



Razhroščevalniki

Debuggers

Pregled predstavitve

- **Pregled razhroščevalnikov**
 - Opis
 - Tipi
 - Funkcionalnosti
 - Primeri
 - GDB
- Delovanje razhroščevalnikov

Opis razhroščevalnikov

omogočajo **spremljanje poteka izvajanja** ciljnega programa:

- ustavitev
- pregled stanja

Tipi razhroščevalnikov

- source-level (simbolični) debugger vs machine-level debugger
 - simbolični (za višje programske jezike) potrebujejo mapiranje
- stand-alone debugger vs IDE
 - portabilnost vs integracija
- prevedeni vs interpretirani jezik
 - gdb vs xdebug

Funkcionalnosti razhroščevalnikov

- **prekinitvene točke (breakpoint)**
- **izvajanje programa korak za korakom** (single-stepping / step by step)
- **pregled trenutnih vrednosti** v registrih in spominu
- spreminjanje trenutnega stanja med tekom
- nadaljevanje izvajanja programa na drugi lokaciji
- prekinitvene točke pri dostopih do pomnilnika
- vzvratno razhroščevanje

Primeri razhroščevalnikov

- GDB - GNU debugger
- WinDbg
- Microsoft Visual Studio Debugger
- Eclipse debugger API
- Valgrind
- LLDB
- Chrome Dev Tools
- Xdebug

GDB – GNU Debugger

- zelo prenosljiv (nima GUI, samo CLI)
- teče na večih Unix-like sistemih
- Ada, C, C++, Objective-C, D, Pascal, Fortran, Go, Rust,...
- X86 in X64, IA-64, ARM, Alpha, AVR, H8/300, Motorola 68000, MIPS, PA-RISC, PowerPC, SuperH, SPARC,...
- Napisal Richard Stallman v letu 1986
- Še vedno je v aktivnem razvoju (8.o je izšla junija 2017), ki ga zdaj vodi GDB Steering Committee.

```
GNU gdb 6.1.1 [FreeBSD]
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "i386-marcel-freebsd"...
(gdb) b main
Breakpoint 1 at 0x80491d0: file locale.c, line 228.
(gdb) r
Starting program: /src/FreeBSD/svn/head/usr.bin/locale/locale

Breakpoint 1, main (argc=Error accessing memory address 0x2: Bad address.
) at locale.c:228
228 {
(gdb) l
223 const char *boguslocales[] = { "UTF-8" };
224 #define NBOGUS (sizeof(boguslocales)/sizeof(boguslocales[0]))
225
226 int
227 main(int argc, char *argv[])
228 {
229     int    ch;
230     int    tmp;
231
232     while ((ch = getopt(argc, argv, "ackms:")) != -1) {
(gdb) █
```

GDB – Remote debugging

- komunikacija preko TCP/IP ali serijske naprave - GDB protokol
- Tak program lahko ustvarimo z povezovanjem programa z določenimi GDB datotekami (naredimo **`gdb stub`**) ali pa uporabimo **`gdbserver`**.

GDB - Frontends

- KDbg (del KDE razvojnih orodij)
- Emacs
- Qt Creator
- Xcode
- CodeBlocks
- Eclipse C/C++ Development Tools
- CodeLite
- Visual Studio
- GDBGUI

gdbgui - gdb in a browser

127.0.0.1:5000

Load Binary /home/csmith/git/gdbgui/examples/c/hello_c.a myarg done

Enter source file path to view, or load binary then populate and select from dropdown

jump to line fetch disassembly reload file/hide disassembly /home/csmith/git/gdbgui/examples/c/hello.c:33

25 int unionint;

26 double uniondouble;

27 }

28 };

29

30 int main(void) {

31 printf("Hello World\n");

32

33 int retval = 1;

34

35 struct mystruct_t s; /* sizeof(struct mystruct_t) bytes are allocated for s,

36 but still contain garbage */

37 s.value = 100;

38 s.string = "pass";

39 s.substruct.dbl = 567.8;

40

41 s.letter = 'P';

42 s.fp = 123.4;

43

44 s.ptr = say_something; /* address of function */

45 s.ptr = &say_something; /* also address of function */

46 s.unionint = 0;

47 s.uniondouble = 1.0;

48

49 for (int i = 0; i < 2; i++) {

50

51 printf("i is %d\n", i);

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

122

123

124

125

126

127

128

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

180

181

182

183

184

185

186

187

188

189

190

191

192

193

194

195

196

197

198

199

200

201

202

203

204

205

206

207

208

209

210

211

212

213

214

215

216

217

218

219

220

221

222

223

224

225

226

227

228

229

230

231

232

233

234

235

236

237

238

239

240

241

242

243

244

245

246

247

248

249

250

251

252

253

254

255

256

257

258

259

260

261

262

263

264

265

266

267

268

269

270

271

272

273

274

275

276

277

278

279

280

281

282

283

284

285

286

287

288

289

290

291

292

293

294

295

296

297

298

299

300

301

302

303

304

305

306

307

308

309

310

311

312

313

314

315

316

317

318

319

320

321

322

323

324

325

326

327

328

329

330

331

332

333

334

335

336

337

338

339

340

341

342

343

344

345

346

347

348

349

350

351

352

353

354

355

356

357

358

359

360

361

362

363

364

365

366

367

368

369

370

371

372

373

374

375

376

377

378

379

380

381

382

383

384

385

386

387

388

389

390

391

392

393

394

395

396

397

398

399

400

401

402

403

404

405

406

407

408

409

410

411

412

413

414

415

416

417

418

419

420

421

422

423

424

425

426

427

428

429

430

431

432

433

434

435

436

437

438

439

440

441

442

443

444

445

446

447

448

449

450

451

452

453

454

455

456

457

458

459

460

461

462

463

464

465

466

467

468

469

470

471

472

473

474

475

476

477

478

479

480

481

482

483

484

485

486

487

488

489

490

491

492

493

494

495

496

497

498

499

500

501

502

503

504

505

506

507

508

509

510

511

512

513

514

515

516

517

518

519

520

521

522

523

524

525

526

527

528

529

530

531

532

533

534

535

536

537

538

539

540

541

542

543

544

545

546

547

548

549

550

551

552

553

554

555

556

557

558

559

560

561

562

563

564

565

566

567

568

569

570

571

572

573

574

575

576

577

578

579

580

581

582

583

584

585

586

587

588

589

590

591

592

593

594

595

596

597

598

599

600

601

602

603

604

605

606

607

608

609

610

611

612

613

614

615

616

617

618

619

620

621

622

623

624

625

626

627

628

629

630

631

632

633

634

635

636

637

638

639

640

641

642

643

644

645

646

647

648

649

650

651

652

653

654

655

656

657

658

659

660

661

662

663

664

665

666

667

668

669

670

671

672

673

674

675

676

677

678

679

680

681

682

683

684

685

686

687

688

689

690

691

692

693

694

695

696

697

698

699

700

701

702

703

704

705

706

707

708

709

710

711

712

713

714

715

716

717

718

719

720

721

722

723

724

725

726

727

728

729

730

731

732

733

734

735

736

737

738

739

740

741

742

743

744

745

746

747

748

749

750

751

752

753

754

755

756

757

758

759

760

761

762

763

764

765

766

767

768

769

770

771

772

773

774

775

776

777

778

779

780

781

782

783

784

785

786

787

788

789

790

791

792

793

794

795

796

797

798

799

800

801

802

803

804

805

806

807

808

809

810

811

812

813

814

815

816

817

818

819

820

821

822

823

824

825

826

827

828

829

830

831

832

833

834

835

836

837

838

839

840

841

842

843

844

845

846

847

848

849

850

851

852

853

854

855

856

857

858

859

860

861

862

863

864

865

866

867

868

869

870

871

872

873

874

875

876

877

878

879

880

881

882

883

884

885

886

887

888

889

890

891

892

893

894

895

896

897

898

899

900

901

902

903

904

905

906

907

908

909

910

911

912

913

914

915

916

917

918

919

920

921

922

923

924

925

926

927

928

929

930

931

932

933

934

935

936

937

938

939

940

941

942

943

944

945

946

947

948

949

950

951

952

953

954

955

956

957

958

959

960

961

962

963

964

965

966

967

968

969

970

971

972

973

974

975

976

977

978

979

980

981

982

983

984

985

986

987

988

989

990

991

992

993

994

995

996

997

998

999

1000

push %rbp main+0 0x4005d1

mov %rsp,%rbp main+1 0x4005d2

sub \$0x50,%rsp main+4 0x4005d5

mov \$0x400738,%edi main+8 0x4005d9

callq 0x400470 <puts@plt> main+13 0x4005de

movl \$0x1,-0x48(%rbp) main+18 0x4005e3

movl \$0x64,-0x40(%rbp) main+25 0x4005ea

movq \$0x400744,-0x38(%rbp) main+32 0x4005f1

movsd 0x167(%rip),%xmm0 # 0x400768 main+40 0x4005f9

movsd %xmm0,-0x30(%rbp) main+48 0x400601

movb \$0x50,-0x3c(%rbp) main+53 0x400606

movss 0x15e(%rip),%xmm0 # 0x400770 main+57 0x40060a

movss %xmm0,-0x28(%rbp) main+65 0x400612

movq \$0x4005b6,-0x20(%rbp) main+70 0x400617

movq \$0x4005b6,-0x20(%rbp) main+78 0x40061f

movl \$0x0,-0x10(%rbp) main+86 0x400627

movsd 0x142(%rip),%xmm0 # 0x400778 main+93 0x40062e

movsd %xmm0,-0x10(%rbp) main+101 0x400636

addl \$0x1,-0x44(%rbp) main+135 0x400658

cmpl \$0x1,-0x44(%rbp) main+139 0x40065c

jle 0x400644 <main+115> main+143 0x400660

mov -0x44(%rbp),%eax main+115 0x400644

mov %eax,%esi main+118 0x400647

mov \$0x400749,%edi main+120 0x400649

mov \$0x0,%eax main+125 0x40064e

callq 0x400480 <printf@plt> main+130 0x400653

mov -0x38(%rbp),%rax main+145 0x400662

mov \$0x400744,%esi main+149 0x400666

mov %rax,%rdi main+154 0x40066b

callq 0x4004a0 <strcmp@plt> main+157 0x40066e

test %eax,%eax main+162 0x400673

jne 0x40067e <main+173> main+164 0x400675

movl \$0x0,-0x48(%rbp) main+166 0x400677

threads

inferior program: PID 28075

SIGINT send to inferior

selected process 28075, core 1, stopped, id 1

func	file	addr	args
main	hello.c:33	0x4005e3	

local variables

retval: 0 int

- s:{...} struct mystruct_t

value: 1 int

letter: 0 char

+ string: 0x4006fd char *

+ substruct:{...} struct {...}

- <anonymous struct>:{...} struct {...}

fp: 0 float

ptr: 0x4006b0 void *

struct_size: 4195520 size_t

- <anonymous union>:{...} union {...}

unionint: -7600 int

uniondouble: 6.9533558074595144e-310 double

expressions

Tree

memory

address	hex	char
0x400738	0x4005fd	8
more		
0x400738	48 65 6c 6c 6f 20 57 6f	Hello.Wo
0x400740	72 6c 64 00 70 61 73 73	rld.pass
0x400748	00 69 20 69 73 20 25 64	.i.is..d
0x400750	0a 00 72 65 74 75 72 6e	..return
more		

breakpoints

☒ main thread groups: i1

/home/csmith/git/gdbgui/examples/c/hello.c:31

printf("Hello World\n");

registers

name	value (hex)	value (decimal)
------	-------------	-----------------

gdbgui - gdb in a browser

127.0.0.1:5000

Load Binary /home/csmith/git/gdbgui/examples/c/hello_c.a myarg done

Enter source file path to view, or load binary then populate and select from dropdown

jump to line fetch disassembly reload file/hide disassembly /home/csmith/git/gdbgui/examples/c/hello.c:33

25 int unionint;

26 double uniondouble;

27 }

28 };

29

30 int main(void) {

31 printf("Hello World\n");

32

33 int retval = 1;

34

35 struct mystruct_t s; /* sizeof(struct mystruct_t) bytes are allocated for s,

36 but still contain garbage */

37 s.value = 100;

38 s.string = "pass";

39 s.substruct.dbl = 567.8;

40

41 s.letter = 'P';

42 s.fp = 123.4;

43

44 s.ptr = say_something; /* address of function */

45 s.ptr = &say_something; /* also address of function */

46 s.unionint = 0;

47 s.uniondouble = 1.0;

48

49 for (int i = 0; i < 2; i++) {

50

51 printf("i is %d\n", i);

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

122

123

124

125

126

127

128

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

180

181

182

183

184

185

186

187

188

189

190

191

192

193

194

195

196

197

198

199

200

201

202

203

204

205

206

207

208

209

210

211

212

213

214

215

216

217

218

219

220

221

222

223

224

225

226

227

228

229

230

231

232

233

234

235

236

237

238

239

240

241

242

243

244

245

246

247

248

249

250

251

252

253

254

255

256

257

258

259

260

261

262

263

264

265

266

267

268

269

270

271

272

273

274

275

276

277

278

279

280

281

282

283

284

285

286

287

288

289

290

291

292

293

294

295

296

297

298

299

300

301

302

303

304

305

306

307

308

309

310

Pregled predstavitve

- Pregled razhroščevalnikov
- **Delovanje razhroščevalnikov (prevedeni jeziki)**
 - PTRACE
 - Breakpointi
 - Strojna podpora
 - Informacije za razhroščevanje
- Delovanje razhroščevalnikov (interpretirani jeziki, virtual machine)

ptrace

- Sistemski klic v Linuxu, ki ga uporablja veliko število debuggerjev
- `long ptrace(enum __ptrace_request request, pid_t pid, void *addr, void *data);`
- *The ptrace() system call provides a means by which one process (the "tracer") may observe and control the execution of another process (the "tracee"), and examine and change the tracee's memory and registers. It is primarily used to implement breakpoint debugging and system call tracing. - `man 2 ptrace`*

ptrace – pripenjanje procesu

- PTRACE_TRACEME (fork() -> otrok gre TRACEME -> exec, starš pa wait())
- PTRACE_ATTACH
- *While being traced, the tracee will stop each time a signal is delivered, even if the signal is being ignored. (An exception is SIGKILL, which has its usual effect.) The tracer will be notified at its next call to waitpid(2) (or one of the related "wait" system calls); that call will return a status value containing information that indicates the cause of the stop in the tracee. While the tracee is stopped, the tracer can use various ptrace requests to inspect and modify the tracee. The tracer then causes the tracee to continue, optionally ignoring the delivered signal (or even delivering a different signal instead).*
- Primer v examples/ptrace_traceme_example.c

Ukazi – gdb / ptrace

gdb	ptrace	Razlaga
(gdb) start	PTRACE_TRACEME	Omogoča staršu, da lahko spremlja proces
(gdb) attach pid	PTRACE_ATTACH	Pripetje procesu, ki se že izvaja
(gdb) stop	kill(child_pid, SIGSTOP) (or PTRACE_INTERRUPT)	Ustavi ciljni proces
(gdb) continue	PTRACE_CONT	Nadaljuje izvajanje procesa
(gdb) info registers	PTRACE_GET(FP)REGS(ET) in PTRACE_SET(FP)REGS(ET)	Omogoča pregled in spreminjanje vrednosti registrov
(gdb) x	PTRACE_PEEKTEXT in PTRACE_POKE TEXT	Dovoljuje pregled in pisanje po spominu.

Breakpoint (prekinitvene točke)

- Izbrani ukaz si shrani v spomin, na to mesto pa zapiše:
 - poseben ukaz (na x86 je to INT 3)
 - nedefinirano kodo (npr. pri ARM v ta namen rezervirane posebne kode, ki so v dokumentaciji označene kot nedefinirane, da se lahko uporabljajo v ta namen)

Pasti

- Sprožijo se, ko procesor ne more normalno delovati zaradi napake ali napačnih podatkov.
- Past se lahko obravnava podobno kot prekinitev - sprožijo se enaki mehanizmi:
 - push vseh registrov na stack
 - izvajanje prekinitvenega servisnega podprograma
 - ...
 - le da jo sproži programska oprema namesto strojne.
- Primeri sprožitve pasti:
 - deljenje z 0
 - breakpoint
 - neznan ukaz

Pasti – INT X

- Večina CPE ima posebne ukaze za proženje pasti za debugger
- Sam ukaz INT X sproži programsko prekinitev, kjer X predstavlja prekinitev, ki naj se sproži (0-255).
 - Npr., ukaz INT 0x21 (33 v desetiškem sistemu) bo PC nastavil na 34. vektor v prekinitveni tabeli
- Ene izmed bolj znanih prekinitev na x86 so:
 - INT 0x21 - MS-DOS API call
 - INT 0x80 - Unix sistemski klic
 - INT 3 – Namenjen razhroščevalnikom.
 - Zapiše se le z enim bajtom - njegov opcode je 0xCC, čeprav se načeloma INT X zapiše z dvema bajtoma, torej 0xCD 0x03. Ker so nekateri ukazi na x86 lahko dolgi samo en bajt, s tem ob nastavitvi prekinitvene točke ne povozimo še drugih ukazov.

Izvedba breakpointa

- Ko CPE sproži past in pokliče OS, razhroščevalnik:
 1. Zamenja past (INT 3) s prvotnim ukazom na tistem mestu
 2. PC zmanjša za 1, ker se je po izvedbi pasti premaknil za eno predaleč
 3. Poda nadzor uporabniku, in ta lahko vidi vrednosti spremenljivk, klicni sklad, itd.
 4. Če uporabnik ne odstrani prekinitvene točke na tem mestu, razhroščevalnik na to mesto past spet doda.

(Ker mora najprej izvesti še ta ukaz, jo najprej doda na naslednji ukaz, se s tem pri naslednjem ukazu ustavi, in jo zdaj nastavi na pravi ukaz, naslednjega pa spet nadomesti s prvotno kodo in izvede).
- Primer v `examples/ptrace_setting_breakpoint_example.md`

Conditional breakpoint

- Izvedejo se le ob izpolnitvi danih pogojev
- Navadne prekinitvene točke, nato pa preveri, če se pogoji ujemajo danim podatkom, in le v tem primeru poda nadzor uporabniku
- Počasno za remote debugger...
 - prenos iz ciljne naprave na uporabnikovo napravo predstavlja veliko overheada, zato lahko pogoje preko gdb stuba preverimo kar na ciljni napravi, ali pa uporabimo interpreter, naložen kot deljen objekt znotraj ciljnega programa.

Software breakpoints

- To, kar smo si pogledali do zdaj
- Ni omejitve v številu
- Potrebujemo spremeniti program
 - lahko je nevarno
 - memory write access

Debug hardware

- Strojna oprema lahko pomaga
 - razhroščevanje z ustavljanjem procesorja (halting mode debugging)
 - enostavno izvajanje korak za korakom (single stepping)
 - strojna podpora za breakpointe in watchpointe
- Drago glede na prostor na siliciju

Hardware breakpoint

- Komparator, ki spremlja PC in primerja z programsko vneseno vrednostjo
- Če se ujema, fetch exception -> SIGSEV ali SIGTRAP
- Lahko dostopamo s posebnimi ukazi
- Omejeni (na x86 npr. 4)

Watchpoint

- Prekinitvene točke v spominu
- Pisanje, branje, ali oboje
- Komparator, ki spremlja vodilo z naslovom dostopa do spomina

Halting mode debugging

- Procesor ustavi izvajanje, ustavi uro
- DMA, RTC, še vedno teče
- Preko posebnega kanala lahko dostopamo do cevovoda in vstavljamo ukaze
- Lahko dostopamo do posebnih sistemskih registrov
- Do spomina lahko dostopamo z load in store ukazi -> čez predpomnilnik, pravilne vrednosti spremenljivk
- Veliko dela z pravilnim nadziranjem cevovoda, predpomnilnika...
- Lahko pa dostopamo direktno na vodilo, a lahko dobimo napačne vrednosti zaradi predpomnilnika

Debug information

- Za razhroščevanje v višjih jezikih potrebujemo preslikavo iz izvorne kode v strojno kodo
- Potrebujemo dodatne informacije, o funkcijah, spremenljivkah...

DWARF

- Dodatne informacije v datoteki ELF
- Primer: `examples/dwarf_debug_info.md`
- Optimizacija še vedno lahko prinese veliko težav

Call stack (backtrace)

- Stack unwinding
- Z uporabo Frame Pointerjev lahko dobimo informacije o klicih funkcije do trenutne vrstice
- Pri pretirani optimizaciji spet lahko pride do težav
- (gdb) bt

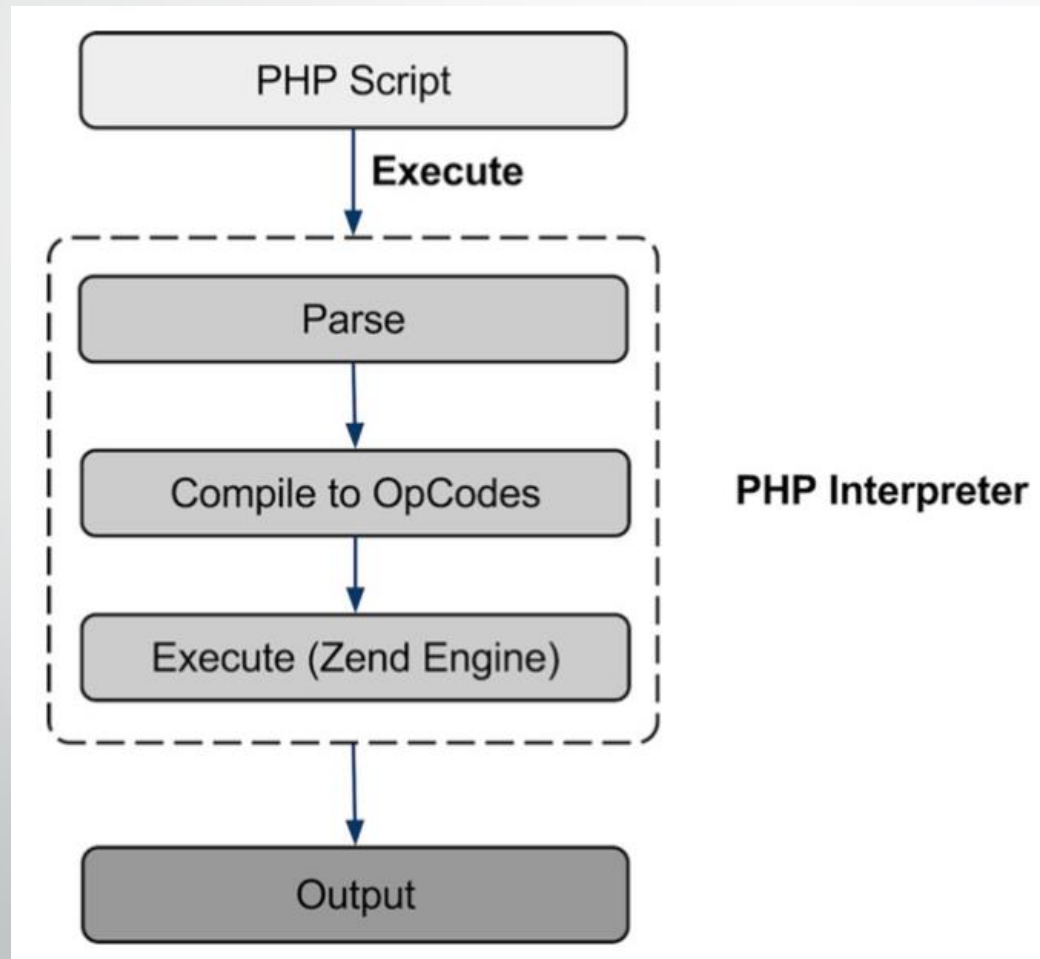
```
#0 c () at c.c:5  
#1 0x00008388 in b () at b.c:5  
#2 0x00008374 in a () at a.c:5
```

```
c.c:3: int c(void)  
4: {  
5:     return 1;  
6: }  
  
b.c:3: int b(void)  
4: {  
5:     return c();  
6: }  
  
a.c:3: int a(void)  
4: {  
5:     return b() + 1;  
6: }
```

Xdebug

- Remote debugger
- 1 uporablja podoben protokol GDB
- 2 pa nov, DBGP
- <https://xdebug.org/docs/remote>

PHP Execution Life Cycle



Kako deluje?

- Demo 😊

REFERENCE

- Jonathan B. Rosenberg. How Debuggers Work: Algorithms, Data Structures, and Architecture. John Wiley & Sons. ISBN 0-471-14966-7.
- [Debuggers - Wikipedia](#)
- [How do debuggers \(really\) work - Video](#)
- [How debuggers work - Part 1](#)
- [How debuggers work - Part 2 - Breakpoints](#)
- [How debuggers work - Part 3 - Debugging information](#)
- [GDB The GNU Project Debugger](#)
- [GDB - Wikipedia](#)
- [man 2 ptrace](#)
- [INT \(x86 instruction\) - Wikipedia](#)
- [Writing a linux debugger](#)