

Programação orientada a objetos em C puro: o caso do htop

Hisham Muhammad
h@hisham.hm
[@hisham_hm](https://github.com/hisham_hm)

FISL 16, 2015

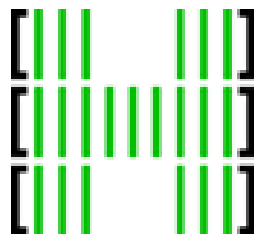
Quem sou eu

Hisham Muhammad - h@hisham.hm



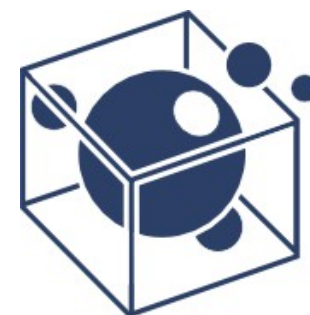
GoboLinux

gobolinux.org



htop

hisham.hm/htop



LuaRocks

luarocks.org

O htop

The screenshot shows the htop interface with the following summary statistics:

- Tasks: 78 total, 1 running
- Load average: 0.17 0.19 0.08
- Uptime: 6 days, 19:50:19
- Mem: 148/500MB
- Sup: 6/511MB

The process list table is as follows:

| PID | USER | PRI | NI | VRT | RES | SHR | S | CPU% | MEM% | Command |
|-------|--------|-----|----|-------|-------|-------|---|------|------|---|
| 1 | gobo | 16 | 0 | 1360 | 504 | 1328 | S | 0.0 | 0.1 | init [3] |
| 271 | gobo | 17 | 0 | 1460 | 636 | 1348 | S | 0.0 | 0.1 | - devfsd /System/Kernel/Devices |
| 942 | gobo | 17 | 0 | 3080 | 1220 | 2812 | S | 0.0 | 0.2 | - /bin/sshd |
| 2978 | gobo | 16 | 0 | 1356 | 328 | 1316 | S | 0.0 | 0.1 | - /Programs/Runit/Current/sbin/runsvdir |
| 2995 | gobo | 16 | 0 | 1344 | 288 | 1320 | S | 0.0 | 0.0 | - runsv Klogd |
| 2997 | gobo | 15 | 0 | 1484 | 448 | 1320 | S | 0.0 | 0.1 | - klogd -n |
| 2996 | gobo | 16 | 0 | 1344 | 288 | 1320 | S | 0.0 | 0.0 | - runsv Syslogd |
| 2998 | gobo | 16 | 0 | 1540 | 600 | 1368 | S | 0.0 | 0.1 | - syslogd -n -m 0 |
| 2981 | gobo | 16 | 0 | 1352 | 472 | 1312 | S | 0.0 | 0.1 | - /sbin/agetty tty3 9600 |
| 2982 | gobo | 16 | 0 | 1352 | 472 | 1312 | S | 0.0 | 0.1 | - /sbin/agetty tty4 9600 |
| 2983 | gobo | 16 | 0 | 1352 | 472 | 1312 | S | 0.0 | 0.1 | - /sbin/agetty tty5 9600 |
| 2984 | gobo | 16 | 0 | 1352 | 472 | 1312 | S | 0.0 | 0.1 | - /sbin/agetty tty6 9600 |
| 3577 | news | 16 | 0 | 99M | 2424 | 99448 | S | 0.0 | 0.4 | - /Programs/INN/2.4.1/bin/immd -p 3 |
| 3578 | news | 21 | 4 | 3932 | 1396 | 2532 | S | 0.0 | 0.2 | - /System/Links/Executables/perl -w |
| 15956 | gobo | 16 | 0 | 1348 | 468 | 1312 | S | 0.0 | 0.1 | - /sbin/agetty tty2 9600 |
| 19019 | hisham | 15 | 0 | 5232 | 1920 | 4768 | S | 0.0 | 0.3 | - -zsh |
| 19178 | hisham | 20 | 0 | 4504 | 1292 | 4268 | S | 0.0 | 0.2 | - /bin/sh /System/Links/Executables/ |
| 19190 | hisham | 15 | 0 | 2192 | 628 | 2144 | S | 0.0 | 0.1 | - xinit /Users/hisham/.xinitrc - |
| 19191 | gobo | 15 | 0 | 91872 | 23312 | 78164 | S | 1.3 | 3.8 | - X :0 |
| 19201 | hisham | 16 | 0 | 4504 | 1312 | 4268 | S | 0.0 | 0.2 | - /bin/sh /System/Links/Exec |
| 19251 | hisham | 16 | 0 | 1352 | 312 | 1312 | S | 0.0 | 0.1 | - kwrapper kmsserver |

Navigation menu: 1Help 2Setup 3Search 4Invert 5Tree 6SortBy 7Nice - 8Nice + 9Kill 10Quit

<http://hisham.hm/htop/>

Por que C?

portabilidade

de hardware? de sistema? de distro?

desempenho

desempenho CPU – footprint de RAM

controle

sobre a execução – sobre a memória

Qual C?

K&R C (1978)

ANSI C (1989)

ISO C99 (1999)

ISO C11 (2011)



Qual C?

K&R C (1978)

ANSI C (1989)

ISO C99 (1999)

ISO C11 (2011)



Por que não C++?

"C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do it blows your whole leg off."

Bjarne Stroustrup, criador de C++

Dependências do htop



build:

Autotools
(Automake, Autoconf...)

runtime:

NCurses

Abordagem de desenvolvimento

código simples

o mais alto-nível possível

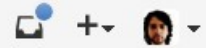
pagar o preço
somente das features
que usarmos

github.com/hishamhm/htop



This repository Search

Pull requests Issues Gist



hishamhm / htop

Unwatch 42

Star 460

Fork 91

htop is an interactive text-mode process viewer for Linux. It aims to be a better 'top'. — Edit

534 commits

3 branches

10 releases

12 contributors



branch: master

htop / +



Merge pull request #207 from eworm-de/settings



hishamhm authored 22 days ago

latest commit b1aea7f748

| | | |
|-----------------|--|--------------|
| freebsd | Cast FreeBSDProcess_new to Process_New | a month ago |
| linux | fix compiler warnings | 2 months ago |
| m4 | Remove generated files from version history | 4 years ago |
| scripts | Sorry about the mega-patch. | 6 months ago |
| unsupported | Add files to unsupported platform. | 4 months ago |
| .gitignore | add */.dirstamp to .gitignore | 2 months ago |
| AUTHORS | Initial import. | 10 years ago |
| Action.c | Visual tweaks: change color when following, add Broken Gray theme. | 3 months ago |
| Action.h | handle clicks on panel header line | 4 months ago |
| Affinity.c | Sorry about the mega-patch. | 6 months ago |
| Affinity.h | Sorry about the mega-patch. | 6 months ago |
| AffinityPanel.c | Simplify constructors. | 4 months ago |
| AffinityPanel.h | Sorry about the mega-patch. | 6 months ago |

Code

Issues 31

Pull requests 28

Wiki

Pulse

Graphs

Settings

SSH clone URL

git@github.com:h

You can clone with HTTPS, SSH, or Subversion.

Download ZIP

Programação orientada a objetos

Objetos

Classes

Herança

Subtipagem

Métodos virtuais

Estruturas de dados



Peraí, isso é "Java"?

Não, apenas algumas convenções:

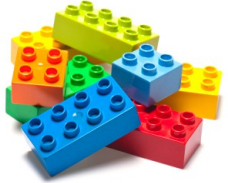
Uma classe por arquivo

Nomenclatura "CamelCase"



```
this  
setDefaultBar  
Vector  
Hashtable  
indexOf  
Object  
AvailableColumnsPanel
```

Objetos



objetos = atributos + métodos

Objetos



objetos = atributos + métodos

em C:

atributos = struct

métodos = funções

Construtor e destrutor

```
MyObject* MyObject_new(int foo) {  
    MyObject* this = malloc(sizeof(MyObject));  
    this->foo = foo;  
    return this;  
}
```

```
MyObject_delete(MyObject* this) {  
    free(this);  
}
```

Métodos

```
int MyObject_sumValue(MyObject* this, int value) {  
    this->foo += value;  
    return this->foo;  
}
```


Objetos e métodos no htop

Em quase todo lugar:

Painéis

Elementos dos painéis

Processos

Lista de processos

Medidores

Cabeçalho

Barras de funções

Visibilidade



public vs. private (vs. ...)

Visibilidade



public vs. private (vs. ...)

em C:

métodos privados = funções static

Métodos públicos e privados

```
void MyObject_publicMethod(MyObject* this) {  
    ...  
}
```

```
static void MyObject_privateMethod(MyObject* this) {  
    ...  
}
```

Se...

Métodos públicos e privados

```
void MyObject_publicMethod(MyObject* this) {  
    ...  
}
```

```
static void MyObject_privateMethod(MyObject* this) {  
    ...  
}
```

Se temos uma classe por arquivo

Subtipagem (!= herança!)



relação "X is a Y"

Subtipagem (!= herança!)



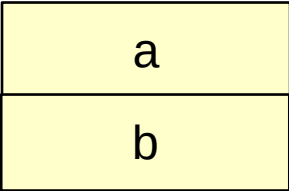
relação "X is a Y"

em C:

"Onde os casts são válidos?"

Subtipagem em C

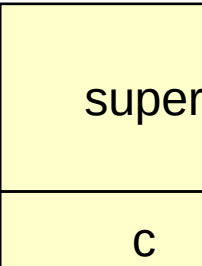
```
typedef struct Dog_ {  
    int a;  
    double b;  
} Dog;
```



A diagram showing the memory layout of a Dog struct. It consists of two vertically stacked yellow rectangular boxes. The top box contains the letter 'a' and the bottom box contains the letter 'b'.

```
Dog* d = ...;  
printf("%d", d->a);
```

```
typedef struct Pug_ {  
    Dog super;  
    char c;  
} Pug;
```

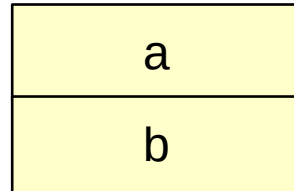


A diagram showing the memory layout of a Pug struct. It consists of two vertically stacked yellow rectangular boxes. The top box contains the word 'super' and the bottom box contains the letter 'c'.

```
Pug* p = ...;  
printf("%c", p->c);  
printf("%d", p->super.a);  
Dog* x = (Dog*) &p;  
printf("%d", x->a);
```


Subtipagem em C

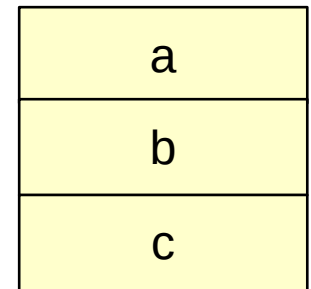
```
typedef struct Dog_ {  
    int a;  
    double b;  
} Dog;
```



```
Dog* d = ...;  
printf("%d", d->a);
```

Pug is a Dog

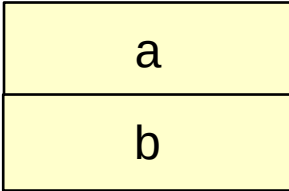
```
typedef struct Pug_ {  
    Dog super;  
    char c;  
} Pug;
```



```
Pug* p = ...;  
printf("%c", p->c);  
printf("%d", p->super.a);  
Dog* x = (Dog*) &p;  
printf("%d", x->a);
```

Subtipagem em C

```
typedef struct Dog_ {  
    int a;  
    double b;  
} Dog;
```

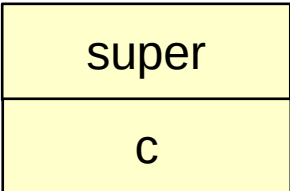


A diagram showing the memory layout of a Dog struct. It consists of two yellow rectangular boxes stacked vertically. The top box contains the letter 'a' and the bottom box contains the letter 'b'.

```
Dog* d = ...;  
printf("%d", d->a);
```

Pug is **not** a Dog!

```
typedef struct Pug_ {  
    Dog* super;  
    char c;  
} Pug;
```



A diagram showing the memory layout of a Pug struct. It consists of two yellow rectangular boxes stacked vertically. The top box contains the word 'super' and the bottom box contains the letter 'c'.

```
Pug* p = ...;  
printf("%c", p->c);  
printf("%d", p->super->a);  
Dog* x = (Dog*) &p;  
printf("%d", x->a);
```

Subtipagem em C

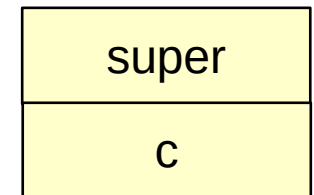
```
typedef struct  
    int a;  
    double b;  
} Dog;
```

```
Dog* d = ...;  
printf("%d", d->
```

Pug is **not** a Dog!?



```
struct Pug_ {  
    super;  
    c;  
};
```



```
...;  
printf("%c", p->c);  
printf("%d", p->super->a);  
Dog* x = (Dog*) &p;  
printf("%d", x->a);
```

Subtipagem no htop

Process is an Object

Meter is an Object

Panel is an Object

 MainPanel is a Panel

 MetersPanel is a Panel

 etc.

um Panel apresenta uma lista de Objects

Herança



Quais classes
reaproveitam código de quais



Quais classes
reaproveitam código de quais

em C:

como organizamos nossos
ponteiros para funções

Métodos virtuais



Em C++ temos...

```
class Pessoa {
    void ola() { printf("Bom dia"); }
    virtual void bye() { printf("Adeus"); }
};

class Barulhento: Pessoa {
    void ola() { printf("BOM DIA!"); }
    virtual void bye() { printf("FALOU!"); }
};

...

Barulhento joaozinho;
Pessoa* p = &joaozinho;
p->ola(); p->bye();
```

Métodos virtuais

Em C...

```
void Pessoa_ola() { printf("Bom dia");  
void Pessoa_bye() { printf("Adeus"); }  
  
void Barulhento_ola() { printf("BOM DIA!"); }  
void Barulhento_bye() { printf("FALOU!"); }  
  
...  
  
Barulhento* joaozinho = Barulhento_new();  
Pessoa* p = joaozinho;  
Pessoa_ola(); p->klass->bye();
```


Tabela de ponteiros de função

Em C, criamos uma struct contendo os ponteiros dos métodos virtuais e os "atributos de classe"

C++ e Java não fazem mágica

Métodos virtuais no htop

Classes que derivam de Object redefinem

delete

display

Classe Panel usa o método display

Na prática, uma minoria das funções!

Estruturas de dados



Linguagens OO tipicamente oferecem coleções padrão

Estruturas de dados



Linguagens OO tipicamente oferecem coleções padrão

em C:

Coleções (vetores, hashtables) podem nos ajudar muito na gerência de memória

C e as segmentation faults

ponteiro não inicializado

ponteiro contém valor velho

ponteiro contém valor inválido

C e as segmentation faults

ponteiro não inicializado

erro na inicialização do objeto

ponteiro contém valor velho

ponteiro contém valor inválido

C e as segmentation faults

ponteiro não inicializado

erro na inicialização do objeto

ponteiro contém valor velho

problema no gerência do tempo de vida

ponteiro contém valor inválido

C e as segmentation faults

ponteiro não inicializado

erro na inicialização do objeto

ponteiro contém valor velho

problema no gerência do tempo de vida

ponteiro contém valor inválido

erro em aritmética de ponteiros / limites

Separation of concerns

Classes implementando estruturas de dados de alto nível:

Vector, Hashtable, Stack...

Isolar código que faz "coisas complicadas" com ponteiros nessas classes

O resto do código fica simples:

```
Vector_insert(v, 3, data);
```

Ownership

Gerência de tempo de vida dos objetos

Cada objeto tem **um** dono

Geralmente é:
ou quem o criou
ou uma **coleção**

Coleções com ownership

```
Vector* Vector_new(ObjectClass* type,  
                  bool owner, int size)
```

Se o Vector é "owner",
ele chama o método delete do Object
ao remover um elemento

(o destrutor certo é chamado
graças à herança!)

Estruturas de dados no htop

Vector

cada painel tem um

o ScreenManager controla um Vector de Panels

Hashtable

processTable

usersTable

Ownership no htop

Código simétrico:

Quase todos os malloc() e free() do programa ficam em construtores e destrutores

Granularidade no controle:

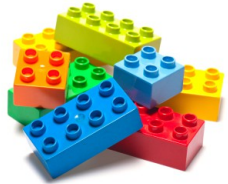
Quase todas as chamadas explícitas a destrutores são para destruir alguma coleção, e não objetos individuais

Um pouquinho de introspecção



RTTI (Run Time Type Information)

Um pouquinho de introspecção



RTTI (Run Time Type Information)

em C:

Como nossas VTables são explícitas, podemos usá-las para identificar as classes em tempo de execução

Útil para debugging (das coleções!)

Comentários finais

Programação C precisa ser disciplinada

Comentários finais

Programação C precisa ser disciplinada

OO é uma disciplina de programação

Comentários finais

Programação C precisa ser disciplinada

OO é uma disciplina de programação

OO em C funciona bem e é eficiente

Comentários finais

Programação C precisa ser disciplinada

OO é uma disciplina de programação

OO em C funciona bem e é eficiente

Algumas poucas macros escondem as partes mais feias e/ou repetitivas

Comentários finais

Programação C precisa ser disciplinada

OO é uma disciplina de programação

OO em C funciona bem e é eficiente

Algumas poucas macros escondem as partes mais feias e/ou repetitivas

Use C se fizer sentido para os seus requisitos

Reuso!

```
Field_delete(saveAsField);
return saved;
}

int main(int argc, char** argv) {

    if (argc > 1) {
        if (String_eq(argv[1], "--version")) {
            printVersionFlag();
        }
    } else {
        fprintf(stderr, "Usage: dit <filename>\n");
        exit(0);
    }

    int quit = 0;

    struct stat st;
    stat(argv[1], &st);
    if (S_ISDIR(st.st_mode)) {
        fprintf(stderr, "dit: %s is a directory.\n", argv[1]);
        exit(0);
    }
}
Lin=80 Col=12 [*] dit.c
```

<http://hisham.hm/dit/>

Obrigado!

Perguntas?

htop e mais em:

<http://hisham.hm/> [@hisham_hm](https://twitter.com/hisham_hm)

Avalie essa apresentação usando o Makadu!

APK do aplicativo Android:

<http://www.makadu.net/makadu.apk>

Estes slides são licenciados pela Creative Commons CC BY 4.0:
<https://creativecommons.org/licenses/by/4.0/>

