Sistemas Operativos (MIEIC @ FEUP)

Programming basics[1]

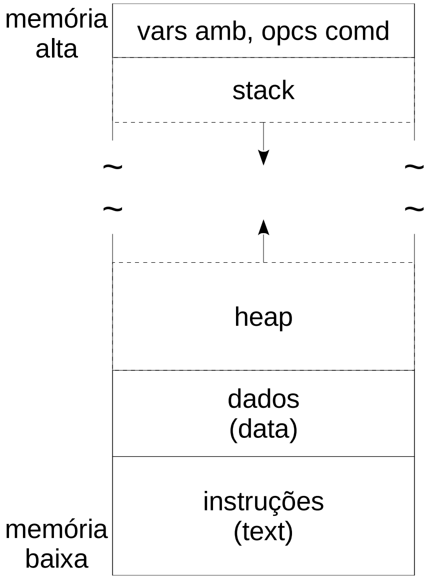
Computer program

- set of instructions and data necessary to meet a goal
  - instructions + data
  - usually stored in files (on disk -> durability)
  - it is associated or not to a given platform
    - platform: operating system et al. + processor et al.
- when running, it is called a **process**

1 **Disclaimer**: not counting lapses, the following material is mostly correct. :-)

Process

- activity and resource usage necessary to fulfill a program
- managed by the operating system
- has associated:
  - address space (memory)
    - instructions (*text*)
    - data
    - work zone (*heap*, *stack*)
  - variables in system tables
  - other necessary resources (shared ...)
    - communication ports
    - semaphores...



Typical address space layout of a Unix process, running a C program.

Program's code

- instructions and data kept in files
- but...
  - who writes (develops) it?
  - who uses it?
  - is it understandable?
  - is it structured?
  - can it be modified?
  - where is it run?
  - ...

Code: comprehension

- sources
  - usually, textual information
    - understandable by those who know the (programming) language in which it was written
  - can be used (run, executed) "directly" or not
  - having the sources is "owning" the program
- binaries
  - information encoded in the processor's natural language
    - understandable only after decoding
  - can be used (run) alone or with something more (loading code)
  - having the binaries is “being able to use” the program

Code: structure

- specific parts (*programmer*!)
  - declarations (e.g. inclusion files, classes)
  - instructions (e.g. routines)
  - data (e.g. preset variables)
- general parts (*development and running system*)
  - declarations -> inclusion files (text!)
    - Unix: `/usr/include/`
  - libraries -> binary files!
    - static
      - Unix: `/usr/lib/` (e.g. C: `libc.a` ; C++: `libstdc++.a`)
    - dynamic (or shared)
      - Unix: `/usr/lib/` (e.g. C: `libc.so` ; C++: `libstdc++.so`)
      - MSWindows: `*.dll`

.a --> archive
.so --> shared object
.dll --> dynamic link library

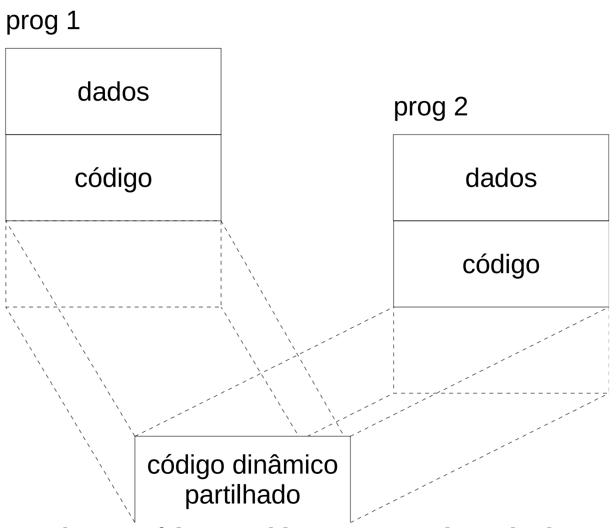
Code: shared dynamic libraries

Pro:

- sharing of memory space
- automatic application update
  - limitations...
  - danger of breakage...
- executable file has minimum size

Con:

- mobility...
- performance...



Sharing of dynamic library routines by multiple processes

Developing a program

- gather requisites, meditate, plan, choose a programming environment
  - basic tools: brain, paper and pencil...
- write program in a programming language -> **source code**
  - basic tools: text editor (e.g. `kate`)
- compile -> **object code** [1]
  - basic tools: compiler (machine code or bytecode)
- generate executable -> **object code from libraries** (e.g. C, C++)
  - basic tools: *linker*
- run -> **source, “intermediate” (*bytecode*) or machine code**
  - basic tools: *loader*, interpreter (*bytecode* or source)
- correct and improve -> **source code**
  - basic tools: brain, *debugger*, *profiler*, `time` (Unix)...

1 see initial **Disclaimer** and https://stackoverflow.com/questions/6889747/is-python-interpreted-or-compiled-or-both

Running a program

- have:
  - source code, *bytecode* or machine code
- command:
  - *shell* (text or graphical)
    - operating system
    - compiler or interpreter
    - loader
    - (dynamic) linker
    - hardware: processor, memory, bus, I/O devices...
  - ... *software simulator*
    - idem...

Debugging a program

- avoid the need for debugging altogether
  - program carefully, test for error in return values, verify source code's syntax
- prepare execution environment
  - interpreter / compiler + computer / simulator
- edit source code
  - make corrections or prepare code for debugging
- use debugger (e.g. `gdb` (text) , `ddd` (graphical))
  - controlled execution (breaks, jumps...)
  - revelation of variable's values
- use brains
  - knowledge, ingenuity, patience, intuition

Auxiliary Tools

- (integrated) development environment (IDE) e.g. Eclipse
- documentation
  - general user manuals
  - reference manuals (API – *Application Program Interface*)
    - `man`, `info` (e.g. `man atoi`)
- programmed building
  - e.g. `make` (see below)
- compilation environment
  - preprocessor (e.g. `g++ -E`)
    - `#include <iostream.h>` ; `#define TWO 2`
  - assembler (e.g. `g++ -S`)
    - *assembly* code

`make`

- is an interpreter of "programs"; these, should:
  - be in a text file, *makefil*e
  - use a very special language (similar to shell's)
  - be written in blocks similar to culinary recipes:
    - final dish, ingredients, cooking instructions
  - operates based on
    - dependency rules between ingredients and products (dishes)
    - comparison of age between ingredients and products (dishes)
- is mainly used for repeated tasks
  - preparation, update and installation of programs, documentation...
- usage
  - `shell> make`
  - `shell> make -f makefile-name`

...make (cont.)

```
# Makefile very simple example.
#
# Two executables are to be created: exe exe.stat
#
# Their source code is common: exe.c
#    which uses (includes): exe.h

all: exe exe.stat

exe: exe.c exe.h
     cc exe.c -o exe

exe.stat: exe.c exe.h
     cc -static exe.c -o exe.stat


clean:
     rm -f exe exe.stat
```

Interesting examples of compilations

- generation of "static" and "dynamic" executables

  -
    ```
    shell> make hello
    ```

  -
    ```
    shell> make hello.stat
    ```

  -
    ```
    shell> ls -l hello*
    -rwxrwxr-x 1 user grp   8304 fev  2 17:06 hello
    -rw-r--r-- 1 user grp     75 fev  2 17:02 hello.c
    -rwxrwxr-x 1 user grp 845120 fev  2 17:06 hello.stat
    ```
- generation of *assembly* code

  -
    ```
    shell> make hello.asm
    ```

  -
    ```
    shell> ls -l hello.asm
    -rw-rw-r-- 1 usr grp     463 fev  2 17:16 hello.asm
    ```
- generation of “pre-compilation” code (after preprocessor)

  -
    ```
    shell> make hello.prec
    ```

  -
    ```
    shell> ls -l hello.prec
    -rw-rw-r-- 1 user grp  17968 fev  2 17:19 hello.prec
    ```

Source code of compilation examples

hello.c

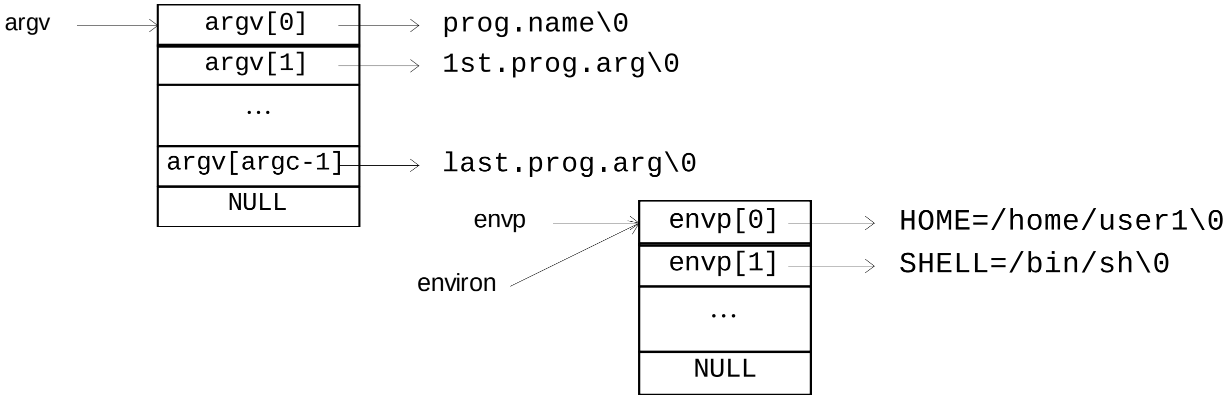
```
#include <stdio.h>

int main() {
      printf("\nHello World!\n");
      return 0;
}
```

makefile

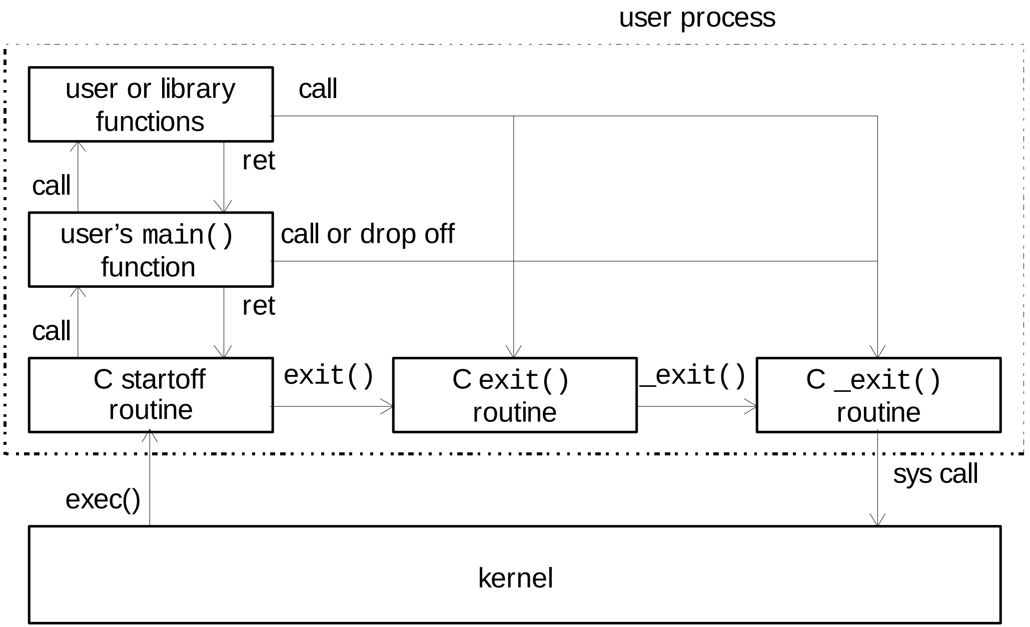
```
all: hello hello.stat hello.asm hello.prec

hello: hello.c
      cc -Wall -o hello hello.c
hello.stat: hello.c
      cc -Wall -static -o hello.stat hello.c
hello.asm: hello.c
      cc -Wall -S -o hello.asm hello.c
hello.prec: hello.c
      cc -Wall -E -o hello.prec hello.c
clean:
      rm -f hello hello.stat hello.asm hello.prec
```

"Life and death" of a C program

- beginning
  - startup code in C!
  - `int main(int argc, char *argv[], char *envp[]);`
- ending
  - `void exit(int status);`



..."Life and death" of a C program



The beginning and ending of a C.