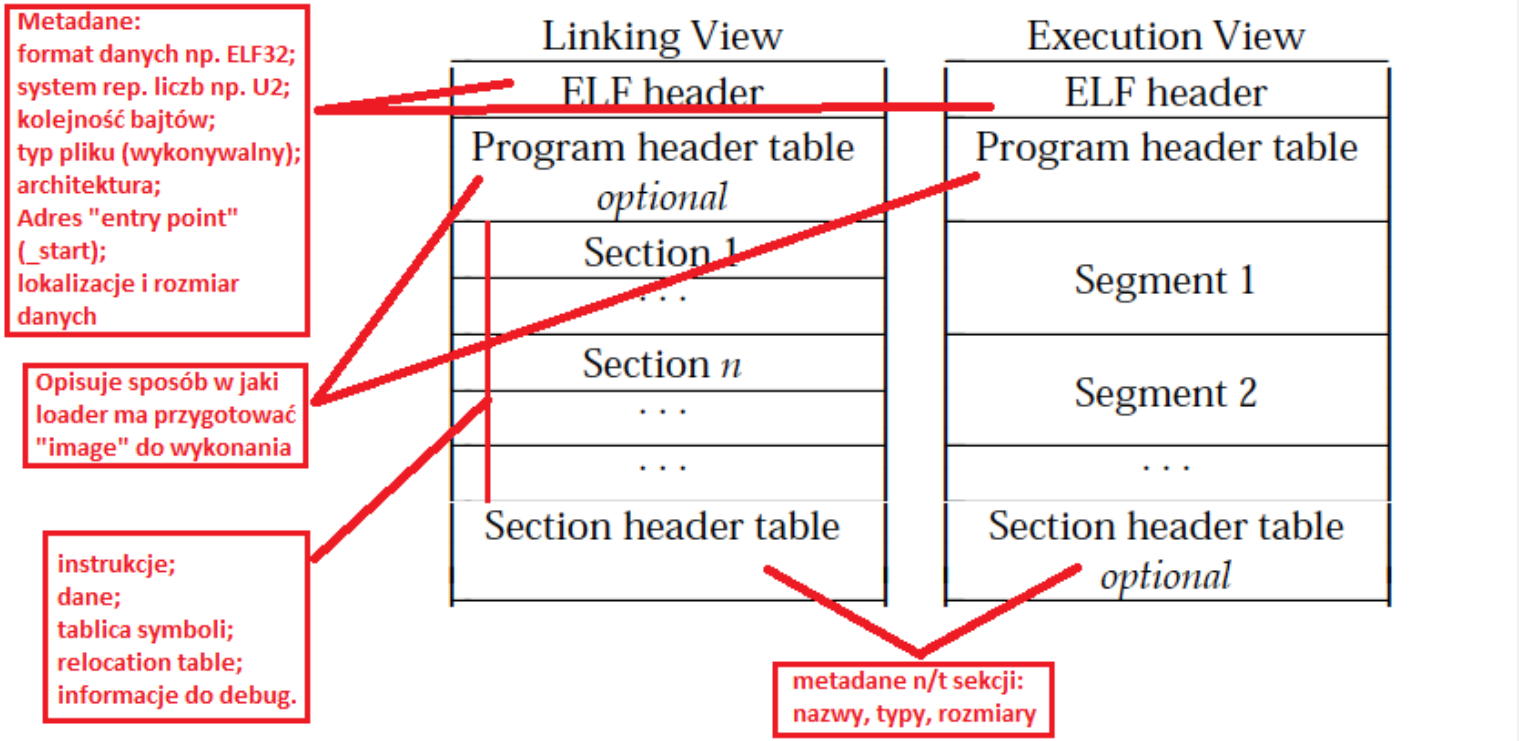
# Format plików binarnych

## ELF (Executable Linkable Format)

- opracowany przez Unix System Laboratories, wywodzi się z System V; „nazwany“ standardem w 1999r.
- standard, który obowiązuje w GNU/Linux, FreeBSD, wielu innych uniksowych systemach operacyjnych, a także w konsolach gier (PSP, PS2, PS3), telefonach (Symbian OS, Bada, ...), ...
- określa format plików wykonywalnych, obiektowych, bibliotek współdzielonych i zrzutów pamięci
- nie jest związany z konkretnym procesorem, ani architekturą
- pozwala różnym kompilatorom, asemblerom, linkerom, debuggerom ”współpracować”

# Executable Linkable Format - c.d.

## Budowa



# ELF - odczytywanie informacji

## file

```
$file ex.o ex
ex.o: ELF 64-bit LSB relocatable, x86-64, version 1
(SYSV), not stripped
ex: ELF 64-bit LSB executable, x86-64, version 1
(SYSV), dynamically linked (uses shared libs), for
GNU/Linux 2.6.26, BuildID[sha1]=0x2ddedab8e7
68061080c4953c9824982918bafb59, not stripped
```

# ELF - odczytywanie informacji c.d.

## readelf

składnia:
readelf *OPCJA*... *PLIK*...
OPCJE:

```
[-h|--file-header]
[-l|--program-headers|--segments]
[-S|--section-headers|--sections]
[-a|--all] to samo co -hlS <więcej_opcji>
```

# ELF - odczytywanie informacji c.d.

## readelf - przykład

```
$readelf -h ex2
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x400440
  Start of program headers:          64 (bytes into file)
  Start of section headers:          2960 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         8
  Size of section headers:           64 (bytes)
  Number of section headers:         35
  Section header string table index: 32
```


# cpp — C Preprocessor

```
1  #define N 5
2  #define SQRT(x) x*x
3  ...
4  int a = N;
5  int b = SQRT(a++);
```

# cpp — C Preprocessor

```
1  #define N 5
2  #define SQRT(x) x*x
3  ...
4  int a = N;
5  int b = SQRT(a++);
```

Wywołanie *cpp plik.c*:

```
1  int a = 5;
2  int b = a++*a++;
```

# cpp — C Preprocessor

```
1  #define N 5
2  #define SQRT(x) x*x
3  ...
4  int a = N;
5  int b = SQRT(a++);
```

Wywołanie *cpp plik.c*:

```
1  int a = 5;
2  int b = a++*a++;
```

## Co robi?

1. Dołączanie (sklejanie w jeden) plików
2. Rozwijanie makr
3. Kompilacja warunkowa
4. Usuwanie komentarzy

# gcc - to co się przyda

## gcc -g

Flaga przydatna do dalszego debugowania przez gdb

## gcc -v

verbose — dodatkowy output, warto na niego zwracać uwagę

## gcc -*

Często działa w przypadku innych narzędzi

# gcc - to co się przyda

## gcc -g

Flaga przydatna do dalszego debugowania przez gdb

## gcc -v

verbose — dodatkowy output, warto na niego zwracać uwagę

## gcc -*

Często działa w przypadku innych narzędzi

## gcc -S

- Tylko kompilacja (i preprocesor)
- Nie uruchamia assemblacji (gas) oraz linkera (ld)
- Tworzy plik *.s*

-fverbose-asm

- W komentarzach pojawiają się dodatkowo nazwy używanych zmiennych
- Dla kompilatora Intel domyślne

# gcc — to co się przyda

## gcc -m*

- Opcje związane z architekturą
- Dostępne: ARM, AVR, i386, x86-64, IA-64, MicroBlaze...

## -masm=dialect

- intel (dodaje dyrektywę .intel_syntax rozumianą przez GAS)
- att (domyślny)

## -m (i386, x86-64)

- -m32 - sizeof(int, long, *ptr) == 4
- -mx32 - to co wyżej, ale kod dla x86-64
- -m64 - sizeof(int) == 4, sizeof(long, *ptr) == 8

# gcc — to co się przyda — i386, x86-64

## gcc -march=cpu_type

- Tworzy kod pod konkretną architekturę
- Brak gwarancji działania na innych architekturach

## gcc -mtune=cpu_type

- Dostraja kod pod konkretną architekturę
- Gwarancja działania na architekturze, na której gcc został skonfigurowany

## cpu_types

- i386, pentium, pentium3, core2, corei7, atom, athlon-xp . . .
- native — maszyna, na której aktualnie pracujemy
- -mtune=generic — dla najpopularniejszych IA32/AMD64/EM64T — jeżeli nie bardzo wiemy na czym będzie działać

# gcc — cross compilation

## Wymagania

- binutils zbudowany pod konkretną platformę (opcja *–target=*)
- gcc skonfigurowane z podobną opcją
- biblioteki skompilowane pod docelową platformę

# gas — GNU Assembler

- Od jakiegoś czasu wspiera dwie składnie: AT&T oraz Intel
- Aby używać składni Intela, należy umieścić dyrektywę .intel_syntax na początku pliku *.s*
- Dla AT&T dyrektywa ta to .att_syntax
- Po obu dyrektywach może wystąpić opcjonalny argument *prefix* lub *noprefix*. Określa on, czy rejestry wymagają poprzedzenia prefiksem *%*

# gas — GNU Assembler

- Od jakiegoś czasu wspiera dwie składnie: AT&T oraz Intel
- Aby używać składni Intela, należy umieścić dyrektywę .intel_syntax na początku pliku *.s*
- Dla AT&T dyrektywa ta to .att_syntax
- Po obu dyrektywach może wystąpić opcjonalny argument *prefix* lub *noprefix*. Określa on, czy rejestry wymagają poprzedzenia prefiksem *%*

## i386 options

```
[--32|--n32|--64] [-n]
[-march=CPU[+EXTENSION...]] [-mtune=CPU]
```

# gas — GNU Assembler

## Output

`-a[cdghlns]`

- c — omits false conditionals from a listing
- d — omits debugging directives from the listing
- g — prints a first section with general assembly information, like as version, switches passed, or time stamp
- h — requests a high-level language listing
- l — requests an output-program assembly listing
- s — requests a symbol table listing
- n — controls forms processing

# ld — linker

- Przy podawaniu opcji **-l** ważna kolejność — linker odczytuje od lewej do prawej, czyli lib zawierający definicję pojawia się PO pliku źródłowym, który z niego korzysta; Przykład: gcc -c elo.o -lm
- Przeszukiwanie katalogów pod kątem:
  - plików nagłówkowych (include path): -I
  - plików biblioteki (link path): -L
- Opcja -static wymusza linkowanie statyczne
- ldd - pokazuje biblioteki statyczne powiązane z plikiem wykonywalnym
- Poza prostymi przypadkami (ld elo.o -o elo) bardzo ciężki do wywoływania samemu, ze względu na mnogość wymaganych opcji (np. dla zwykłego HelloWorld w C) jest ich około 20

# collect2 — co to takiego?

collect2: ld returned 1 exit status

# collect2 — co to takiego?

collect2: ld returned 1 exit status

- ld to nie ld, to collect2
- collect2 przeszukuje różne ścieżki w poszukiwaniu *real-ld* — właściwego linkera, uważając żeby nie znalazł samego siebie
- collect2 linkuje program, przeszukuje output linkera w poszukiwaniu konstruktorów. Jeżeli je znajdzie, tworzy ich tablicę, którą zapisuje do pliku .c. Plik ten jest ponownie linkowany z właściwym programem
- Właściwe wywołania konstruktorów obsługiwane są przez funkcję *__main*, wywoływaną na początku *main*
- Wywoływanie *__main* jest konieczne nawet w przypadku plików .c, gdyż umożliwia łączenie kodu C z obiektowym C++
- Nowe implementacje ELF tego nie potrzebują, same w sobie zawierają odpowiedne sekcje

# archiver — ar

- Używany w celu tworzenia oraz edycji archiwów — głównie statycznych bibliotek *.a*
- Do innych celów raczej wyparty np. przez *tar*
- Brak standardu opisującego tworzony format — są jednak dwa popularne — BSD oraz GNU
- Potrafi zapisać w archiwum prawa dostępu (mode), timestamp, właściciela, grupę oraz później to odtworzyć
- Pozwala na tworzenie indeksu symboli zdefiniowanych w plikach *.o* wchodzących w skład archiwum
- Daje możliwość stworzenia tzw. chudego archiwum — przykład
- Twórcy *ara* stworzyli prosty język skryptowy służący do jego obsługi (poza wierszem poleceń)

# Debugowanie

Debugowanie to kontrolowanie wykonania programu, mające na celu testowanie jego działania, lokalizację źródeł błędów, w szczególności poprzez:

- krokowe wykonywanie instrukcji programu
- monitorowanie stosu, ustawianie wartości zmiennych / rejestrów

## Co jest potrzebne?

- debugger - program obsługujący format zapisu pliku wykonywalnego
- symbole debugowania (np. znajdujące się w samym pliku wykonywalnym) (flaga gcc -g)
- zalecane wyłączenie optymalizacji przez kompilator (gcc -O0)
- źródła


# GNU Project Debugger - GDB (wprowadzenie)

- pierwsza wersja: 1986, Richard Stallman
- aktualna stabilna wersja: 7.5.1
- Obsługiwane architektury (niektóre): ARM, x86, x86-64, IA-64, PowerPC, SPARC
- System operacyjny: systemy uniksowe oraz z rodziny Windows
- Obsługiwane języki: Ada, C, C++, Objective-C, D, Go, Java, Fortran, Pascal, Modula-2, OpenCL, asemblery
- posiada Command Line Interface, oraz Text User Interface. Brak GUI zapewnia lepszą przenośność.

# GDB - podstawy

1. Uruchomienie debuggera

   `$gdb <program>` - uruchamia debugger

   `$gdb <program> <proces_nr>` - j/w, debugger dołącza się do procesu `<proces_nr>` (automatyczny break)

   `$gdb --args <program> <argumenty>` - program będzie uruchamiany z podanymi argumentami (każdorazowo)
2. Zastawienie pułapek

   *breakpointy*

   `(gdb) break <miejsce>` - zastawia pułapkę w `<miejsce>`

## Przykłady

```
break 10, b main, b source.cpp:23, b source.cpp:recalculate, b *0x400590
```

`b 10 if (i==4)` - warunkowy breakpoint

# GDB - podstawy (2)

*watchpointy - zatrzymanie przy próbie:*

- `(gdb) watch <wyrażenie>` - zapisu
- `(gdb) rwatch <wyrażenie>` - odczytu
- `(gdb) awatch <wyrażenie>` - zapisu lub odczytu...

...miejsca w pamięci określonego przez `<wyrażenie>`, najczęściej jest to nazwa zmiennej.

Uwaga: lokalizacje breakpointów można wyeksportować do pliku używając polecenia `(gdb) save breakpoints <nazwa_pliku>`. Do importu używa się komendy `(gdb) source <nazwa_pliku>`.

3. Uruchomienie programu

`(gdb) run` - startuje proces

`(gdb) run <argumenty>` - startuje proces, przesyłając mu podane argumenty (jednorazowo), argumenty podane w tym miejscu zastępują te podane w opcjach gdb.

# GDB - polecenia

## „Kroczkowanie“

- `(gdb) next`
- `(gdb) nexti`
- `(gdb) step`
- `(gdb) stepi`
- `(gdb) continue`
- `(gdb) finish`

## Podgląd wartości

- `(gdb) print <wyr>`
- `(gdb) info args`
- `(gdb) info locals`
- `(gdb) info variables`
- `(gdb) info registers`
- `(gdb) bt full`

## Modyfikacja wartości / przebiegu

- `(gdb) set var x=5`
- `(gdb) print fun(arg)`
- `(gdb) return`

# GDB - debugowanie wstecz

Dostępne jest począwszy od wersji 7. GDB, dla i386-linux i amd64-linux.

1. Uruchomienie rejestracji: `(gdb) record`
2. Wykonywanie programu
3. Wykonywanie wstecz (możliwe do (od ;-)) momentu rozpoczęcia rejestracji)

## Instrukcje ”cofania”

`reverse-next`, `reverse-nexti`, `reverse-step`, `reverse-stepi`, `reverse-continue`, `reverse-finish`

## Checkpointy

”Stan” procesu - wartość zmiennych, rejestrów, stosu - można zapisać, tworząc jakby ”obraz” aktualnego stanu, za pomocą polecenia `(gdb) checkpoint`. Taki checkpoint dostaje swój unikalny id, który można sprawdzić tak: `(gdb) info checkpoints`. Polecenie `restart <checkpoint_id>` przywraca zapisany stan. Zmienia się tylko pid.

# GDB - TUI

Tekstowy interfejs użytkownika wymaga biblioteki curses.
Zalety:

- umożliwia podgląd na bieżąco wartości rejestrów, kodu źródłowego i języka asemblera
- czytelniej niż w CLI prezentuje kod źródłowy.
- wygodny tryb ”Single Key Mode” (`CTRL+X S`)

Sposoby uruchomienia:

- `$gdb -tui`
- po uruchomieniu gdb, przełączenie do trybu TUI za pomocą kombinacji `CTRL+X A`

Nie potrafi prezentować wartości zmiennych w czasie rzeczywistym.

# Data Display Debugger (DDD)

- DDD jest graficznym interfejsem użytkownika przeznaczonym dla GDB.
- Pozwala na bieżąco wyświetlać stos, zmienne lokalne i argumenty.
- Pozwala definiować własne wyświetlacze prezentujące własne wyrażenia.
- Umożliwia wyrysowanie wartości tablicy w postaci wykresów 2D lub 3D.

# objdump

Prezentuje wybrane dane z plików binarnych.
Składnia: `objdump <opcje> <pliki>`

## opcje służące dezasemblacji

- `-d` - dezasembluje tylko te sekcje, które wydają się zawierać instrukcje
- `-D` - dezasembluje wszystkie sekcje
- `-M intel-mnemonics` - używa mnemoników i składni Intela
- `--endian={big|little}` - określa kolejność bajtów
- `-l` - dodaje na wyjściu nazwy plików i numery linii
- `-S` - wyświetla wynik dezasemblacji przepleciony odpowiednimi instrukcjami z kodu źródłowego.
- `--start-address=<adr1>, --stop-address=<adr2>` - wyznaczają granice obszaru do dezasemblacji

## opcje służące badaniu zawartości

- `-s -j <nazwa>` - wyświetla całą zawartość wybranej sekcji
- `-xw` - wyświetla wszystkie nagłówki (dodatkowo w szerokim **w**ide) formacie

# strings

Domyślnie wyszukuje cztero-i-więcej-znakowe ciągi znaków drukowalnych, zakończone znakiem niedrukowalnym, pochodzące z już załadowanych i zainicjalizowanych sekcji plików binarnych.

**Składnia:**

```
$strings [<opcje>] <plik(i)>
```

**Wybrane opcje:**

- `-a` - sprawdza wszystkie sekcje
- `-n <dlg>` - ustala minimalną długość ciągu znaków
- `-e <kodowanie>` - ustala kodowanie

# GNU profiler gprof

Które fragmenty programu zajmują najwięcej czasu? Które funkcje są najczęściej wywoływane i przez które?

**Trzy etapy analizy:**

1. Kompilacja z włączonym profilowaniem $\Rightarrow$ dodanie flagi `-pg` przy kompilacji i linkowaniu przez gcc.
2. Wygenerowanie pliku gmon.out z informacjami poprzez wykonanie programu. Program musi zakończyć się w sposób prawidłowy.
3. Interpretacja wyników

W celu interpretacji wyników należy wywołać polecenie:

```
$gprof <opcje> <program=a.out>
<plik_z_danymi=gmon.out>
```

Można podać kilka plików z danymi, wtedy statystyka zbierana jest z sumy informacji.

# GNU profiler gprof (c.d.)

**Wybrane opcje:**

- `-z` - podaje w statystykach także niewywoływane funkcje
- `-b` - pomija długie objaśnienia na wyjściu

## Błędy pomiarowe

Ilość wywołań funkcji nie jest podatna na błędy związane z próbkowaniem, w przeciwieństwie do czasu ich wykonywania. Dokładność pomiaru czasu jest dobra, gdy całkowity czas wykonywania programu jest znacznie większy od czasu próbkowania. Typowy okres próbkowania wynosi 0.01s. Blad pomiaru można oszacować jako $\sqrt{n}$, gdzie $n$ - ilość próbek.

# gcov

- prosty profiler wykorzystywany głównie w celu określenia ile razy wykonują się poszczególne linie w programie

```
grep '######' *.gcov
```

pokaże linie, które nie są wykonywane