

Conceitos Básicos sobre Programação Prática

Programa de computador

- conjunto de instruções e informação necessários ao alcance de um objectivo
 - instruções + dados
 - normalmente, guardados em ficheiros (durabilidade)
 - característico ou não de uma dada plataforma
 - plataforma: sistema operativo & cia. + processador & cia.
- quando em execução, designa-se **processo**

Processo

- actividade e recursos necessários ao cumprimento de um programa
- controlado pelo sistema operativo
- tem associado:
 - espaço de endereçamento (memória)
 - instruções (*text*)
 - dados (*data*)
 - zona de trabalho (*heap, stack*)
 - variáveis nas tabelas de sistema
 - recursos necessários (partilhados!...)

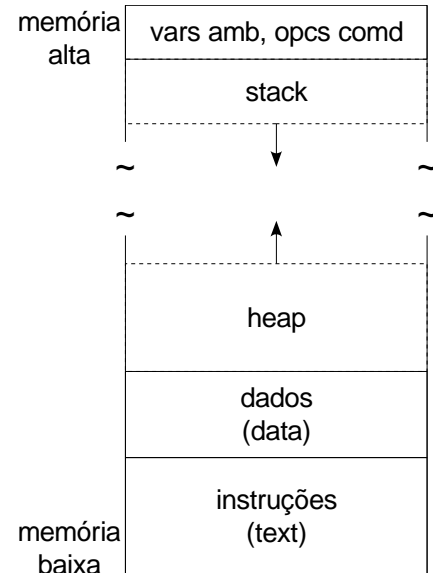


Fig. 1: Espaço de endereçamento típico de um processo Unix, executando um programa em C.

Código de Programa

- instruções e informação de dados guardados em ficheiros
- mas...
 - quem o faz?
 - quem o usa?
 - é compreensível?
 - é estruturado?
 - pode ser alterado?
 - onde é executado?
 - ...

Código: compreensão

- fontes
 - informação normalmente textual
 - > compreensível por quem conhecer a linguagem em que foi escrita (linguagem de programação)
 - podem ser utilizadas (executadas) “directamente” ou não
 - ter as fontes é “**possuir**” o programa
- binários
 - informação codificada na linguagem natural do processador
 - > só compreensível após descodificação
 - podem ser usados (executados) sozinhos ou necessitam de “acompanhamento” (código de *loading*)
 - ter os binários é “**poder usar**” o programa

Código: estrutura

- partes específicas (*programador!*)
 - declarações (e.g. ficheiros de inclusão, classes)
 - instruções (e.g. rotinas)
 - dados (e.g. variáveis pré-definidas)
- partes gerais (*sistema de desenvolvimento e execução*)
 - declarações -> ficheiros de inclusão (texto!)
 - Unix: `/usr/include/`
 - bibliotecas (*libraries*) -> arquivos binários!
 - estáticas
 - Unix: `/usr/lib/` (e.g. `g++ -static`)
 - C: `libc.a` (`a` = *archive*); C++: `libstdc++.a`
 - dinâmicas (ou partilhadas)
 - Unix: `/usr/lib/`
 - C: `libc.so` (`so` = *shared object*); C++: `libstdc++.so`
 - MSWindows: `*.dll` (`dll` = *dynamic link library*)

Bibliotecas dinâmicas partilhadas

- partilha de espaço de memória
- actualização automática de aplicações
 - limitações...
 - perigo de ruptura...
- ficheiro executável tem tamanho mínimo

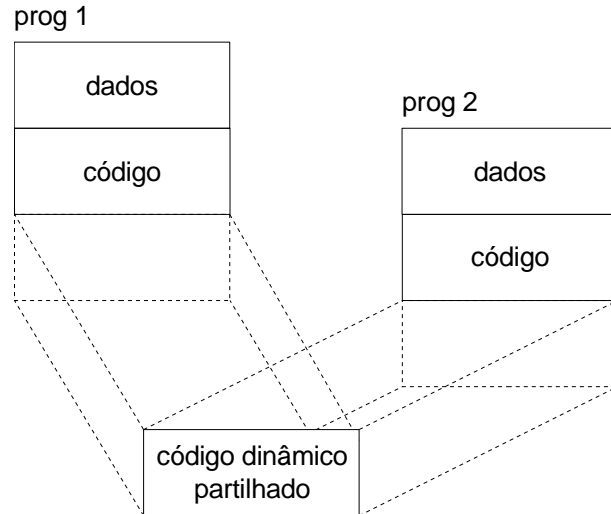


Fig. 2: Partilha de rotinas de bibliotecas dinâmicas por vários processos.

Construção de um programa

- obter requisitos, meditar, planear, escolher ambiente de programação
 - ferramenta base: cérebro, papel e lápis...
- escrever programa numa linguagem de programação -> **código fonte**
 - ferramenta base: editor de texto (e.g. kate)
- compilar -> **código objecto**
 - ferramentas base: compilador (máquina ou *bytecode*)
- gerar executável -> **código objecto de bibliotecas** (e.g. C, C++)
 - ferramenta base: *linker*
- executar -> **código fonte, “intermédio” (*bytecode*) ou máquina**
 - ferramentas base: *loader*, interpretador (*bytecode* ou fonte)
- corrigir programa -> **código fonte**
 - ferramentas base: cérebro e depurador (*debugger*)
- aperfeiçoar o programa -> **código fonte**
 - ferramentas base: *profiler* (e.g. gprof), *time* (Unix)

Execução de um programa:

- possuir:
 - código fonte -> **fontes interpretadas** (e.g. BASIC, Bourne shell)
 - código máquina -> **fontes compiladas** (e.g. C, C++)
 - código “intermédio” (*bytecode*) -> **fontes “semi-compiladas”** (e.g. JAVA)
- comandar:
 - interpretador de comandos (*shell*, normal ou gráfica)
 - sistema operativo
 - interpretador de linguagem
 - carregador (*loader*)
 - ligador dinâmico (*dynamic linker*)
 - computador (*hardware*)
 - processador, memória interna, barramentos, dispositivos E/S...
 - ... simulador (*software*)
- executar:
 - bash> a.out ; clicar no ícone do programa ; ...

Depuração de um programa (*debugging*):

- editor (e.g. kate)
 - alterar o código fonte
- ambiente de execução (e.g. PC com JVM - Java Virtual Machine)
 - interpretador / compilador
 - preparar e executar o programa
 - computador / simulador
 - fornecer os recursos ao programa
- depurador (e.g. gdb (texto) ; e.g. ddd (gráfico))
 - executa de forma controlada (saltos, paragens...)
 - revela os valores das variáveis
- cérebro
 - conhecimento, engenho, paciência, intuição

Ferramentas auxiliares

- ambiente de desenvolvimento (IDE – *Integrated Development Environment*)
e.g. Eclipse, KDevelop
- documentação
 - manuais gerais de utilização
 - manuais de referência (API – *Application Program Interface*)
 - `man`, `info` (e.g. `shell> man atoi`; e.g. `man:/atoi` (KDE))
 - biblioteca normalizada C; chamadas ao sistema
- construção programada (controlada)
 - e.g. `make` (FIG)
- ambiente de compilação
 - pré-processador (e.g. `g++ -E`)
 - `#define DOIS 2`
 - `#include <iostream.h> // em /usr/include`
 - assembler (e.g. `g++ -S`)
 - código *assembly*

make

- é um interpretador de “programas” que
 - estão num ficheiro de texto, *makefile*
 - usam uma linguagem própria (*~shell*)
 - blocos são semelhantes às receitas de culinária:
 - produto final, ingredientes, operações
 - funciona com base em
 - regras de dependência entre ingredientes e produtos
 - comparação de idades entre ingredientes e produtos
- pode ser usado em aplicações múltiplas
 - preparação e actualização de programas, documentação...
- utilização
 - `shell> make [existe Makefile]`
 - `shell> make -f fich-makefile`

...make (cont.)

```
# Makefile example. Two executables are to be created: ex1 , ex2
#
# Their source code is, respect.: ex1.c , ex2.c
# The sources use, respect.:(com.h , ex1.h) , com.h
#
all: ex1 ex2

ex1: ex1.o
    cc ex1.o -o ex1

ex2: ex2.o
    cc ex2.o -o ex2

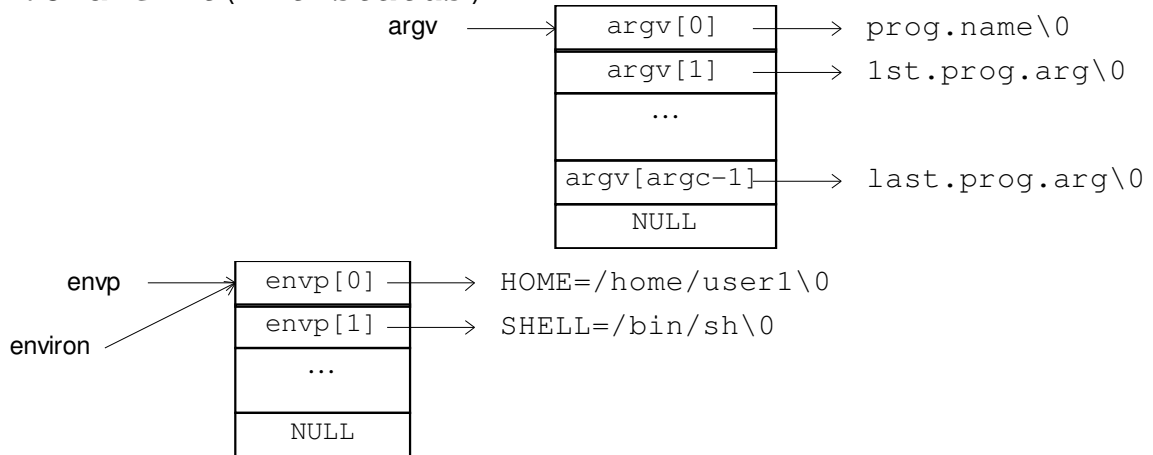
ex1.o: ex1.c com.h ex1.h
    cc -Wall -c ex1.c

ex2.o: ex2.c com.h
    cc -Wall -c ex2.c

clean:
    rm -f ex1 ex2 ex1.o ex2.o
```

Vida e morte de um programa em C:

- arranque
 - rotina de arranque em C!
 - `int main(int argc, char * argv[], char * envp[]);`
- terminação
 - `void exit(int status);`



...Vida e morte de um programa em C: (cont.)

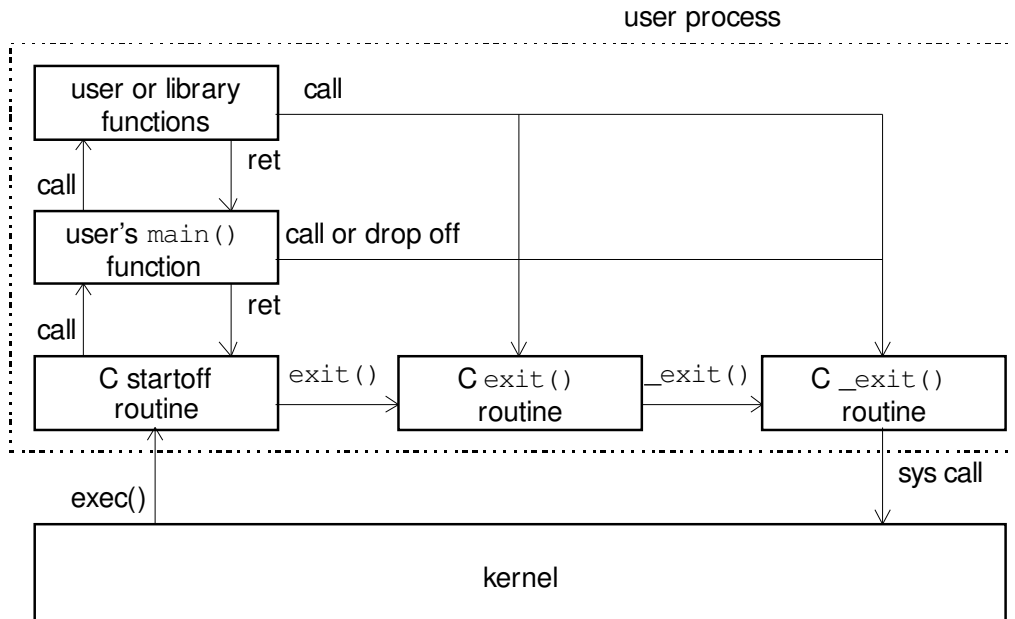


Fig. 3: Arranque e terminação de um programa em C.

Utilização do compilador e observação de aspectos diversos:

- criação de executáveis “estáticos” e “dinâmicos”
 - shell> g++ test.C -o teste
 - shell> g++ -static test.C -o test.stat
 - shell> ls -l test*
-rw-r--r-- 1 user grp 366 2007-03-27 15:18 test.C
-rwxr-xr-x 1 user grp 8335 2007-04-07 10:42 test
-rwxr-xr-x 1 user grp 1184013 2007-04-07 14:33 test.stat
- criação de código assembly
 - shell> g++ -S test.C -o test.asmb
 - shell> ls -l test.prec
-rw-r--r-- 1 user grp 4910 2007-04-07 14:34 test.asmb
- criação de código “pré-compilação” (depois do pré-processador)
 - shell> g++ -E test.C -o test.prec
 - shell> ls -l test.C test.prec
-rw-r--r-- 1 user grp 366 2007-03-27 15:18 test.C
-rw-r--r-- 1 user grp 694897 2007-04-07 14:34 test.prec

Mais alguns exemplos:

- TP0 de LSO (LEIC, 2001/02)
 - <http://web.fe.up.pt/~jmcruz/p2/cod/LSO-0102.tp0.depur-profil>
- TPC1 de SOC (LEIC, 2001/02)
 - <http://web.fe.up.pt/~jmcruz/p2/cod/SOC-0102.tpc1.assembly>
- Makefiles
 - <http://web.fe.up.pt/~jmcruz/etc/unix/make/makefile-ex.tar.gz>