Jak (bezboleśnie) wrócić do kodowania na Amigę?

Krystian Bacławski

8 czerwca 2013

Plan seminarium

1. Ustawienia WinUAE.
2. Narzędzia dla AmigaOS.
3. Port szeregowy.
4. Prezentacja: kilka złych programów.
5. Środowisko programisty.
6. Kompilacja programów i bibliotek.
7. Narzędzia pomocnicze.
8. Amiga GCC: sztuczki i kruczki.
9. Plan rozwoju.
10. Literatura i odnośniki

Ustawienia WinUAE

Przegląd:

- procesor: `68040/060` + `MMU` (JIT wył.),
- serial port: `localhost:1234` (po TCP/IP),
- katalogi jako dyski (projekt widziany przez emulator i maszynę na której budujemy).

Emulacja MMU sporo spowalnia WinUAE, ale niezastąpione przy debugowaniu - empirycznie w okolicach szybkiej 68040.

Narzędzia dla AmigaOS

1. **MuLib**: 680x0.library, mmu.library
2. **ViNCEd**: konsola (zamiast KingCON)
3. **Csh**: najlepszy amigowy shell
4. **SnoopDos**: odpowiednik uniksowego strace
5. **SegTracker**: pamięta właścicieli hunków
6. **BlowUp**: treściwe Guru Meditation
7. **PatchWork**: błędne wywołania biblioteczne
8. **MuTools**: narzędzia wymagajace MMU
   - **MuForce**: wykrywa błędne dostępy do pamięci
   - **MuGuardianAngel**: problemy z allokacją pamięci (zamiast MungWall albo Wipeout)

Zastosowania portu szeregowego?

1. Odbieranie komunikatów od narzędzi do debugowania:
   - BlowUp
   - PatchWork
   - MuForce
   - MuGuardianAngel
2. `Printf` debugging: debug.lib
3. Debuggery działające po serialu:
   - ROMWack (KS 2.04) lub SAD (KS 3.x),
   - COP,
   - GNU gdb (częściowo dzięki GDBStub z AROSa)

Jak skonfigurować port szeregowy?

Potrzebujemy PuTTY!

- Dla windowsiarzy: `raw`, `localhost:1234`.
- Dla uniksiarzy: tunelowanie (`remote`) portu na maszynę ze środowiskiem programisty.

Po stronie AmigaOS w Prefs / Serial:

- baud rate: 31250,
- 8n1.

Złe programy i ich debugowanie

1. **BlowUp**: Guru Meditation
   - dzielenie przez zero
   - pułapki i inne takie
2. **MuForce**: odczyt lub zapis
   - pod nieistniejące adresy (`0xDEADC0DE`)
   - indeksowanie tablicy pod adresem zero
3. **MuGuardianAngel**:
   - wszelkie błędy związane z pojedyńczymi blokami (AllocMem, AllocVec, AllocPooled, etc.)
4. **PatchWork:**
   - błędnie skonstruowane wywołania bibliotek systemowych

WinUAE ma wbudowany debugger!

Dostępny pod skrótem `SHIFT+F12`:

- podglądanie stanu rejestrów sprzętowych,
- wykonywanie krokowe,
- przeskakiwanie przez funkcje,
- etc.

Ale lepiej służy jako trainer :-(

Środowisko kompilacji

Potrzebujemy:

- kompilatora C,
- asemblera,
- linkera,
- narzędzi pomocniczych (disassembler, etc.),
- standardowej biblioteki języka C,
- bibliotek AmigaOS.

Będziemy używać m68k-amigaos-toolchain!
Proces instalacji tutaj.

Kompilacja

```
CC = m68k-amigaos-gcc -noixemul
AS = vasm -quiet
CFLAGS = -Wall -O2
ASFLAGS = -Faout -phxass
```

`m68k-amigaos-gcc`: ta sama komenda robi za kompilator C, asembler i konsolidator

`vasm`: tryb kompatybilności z `phxass` (`AsmOne`) i obiekty obsługiwane przez linker

Kompilacja, c.d.

Prosty program:

```
$(CC) $(CFLAGS) -o hello hello.c
```

Obiekt:

```
$(CC) $(CFLAGS) -c -o test.o test.c
```

Konsolidacja:

```
$(CC) -o prog a.o b.o c.o -lmy
```

Asembler:

```
$(AS) $(ASFLAGS) -o test.o test.s
```

Kompilacja, c.d.

Pliki nagłówkowe:

```
$(CC) $(CFLAGS) -Iincdir [...]
```

Biblioteki:

```
$(CC) $(CFLAGS) -Llibdir -lmy [...]
```

Budowanie biblioteki statycznej:

```
m68k-amigaos-ar cr libmy.a a.o b.o
m68k-amigaos-ranlib libmy.a
```

Hello world bez printf

```
#include <inline/exec.h>
#include <dos/dos.h>
#include <inline/dos.h>
#include <workbench/workbench.h>

int __nocommandline=1; /* main() bez pobierania argumentów z CLI */
int __initlibraries=0; /* nie otwieraj bibliotek automatycznie */

struct DosLibrary *DOSBase = NULL;
extern struct WBStartup *_WBenchMsg; /* uruchomione z WB? */

int main(void) {
 if (_WBenchMsg == NULL) {
  if ((DOSBase=(struct DosLibrary*)OpenLibrary("dos.library",37))) {
   Write(Output(),"Hello world\n",12);
   CloseLibrary((struct Library *)DOSBase);
  }
 }
 return 0;
}
```

Łączenie z kodem w asemblerze

```
; eksportujemy symbol
; tak żeby linker go widział
XDEF _c2p1x1_8_c5_bm

    section c2p1x1_8_c5_bm, code

; d0.w  chunkyx [chunky-pixels]
; d1.w  chunkyy [chunky-pixels]
; d2.w  offsx [screen-pixels]
; d3.w  offsy [screen-pixels]
; a0    chunkyscreen
; a1    BitMap

_c2p1x1_8_c5_bm:
    ...
    rts
```

```
#ifndef __C2P_H__
#define __C2P_H__

#include <graphics/gfx.h>

void c2p1x1_8_c5_bm(
  UBYTE *chunky asm("a0"),
  struct BitMap *bitmap asm("a1"),
  UWORD width asm("d0"),
  UWORD height asm("d1"),
  UWORD offsetX asm("d2"),
  UWORD offsetY asm("d3"));

#endif
```

Narzędzia pomocnicze

1. **[P] m68k-amigaos-objdump**:
   - disassemblacja: `-d`
   - sekcje: `-h`
   - relokacje: `-r`
2. **[P] m68k-amigaos-nm**: symbole lokalne, eksportowane lub zewnętrzne
3. **[P] GccFindHit**: znajdź linię z błędem
4. **Hunk2aout**: konwersja bibliotek na `a.out`
5. **sfdc**: gen. plików nagłówkowych `inline`
6. **dumphunks**: wyświetl strukturę `AmigaHunk`
7. **dumpaout**: wyświetl strukturę `a.out`

Przypomnienie AmigaOS ABI

- `d0/d1/a0/a1/fp0/fp1`: scratch registers
- `a4`: base register
- `a5`: frame pointer (można zrobić backtrace)
- `a6`: library pointer (do bieżącej biblioteki)
- `a7`: stack pointer

Uwagi:

- `a4`: tylko dla `-fbaserel`, trzeba uważać i czasem użyć `__saveds`
- `a5`: `-fomit-frame-pointer` wył. użycie

GCC: opcje kompilacji

- `-fbaserel`: sekcja danych adresowana względem rejestru `a4`, max. 64kB
- `-fbaserel32`: j.w. ale min. 68020, bez ograniczenia na wielkość,
- `-msmall-code`: wszystkie skoki względem rejestru `pc`, kod wielkości max. 32kB.
- `-m68060`, `-m68020-60`, `-m68881`: generowanie kodu na konkretny procesor
- `-O2`, `-O3`, `-Os`: rozmiar vs. szybkość kodu

GCC: atrybuty zmiennych i funkcji

- `__regargs`: iff.c
- **`__chip (GNU as nie trawi)`**
- `__saveds`: input.c
- `__interrupt`: vblank.c
- `__amiga_interrupt`: copper.c
- `asm(x) x="a?"/"d?"/"fp?"`: c2p.h

GCC: predefiniowane makra

```
-Damiga
-Dmc68020
-Dmc68060
-D__HAVE_68881__
```

```
#ifdef __HAVE_68881__
 y = sin(x);
#else
 y = IEEEDPSin(x)
#endif
```

GCC: wywołania systemowe

Jak wołać funkcje z bibliotek?

1. otworzyć bibliotekę lub użyć `-lauto`
2. `#include <proto/dos.h>`
3. `Read(fh, ptr, length)`

Kompilator używa makr, a nie woła funkcje z `amiga.lib`. Poniższa opcja spowoduje, że kompilator **nie będzie** przeładowywał `a6` za każdym razem:

`-D__CONSTLIBBASEDECL__=const`

GCC: wstawki asemblerowe

```
static inline unsigned int
swap16(unsigned int a) {
  asm("swap %0": "=d" (a));
  return a;
}

#define swapr(a,b) {       \
 asm("exg %0,%1" :         \
     "+r" (a), "+r" (b)); \
}
```

```
static inline float
fabsf(float x) {
  float value;

  asm("fabs%.x %1,%0"
      : "=f" (value)
      : "f" (x));

  return value;
}
```

`toolchain`: inne ciekawe ficzery

1. `libnix`:
   - tworzenie bibliotek (`library`) i urządzeń (`device`)
   - różne implementacje wejścia do `main()`
   - automatyczne otwieranie bibliotek
   - listy inicjalizatorów i deinicjalizatorów (funkcje wykonywane przed i za `main()`, z priorytetami)
2. `GNU ld`:
   - możliwość tworzenia skryptów linkera (`trackmo`)
   - tworzenie list obiektów
   - wskaźniki na początek i koniec sekcji: `code`, `data` i `bss`

`toolchain`: plany

1. `libnix`:
   - zgodność ze standardem `C99`
   - poprawić zgodność z `Kickstart 1.3` (`A500`!)
2. `GNU cc`:
   - zaimplementować braki w `3.4.6`
   - zrobić przymiarkę do `4.8.x`
   - odpalać testy poprawności
3. `GNU binutils`:
   - pełne wsparcie dla formatu Amiga Hunk
   - linkowanie z bibliotekami amigowymi (ALink, BLink)
4. `GNU gdb`:
   - umożliwić zdalne debugowanie (GDBStub)

Literatura i odnośniki

1. GCC compiler for AmigaOS/m68k
2. ADCD 1.2 i 2.1 (Amiga Developer CD)
3. Amiga Developer Docs
   a. Amiga RKM Devices Manual
   b. Amiga RKM Libraries Manual
   c. Amiga Hardware Reference Manual
   d. Includes and Autodocs
4. libnix - standard C library for amiga GCC
5. AmigaOS-only features of GCC
6. The Amiga Guru Book

Pytania ?