Tehnici de debugging

Mihai Maruseac
mihai.maruseac@gmail.com

ROSEdu

3 februarie 2010

Fundamental Axiom of Programming (Rick Cook)

Programming today is a race between software engineers striving to build bigger and better idiot-proof programs, and the Universe trying to produce bigger and better idiots. So far, the Universe is winning.

Bugs

- Bug = error, flaw, mistake, failure or flaw in computer code producing an unexpected result or behaviour.

Bugs

- Bug = error, flaw, mistake, failure or flaw in computer code producing an unexpected result or behaviour.
- Jim Gray's classification:
  - Bohrbug - repetabil, mereu în același set de condiții (permanent fault)

Bugs

- Bug = error, flaw, mistake, failure or flaw in computer code producing an unexpected result or behaviour.
- Jim Gray's classification:
  - Bohrbug - repetabil, mereu în același set de condiții (permanent fault)
  - Heisenbug - schimbă comportamenul în debugging (transient fault)

Bugs

- Bug = error, flaw, mistake, failure or flaw in computer code producing an unexpected result or behaviour.
- Jim Gray's classification:
  - Bohrbug - repetabil, mereu în același set de condiții (permanent fault)
  - Heisenbug - schimbă comportamenul în debugging (transient fault)
  - Mandelbug - cauze complexe și obscure, haotic, nedeterminist

Bugs

- Bug = error, flaw, mistake, failure or flaw in computer code producing an unexpected result or behaviour.
- Jim Gray's classification:
  - Bohrbug - repetabil, mereu în același set de condiții (permanent fault)
  - Heisenbug - schimbă comportamenul în debugging (transient fault)
  - Mandelbug - cauze complexe și obscure, haotic, nedeterminist
  - Schroedinbug

Efecte

- distrugerea navetelor spatiale (Mariner 1, Ariane 5, Surveyor)
- iradiere excesivă
- panică (Y2K)

Efecte

- distrugerea navetelor spatiale (Mariner 1, Ariane 5, Surveyor)
- iradiere excesivă
- panică (Y2K)
- accidente militare (Patriot, Yorktown)

Efecte

- distrugerea navetelor spatiale (Mariner 1, Ariane 5, Surveyor)
- iradiere excesivă
- panică (Y2K)
- accidente militare (Patriot, Yorktown)
- accidente media (Eve Online)

Efecte

- distrugerea navetelor spatiale (Mariner 1, Ariane 5, Surveyor)
- iradiere excesivă
- panică (Y2K)
- accidente militare (Patriot, Yorktown)
- accidente media (Eve Online)
- spargerea securității (Valgrind + OpenSSL on Debian; 2006-2008)

Cauze

- typo, thinko, mouso

Cauze

- typo, thinko, mouso
- memory leaks, stack smashing, fandango on core (malloc..)

Cauze

- typo, thinko, mouso
- memory leaks, stack smashing, fandango on core (malloc..)
- race conditions

Cauze

- typo, thinko, mouso
- memory leaks, stack smashing, fandango on core (malloc..)
- race conditions
- rounding errors

Cauze

- typo, thinko, mouso
- memory leaks, stack smashing, fandango on core (malloc..)
- race conditions
- rounding errors
- conversii între tipuri

Soluții preventive

- -Wall -Wextra..

Soluții preventive

- -Wall -Wextra..
- Testarea codului

Soluții preventive

- -Wall -Wextra..
- Testarea codului
- Unit testing

Soluții preventive

- -Wall -Wextra..
- Testarea codului
- Unit testing
- După teste, scăpăm doar de Bohrbugs

Soluții preventive

- -Wall -Wextra..
- Testarea codului
- Unit testing
- După teste, scăpăm doar de Bohrbugs
- Analiza (statică a) codului

Comentarii

- Tehnică fundamentală
- Funcționează mereu
- raport beneficii / efort foarte mic

Print*

- printf pentru cod în user space
- printk pentru cod de kernel

Print*

- printf pentru cod în user space
- printk pentru cod de kernel
- modificare cod
- convenții de indentare, formatare, .. interne proiectului
- analiză text >> analiză cod
- ad hoc. temporal. repetitiv
- reduce performanțe, încarcă output
- buffered output (not stderr)

```
#include <stdio.h>

int main()
{
    char a, sir[256];
    int i = 0;
    for (a = 0; a < 256; a++)

        sir[i++] = a;

    printf("Result: %s\n", sir);
    return 0;
}
```

```c
#include <stdio.h>

int main()
{
    char a, sir[256];
    int i = 0;
    for (a = 0; a < 256; a++){
        printf(">>> %d %d %c\n", i, a, a);
        sir[i++] = a;
    }
    printf("Result: %s\n", sir);
    return 0;
}
```

Despre print*

- modificarea codului: eficiență vs. rezultate
- Când preferăm print*?

Despre print*

- modificarea codului: eficiență vs. rezultate
- Când preferăm print*?
- Cod recursiv, prea mult efort cu alte instrumente
- Cod paralel
- Cod greu de depanat altfel (kernel code)
- Heisenbugs

Despre print*

- modificarea codului: eficiență vs. rezultate
- Când preferăm print*?
- Cod recursiv, prea mult efort cu alte instrumente
- Cod paralel
- Cod greu de depanat altfel (kernel code)
- Heisenbugs
- Jurnalizare

Despre print*

- modificarea codului: eficiență vs. rezultate
- Când preferăm print*?
- Cod recursiv, prea mult efort cu alte instrumente
- Cod paralel
- Cod greu de depanat altfel (kernel code)
- Heisenbugs
- Jurnalizare
- efort minim de eliminare
- macrou-ri de debug

Despre print*

- modificarea codului: eficiență vs. rezultate
- Când preferăm print*?
- Cod recursiv, prea mult efort cu alte instrumente
- Cod paralel
- Cod greu de depanat altfel (kernel code)
- Heisenbugs
- Jurnalizare
- efort minim de eliminare
- macrou-ri de debug

Macro debug ideal (v. TT01)

```
#define DEBUG(fmt, ...) fprintf(LOGFILE, fmt, ##__VA_ARGS__)
```

GDB

- GNU Debugger - Depanatorul standard
- Portabil: sisteme + limbaje
- Stallman, 1986

GDB

- GNU Debugger - Depanatorul standard
- Portabil: sisteme + limbaje
- Stallman, 1986
- 5 lucruri majore:
  - Pornire program, specificare parametri
  - Forțare oprire program condiționat sau obligatoriu
  - Examinare pași pentru oprire prematură (seg fault?)
  - Schimbare variabile (instrucțiuni ?)
  - Rulare inversă - din versiunea 7.0 (2009)

GDB

- GNU Debugger - Depanatorul standard
- Portabil: sisteme + limbaje
- Stallman, 1986
- 5 lucruri majore:
  - Pornire program, specificare parametri
  - Forțare oprire program condiționat sau obligatoriu
  - Examinare pași pentru oprire prematură (seg fault?)
  - Schimbare variabile (instrucțiuni ?)
  - Rulare inversă - din versiunea 7.0 (2009) - incomplet

GDB - basic

Compilare

gcc -g -Wall -O0 *sursa*

Depanare

gdb *executabil*

DDD

- Limitare gravă GDB: TUI

DDD

- Limitare gravă GDB: TUI
- Soluție: Data Display Debugger

DDD

- Limitare gravă GDB: TUI
- Soluție: Data Display Debugger
- Avantaj: date >> cod

DDD

- Limitare gravă GDB: TUI
- Soluție: Data Display Debugger
- Avantaj: date >> cod
- Compilare identică, execuție în GUI

DDD

- Limitare gravă GDB: TUI
- Soluție: Data Display Debugger
- Avantaj: date >> cod
- Compilare identică, execuție în GUI
- Demo...

Cheatsheet GDB

Rulare program	run (r)
Listare cod program	list (l)
Afișare stivă apel	backtrace (bt)
Introducere întrerupere	break (b)
Breakpoint condițional	b if cond
Continuare execuție	continue (c)
Avansare un pas (fără apel)	next (n)
Avansare o instrucțiune	step (s)
Continuare până la return	finish (fin)
Repetare comandă anterioară	*blank*
Afișare conținut variabilă	print (p)
Afișare permanentă	display (disp)
Oprire la modificare	watch (wa)
Modificare variabilă	set var *expr*

Pentru toate celelalte există help (h).

Valgrind

- Remember Heisenbugs?

Valgrind

- Remember Heisenbugs?
- Valgrind - depanator memorie generic, framework analiză dinamică
- Mașină virtuală - JIT

Valgrind

- Remember Heisenbugs?
- Valgrind - depanator memorie generic, framework analiză dinamică
- Mașină virtuală - JIT
- cod $\rightarrow$ Intermediate Representation $\xrightarrow{\textit{Instrument de testare}}$ Modificări ale IR $\rightarrow$ Cod mașină rulat în gazdă.

Valgrind

- Remember Heisenbugs?
- Valgrind - depanator memorie generic, framework analiză dinamică
- Mașină virtuală - JIT
- cod $\rightarrow$ Intermediate Representation $\xrightarrow{\textit{Instrument de testare}}$ Modificări ale IR $\rightarrow$ Cod mașină rulat în gazdă.
- performanță – –

Valgrind

- Remember Heisenbugs?
- Valgrind - depanator memorie generic, framework analiză dinamică
- Mașină virtuală - JIT
- cod $\rightarrow$ Intermediate Representation $\xrightarrow{\textit{Instrument de testare}}$ Modificări ale IR $\rightarrow$ Cod mașină rulat în gazdă.
- performanță – –
- cod IR - ideal pentru scrierea de instrumente de testare

Memcheck

- Implicit
- Cod extra în jurul fiecărei instrucțiuni

Memcheck

- Implicit
- Cod extra în jurul fiecărei instrucțiuni
- Sanitizare memorie: validitate + adresabilitate

Memcheck

- Implicit
- Cod extra în jurul fiecărei instrucțiuni
- Sanitizare memorie: validitate + adresabilitate
- Mecanism propriu de adresare și alocare memorie

Memcheck

- Implicit
- Cod extra în jurul fiecărei instrucțiuni
- Sanitizare memorie: validitate + adresabilitate
- Mecanism propriu de adresare și alocare memorie
- De 20 de ori mai lent, extrem de util
- Probleme detectate:
  - memorie neinițializată
  - citiri / scrieri zone eliberate
  - citiri / scrieri final zone alocate
  - scurgeri memorie

Python

5 tehnici de debug:

- print - mai eficient pentru limbaje interpretate

Python

5 tehnici de debug:

- print - mai eficient pentru limbaje interpretate
- logging - 5 nivele personalizate

Python

5 tehnici de debug:

- print - mai eficient pentru limbaje interpretate
- logging - 5 nivele personalizate
- code.interact - doar o simplă inserție

Python

5 tehnici de debug:

- print - mai eficient pentru limbaje interpretate
- logging - 5 nivele personalizate
- code.interact - doar o simplă inserție
- invocare pdb (gdb python) prin cod

Python

5 tehnici de debug:

- print - mai eficient pentru limbaje interpretate
- logging - 5 nivele personalizate
- code.interact - doar o simplă inserție
- invocare pdb (gdb python) prin cod
- invocare pdb din linia de comandă

Java

4 tehnici de debug:

- System.out.println - BAD

Java

4 tehnici de debug:

- System.out.println - BAD
- Applets - draw text

Java

4 tehnici de debug:

- System.out.println - BAD
- Applets - draw text
- jdb (gdb java):

Java

4 tehnici de debug:

- System.out.println - BAD
- Applets - draw text
- jdb (gdb java):
  - Nu respectă standardul de la gdb

Java

4 tehnici de debug:

- System.out.println - BAD
- Applets - draw text
- jdb (gdb java):
  - Nu respectă standardul de la gdb
  - stop at/in în loc de break

Java

4 tehnici de debug:

- System.out.println - BAD
- Applets - draw text
- jdb (gdb java):
  - Nu respectă standardul de la gdb
  - stop at/in în loc de break
  - methods

Java

4 tehnici de debug:

- System.out.println - BAD
- Applets - draw text
- jdb (gdb java):
  - Nu respectă standardul de la gdb
  - stop at/in în loc de break
  - methods
  - greu de folosit, nedocumentat

Java

4 tehnici de debug:

- System.out.println - BAD
- Applets - draw text
- jdb (gdb java):
  - Nu respectă standardul de la gdb
  - stop at/in în loc de break
  - methods
  - greu de folosit, nedocumentat
- Folosire GUI

MPI

- paralelism + debugging = probleme
- depanăm un singur thread sau toate?
- Cel mai bine folosim jurnalizare
- Există utilitare de debug: Total View
- Putem folosi gdb cu un mic trick/cheat

MPI

- paralelism + debugging = probleme
- depanăm un singur thread sau toate?
- Cel mai bine folosim jurnalizare
- Există utilitare de debug: Total View
- Putem folosi gdb cu un mic trick/cheat

Paralelizare gdb + xterm

mpirun -np *N* xterm -e gdb *program*

Expert debugging

- printk vs printf

Expert debugging

- printk vs printf
  - logging levels: <linux/kernel.hi>

```
printk(KERN_CRIT "msg");
```

  - syslogd - jurnalizare
  - line buffered sau nu, în funcție de console_loglevel
  - buffer circular mic (4KB - 1MB)
  - slows down the system

Expert debugging

- printk vs printf
  - logging levels: <linux/kernel.hi>

```
printk(KERN_CRIT "msg");
```

  - syslogd - jurnalizare
  - line buffered sau nu, în funcție de console_loglevel
  - buffer circular mic (4KB - 1MB)
  - slows down the system
- debugging by querying: ps, netstat, ioctl, sysfs, /proc

Expert debugging

- printk vs printf
  - logging levels: <linux/kernel.hi>

```
printk(KERN_CRIT "msg");
```

  - syslogd - jurnalizare
  - line buffered sau nu, în funcție de console_loglevel
  - buffer circular mic (4KB - 1MB)
  - slows down the system
- debugging by querying: ps, netstat, ioctl, sysfs, /proc
- debugging by watching: strace

Expert debugging

- printk vs printf
  - logging levels: <linux/kernel.hi>

```
printk(KERN_CRIT "msg");
```

  - syslogd - jurnalizare
  - line buffered sau nu, în funcție de console_loglevel
  - buffer circular mic (4KB - 1MB)
  - slows down the system
- debugging by querying: ps, netstat, ioctl, sysfs, /proc
- debugging by watching: strace
- using gdb

Kernel debugging with gdb

- invocare ca și cum nucleul ar fi aplicație:

```
gdb /usr/src/linux/vmlinux /proc/kcore
```

Kernel debugging with gdb

- invocare ca și cum nucleul ar fi aplicație:

```
gdb /usr/src/linux/vmlinux /proc/kcore
```

- folosire kdb - pe kdb-enabled kernels

Kernel debugging with gdb

- invocare ca și cum nucleul ar fi aplicație:

```
gdb /usr/src/linux/vmlinux /proc/kcore
```

- folosire kdb - pe kdb-enabled kernels
- virtualizare - greu de realizat

Kernel debugging with gdb

- invocare ca și cum nucleul ar fi aplicație:

```
gdb /usr/src/linux/vmlinux /proc/kcore
```

- folosire kdb - pe kdb-enabled kernels
- virtualizare - greu de realizat
- User Mode Linux

C Preprocessor Macros

- Implicit, toate sunt expandate

C Preprocessor Macros

- Implicit, toate sunt expandate
- Chiar și la -g

C Preprocessor Macros

- Implicit, toate sunt expandate
- Chiar și la -g
- Compilăm cu -g3 -gdwarf-2

C Preprocessor Macros

- Implicit, toate sunt expandate
- Chiar și la -g
- Compilăm cu -g3 -gdwarf-2
- macro expand *expression*

C Preprocessor Macros

- Implicit, toate sunt expandate
- Chiar și la -g
- Compilăm cu -g3 -gdwarf-2
- macro expand *expression*
- info macro *macro*

C Preprocessor Macros

- Implicit, toate sunt expandate
- Chiar și la -g
- Compilăm cu -g3 -gdwarf-2
- macro expand *expression*
- info macro *macro*
- ## și # - nesuportate

C Preprocessor Macros

- Implicit, toate sunt expandate
- Chiar și la -g
- Compilăm cu -g3 -gdwarf-2
- macro expand *expression*
- info macro *macro*
- ## și # - nesuportate
- macro-uri cu număr variabil de argumente - nesuportate

Tracepoints

- Cum depanăm aplicațiile real-time?

Tracepoints

- Cum depanăm aplicațiile real-time?
- trace and collect = tracepoints

Tracepoints

- Cum depanăm aplicațiile real-time?
- trace and collect = tracepoints
- trace *linie*

Tracepoints

- Cum depanăm aplicațiile real-time?
- trace and collect = tracepoints
- trace *linie*
- actions

Tracepoints

- Cum depanăm aplicațiile real-time?
- trace and collect = tracepoints
- trace *linie*
- actions
- collect *vars*

Tracepoints

- Cum depanăm aplicațiile real-time?
- trace and collect = tracepoints
- trace *linie*
- actions
- collect *vars*
- passcount

Tracepoints

- Cum depanăm aplicațiile real-time?
- trace and collect = tracepoints
- trace *linie*
- actions
- collect *vars*
- passcount
- tstart, tstop

Tracepoints

- Cum depanăm aplicațiile real-time?
- trace and collect = tracepoints
- trace *linie*
- actions
- collect *vars*
- passcount
- tstart, tstop
- tfind, tdump - examinare

Tracepoints

- Cum depanăm aplicațiile real-time?
- trace and collect = tracepoints
- trace *linie*
- actions
- collect *vars*
- passcount
- tstart, tstop
- tfind, tdump - examinare
- save-tracepoints *file*

Tracepoints

- Cum depanăm aplicațiile real-time?
- trace and collect = tracepoints
- trace *linie*
- actions
- collect *vars*
- passcount
- tstart, tstop
- tfind, tdump - examinare
- save-tracepoints *file*
- funcționează doar pentru remote debugging

Remote debugging

- Necesită o copie a programului pe gazdă

Remote debugging

- Necesită o copie a programului pe gazdă
- Invocare uzuală

Remote debugging

- Necesită o copie a programului pe gazdă
- Invocare uzuală
- comandă remote: *target remote:port*

Remote debugging

- Necesită o copie a programului pe gazdă
- Invocare uzuală
- comandă remote: *target remote:port*
- gdbserver

Running app

- Putem depana un program care rulează?

Running app

- Putem depana un program care rulează?
- attach *PID*

Running app

- Putem depana un program care rulează?
- attach *PID*
- Procesul va rula și după detașare

State of the Art

- Completare posibilitate rulare înapoi
- Debugging și anti-debugging
- Automatic online debugging (compilare online automată - codepad)
- Differential debugging

Bibliografie, resurse utile

- http://www.quotationspage.com/quote/781.html
- http://catb.org/jargon/html/index.html
- http://sourceware.org/gdb/current/onlinedocs/gdb.html
- http://oopweb.com/CPP/Documents/DebugCPP/Volume/techniques.html
- http://www.gnu.org/software/gdb/news/reversible.html
- http://www.gnu.org/software/ddd/
- http://aymanh.com/python-debugging-techniques
- http://docs.python.org/library/pdb.html
- http://docs.python.org/library/logging.html
- http://www.javaworld.com/javaworld/javaqa/2000-06/04-qa-0623-jdb.html

Bibliografie, resurse utile (cont.)

- http://www.totalviewtech.com/
- http://user-mode-linux.sourceforge.net/
- http://www.linuxjournal.com/article/5749
- http://www-zeuthen.desy.de/dv/documentation/unixguide/infohtml/gdb/gdb_23.html
- http://codepad.org/

Întrebări

?

...

Fundamental Axiom of Programming (Rick Cook)

Programming today is a race between software engineers striving to build bigger and better idiot-proof programs, and the Universe trying to produce bigger and better idiots. So far, the Universe is winning.