PB173 Linux

02 Linkovanie

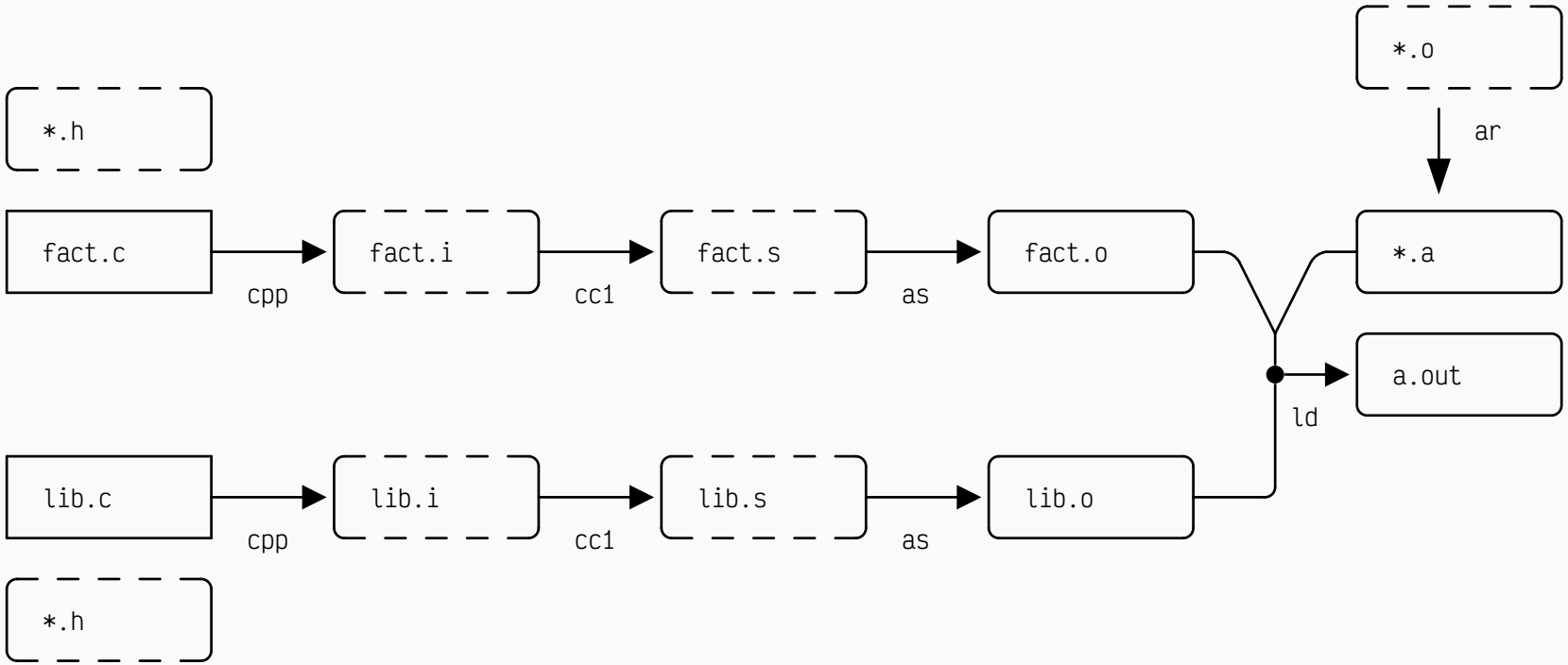
Roman Lacko xlacko1@fi.muni.cz

2022-09-23

Obsah

Linkovanie

Linker


*.o
ar
*.h
fact.c
cpp
fact.i
cc1
fact.s
as
fact.o
*.a
a.out
ld
lib.c
cpp
lib.i
cc1
lib.s
as
lib.o
*.h

Linker

Linker

Program, ktorý spája objektové súbory a archívy do spustiteľného kódu.

Linker: Objektový súbor

- Výsledok prekladu súboru so symbolickými inštrukciami.
- Môže obsahovať odkazy na objekty v iných súboroch
  - v štandardnej knižnici (`printf()`, `stdout`, ...)
  - v inom objektovom súbore (funkcia z nášho `X.h`).

Linker: Objektový súbor

Program `nm` vypíše symboly v objekte.

```
$ nm hello.o
0000000000000000 T main
                 U puts
```

Linker: Objektový súbor

Program `objdump` vypíše rôzne vlastnosti objektu, napríklad kód.

```
$ objdump --disassemble hello.o
0000000000000000 <main>:
   0:   55                          push   %rbp
   1:   48 89 e5                    mov    %rsp,%rbp
   4:   48 8d 05 00 00 00 00        lea    0x0(%rip),%rax
   b:   48 89 c7                    mov    %rax,%rdi
   e:   e8 00 00 00 00              call   13 <main+0x13>
  13:   b8 00 00 00 00              mov    $0x0,%eax
  18:   5d                          pop    %rbp
  19:   c3                          ret
```

S `--disassemble-all` môžete vidieť aj `.data` sekcie.

Linker: Prológ a epilóg

Do výsledného binárneho kódu sa okrem vstupných objektov prikladajú aj ďalšie časti:

Prológ Inicializuje prostredie pre program.

Epilóg Upratanie tesne pred koncom programu.

Štandardná knižnica
Implementácia štandardných funkcií.
Môže sa vynechať.

a prípadne iné knižnice alebo dáta.

API, ABI

API: Application Programming Interface

Vysokoúrovňový pohľad na rozhranie:

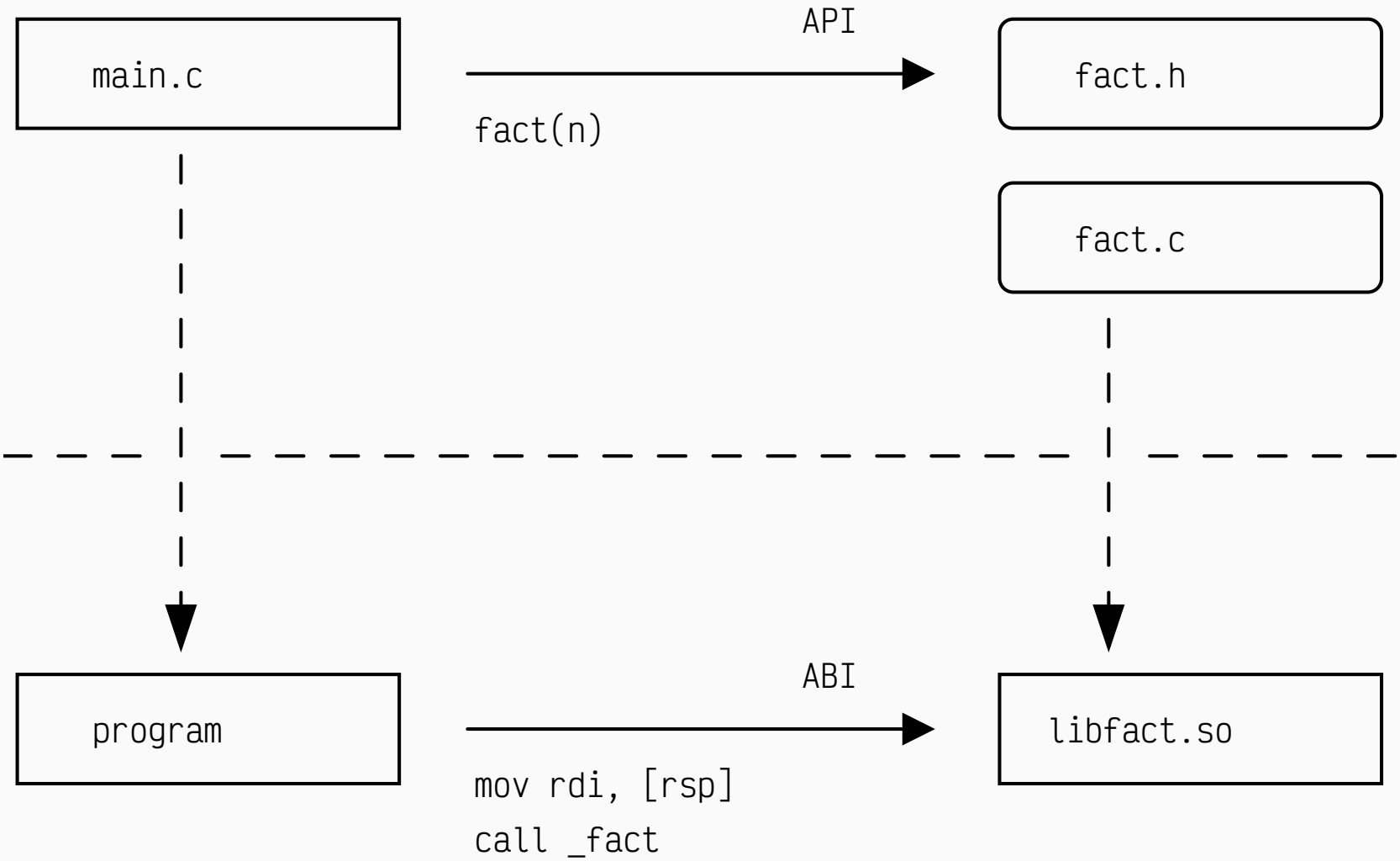
- vysokoúrovňový popis komunikácie
  medzi časťami aplikácie,
- typicky na úrovni jazyka (napr. deklarácia funkcie).

ABI: Application Binary Interface

Nízkoúrovňový pohľad na rozhranie:

- popis volacích konvencií,
- použité registre, rozloženie dát, zarovnanie, ...

API vs ABI


main.c
API
fact.h
fact(n)
fact.c
program
ABI
libfact.so
mov rdi, [rsp]
call _fact

API vs ABI

Dopad zmien vo vývoji kódu môžeme charakterizovať ako

API-kompatibilná zmena

Zmena v komponente, ktorá nevyžaduje zmenu kódu iných komponentov.

ABI-kompatibilná zmena

Zmena v komponente, ktorá nevyžaduje kompiláciu iných komponentov.

Spätne kompatibilná zmena

API a ABI kompatibilná zmena, ktorá zachováva kontrakt.

Nekompatibilné zmeny naopak uvedený zásah vyžadujú.

Knižnice

Knižnice: Modulárne programovanie

Znovupoužiteľné komponenty programu:

- jednoduchší vývoj,
- jednoduchšie testovanie a oprava chýb,
- možnosť verzovania.

Statické knižnice

Linkovanie za kompilácie, čím sa objekty stanú súčasťou programu.

- Obvykle zložené z viacerých objektov.
- Pre pohodlnosť v archíve s príponou `.a`, ktoré vyrába program `ar`.

```
$ ar rsv libfi.a despair.o depression.o fear.o misery.o
$ nm libfi.a
…
$ gcc ... -L. -lfi
```

Statické knižnice

Výhody:

- Program bez závislostí.

Nevýhody:

- Väčší program.
- Pri chybe v knižnici je treba kompilovať znova všetky statické programy, ktoré ju používajú.

Dynamické knižnice

Spustiteľný kód pripojiteľný k programu **za behu**.

- Prípona `.so` ako *shared object*.
- Ďalšie prípony pre verzie.
- Typicky pripojené pri spustení programu,
  môžu sa pripájať aj na požiadanie programu (pluginy).

Závislosti na knižniciach nájde program `ldd`:

```
$ ldd hello
linux-vdso.so.1 (0x00007ffda5c90000)
libc.so.6 => /lib64/libc.so.6 (0x00007fe2786b2000)
/lib64/ld-linux-x86-64.so.2 (0x00007fe2788cd000)
```

Dynamické knižnice: Position Independent Code

Rovnaký kód môže používať viac programov naraz.
Kód teda musí fungovať nezávisle na umiestnení.

```
$ gcc -fPIC -c -o hell.o hell.c
```

Dynamické knižnice: Linkovanie

Linkovanie PIC objektov do knižnice vyžaduje:

- prepínač `-shared`,
- logický názov knižnice, obvykle s verziou API

```
$ gcc -shared -Wl,-soname,libhell.so.1 \
    -o libhell.so hell.o
```

Prepínače pre linker

Prepínač `-Wl` podá čiarkami oddelené parametre linkeru:

```
$ ld -soname libhell.so.1 …
```

Dynamické knižnice: Verzovanie

Sémantické verzovanie

- Najrozšírenejšia schéma verzovania.
- Verzie tvaru `MAJ.MIN[.REV[...]][-SUF]`, pričom obvykle sa zvyšuje

  `MAJ` pri nekompatibilnej zmene API

  `MIN` pri spätne kompatibilnej zmene
  napr. pridanie funkcionality

  `REV, ...` pri plne kompatibilnej zmene
  napr. oprava chyby

Dynamické knižnice: Názvy

Dynamické knižnice majú tri názvy:

Real Name

- názov súboru s kódom knižnice
- typicky `libNAME.so.MAJ.MIN`

Shared Object Name (soname)

- logický názov knižnice
- typicky `libNAME.so.MAJ`

Linker Name

- názov, ktorý použije linker pri `-lLIB`
- typicky `libNAME.so`

Dynamické knižnice: Názvy

Názvy na seba v súborovom systéme často odkazujú

```
$ ls -l /usr/lib64/libpcap*
/usr/lib64/libpcap.so -> libpcap.so.1
/usr/lib64/libpcap.so.1 -> libpcap.so.1.10.1
/usr/lib64/libpcap.so.1.10.1
```

O nastavenie odkazov sa obvykle stará `ldconfig`.

Dynamické knižnice: Dynamický linker

Dynamicky linkované programy nemôžu bežať samostatne:

```
$ readelf --program-headers games/halflife3
INTERP 0x0000000000000318 0x0000000000000318 …
       0x000000000000001c 0x000000000000001c …
       [Interpreter: /lib64/ld-linux-x86-64.so.2]
```

Po spustení programu (`exec*()`) sa vykonajú kroky:

1. Hlavičky programu sa načítajú do pamäte.
2. Načíta sa a spustí dynamický linker (z `.INTERP` hlavičky).
3. Linker prečíta `.dynamic` a dopĺňa symboly z `.dynsym`.
4. Predá riadenie programu symbolu `_start`.

Dynamické knižnice: Dynamický linker

Dynamický linker je možné konfigurovať premennými prostredia.

`LD_LIBRARY_PATH`
: Kde hľadať dynamicky linkované knižnice.

`LD_PRELOAD`
: Zoznam knižníc, ktoré sa majú vložiť pred ostatnými a bezpodmienečne.

`LD_DEBUG`
: Ladenie dynamického linkera.
nápoveda: `LD_DEBUG=help program`

Dynamické knižnice: Konštruktor a deštruktor

Špeciálne označené funkcie, ktoré sa spustia počas prológu resp. epilógu.

```
__attribute__((constructor))
static void bonjour(void)       // Name does not matter.
{ … }

__attribute__((destructor))
static void au_revoir(void)
{ … }
```

Dynamické knižnice

Výhody:

- Menšie programy.
- Efektívnejšie využitie pamäte počítača.
- Jednoduchšie záplaty chýb v knižniciach.

Nevýhody:

- Trochu pomalší štart programu.
- PIC môže byť (skoro zanedbateľne) pomalší.
- Občas problém so závislosťami pri aktualizácii systému.

Pluginy

Pluginy

V širšom význame rozšírenie programu
bez nutnosti jeho úpravy.

V kontexte Linuxu je to dynamická knižnica,
ktorú linker pripojí na požiadanie programu.

Na rozdiel od dyn. knižníc majú typicky
len *Linker Name* (resp. ostatné sú zhodné s týmto).

Pluginy: API

```c
#include <dlfcn.h>      /* Link with '-ldl'! */

void *dlopen(const char *filename, int flags);
int dlclose(void *handle);

void *dlsym(void *handle, const char *symbol);
char *dlerror(void);
```

Špeciálne hodnoty `handle` pre `dlsym()`:

`RTLD_DEFAULT` **(GNU)** Prehľadáva všetky objekty od začiatku do konca.

`RTLD_NEXT` **(GNU)** Prehľadáva objekty za aktuálnym objektom. Vhodné na prekrytie pomocou `LD_PRELOAD`.

Pluginy: Linkovanie

Nezabúdajte na `-ldl`!

Ak na tento prepínač zabudnete, program môže robiť podivné nezmysly, ktoré nerozdýcha ani debugger.

Pri linkovaní **programu** sa môže zísť `-Wl,-rpath,WHERE`, ktorý linkeru pridá adresár, kde hľadať objekty s relatívnou cestou.

`$ORIGIN` Prehľadá adresár s programom.

`$LIB` Prehľadá `/lib` resp. `/lib64` podľa architektúry.

`CESTA` Prehľadá zadanú cestu.

Knižnice a Make

Makefile: Ako linkovať

`LDFLAGS` Prepínače pre linker (`-Wl,...`, `-L...`)

`LDLIBS` Knižnice (`-lLIB`).

Špecialita GNU Make: Závislosť na knižnici.

```
program: $(OBJECTS) -lLIBRARY
```

Namiesto `-LLIBRARY` dosadí cestu ku knižnici.

Makefile: pkg-config (pkg-conf)

V prípade závislosti na knižnici tretej strany nemáme istotu, že sa nachádza na rovnakom mieste v každej distribúcii.

Tento problém rieši `pkg-config`:

```
CFLAGS += $(shell pkg-config --cflags libev)
LDFLAGS += $(shell pkg-config --libs libev)
```

Balík s knižnicou si musí pridať záznam (súbor) do `/lib{64}/pkgconfig` alebo `/usr/share/pkgconfig`.