

Toscop

Fabio Kleis fabiohrkc@gmail.com

May 1, 2023

Toscop é um dashboard de processos para linux escrito em linguagem C. O projeto utiliza syscalls do linux e bibliotecas padrão da linguagem C como *pthread* para programação em multithreading e *ncurses* para programação de uma *tui*, segue uma organização simples separando um diretório para arquivos .h e outro diretório para arquivos .c, utiliza o sistema make para o gerenciamento de compilação:

```
toscop/
├── include/.....C include directory
│   ├── proc_list.h
│   ├── proc_parser.h
│   ├── term_header.h
│   ├── term_procs.h
│   ├── toscop.h
│   ├── toscop_thread.h
│   ├── toscop_win.h
│   └── w_proc.h
├── LICENSE.....MIT LICENSE
├── Makefile ..... GNU Makefile
├── README.md ..... README PLS
└── src/.....C source directory
    ├── main.c
    ├── proc_list.c
    ├── proc_parser.c
    ├── term_header.c
    ├── term_procs.c
    ├── toscop.c
    ├── toscop_thread.c
    ├── toscop_win.c
    └── w_proc.c
```

Antes de prosseguirmos com as explicações do código alguns esclarecimentos sobre a organização do documento devem estar definidas. O objetivo é mostrar de forma resumida as estruturas principais do projeto e seu funcionamento como um todo. Então serão suprimidas muitas informações que podem estar presentes no código fonte do projeto mas não estarem presente neste documento.

O começo do programa está no arquivo *toscop/src/main.c*, nele é chamada função *cli* para tratar os argumentos via linha de comando e parametrizar as variáveis globais, e também a função *run* que inicializa as estruturas globais, cria as threads e faz o join delas, e por fim termina o processo.

```
#include "toscop.h"

int main(int argc, char** argv) {
    cli(argc, argv);
    run();
    pthread_exit(NULL);
}
```

1 Threads do pthread

O programa utiliza duas threads para separar suas funcionalidades, sendo mostrar as informações na tela e atualizar suas informações, as threads são gerenciadas via variáveis globais no arquivo *toscop/include/toscop.h*:

```
#define MUTEX_FUNC(MUTEX, FUNC, ...) do \
    { pthread_mutex_lock(MUTEX); FUNC(__VA_ARGS__); pthread_mutex_unlock(MUTEX); } \
while(0)

extern pthread_mutex_t toscop_mutex;    // mutex global
extern double max_time;                 // tempo maximo para cada refresh
extern bool fdebug;                     // flag de debug
extern term_header* th;                 // gerenciador das informacoes globais
extern term_procs* tp;                  // gerenciador da lista de procs
extern toscop_wm* wm;                   // gerenciador de window para cada info
extern void run(void);
extern void cli(int argc, char** argv);
```

O programa divide as duas principais funcionalidades nas threads *print_th* e *refresh_th*, a estrutura das threads e suas funções estão declaradas no arquivo *toscop/include/toscop_thread.h*:

```
typedef struct toscop_thread_t {
    pthread_t thread_id;
} toscop_thread_t;

extern void* print_th(void* arg);
extern void* refresh_th(void* arg);
```

São definidas variáveis static no começo do arquivo *toscop/src/toscop_thread.c* para o controle da execução das threads e o tempo de atualização das informações globais.

```
static int k_p = 0;
static double refresh_t = 0;
```

1.1 Thread print_th

A thread `print_th` é responsável por mostrar as informações contidas em cada uma das estruturas declaradas globalmente no arquivo `toscop/include/toscop.h`, a função `print_th` está definida no arquivo `toscop/src/toscop_thread.c`:

```
void* print_th(void* arg) {
    (void) arg;

    clock_t pr_t = clock();
    double p_t = 0;

    while (k_p != 'q') {

        // muda o estado interno do do wm com base no k_p
        MUTEX_FUNC(&toscop_mutex, handle_key, wm, &k_p);

        p_t = (double)(clock() - pr_t) / CLOCKS_PER_SEC;

        if (p_t >= 0.2) { // a cada 0.2 segundos é atualizado o print

            MUTEX_FUNC(&toscop_mutex, show_toscop, wm);

            // mostra informacoes de debug (configurado via flag -v)
            if (fdebug)
                MUTEX_FUNC(&toscop_mutex, show_debug_info, wm, refresh_t);

            pr_t = clock(); // reseta clock
        }
    }
    pthread_exit(NULL);
}
```

A condição de parada é definida quando a tecla ‘q’ é pressionada, podendo assim ser feito o `pthread_join`. A funcionalidade dessa thread é mostrar todas as informações do programa através da função `show_topscop` que é envelopada na macro `MUTEX_FUNC` que força essa chamada de função executar com exclusão mútua através do mutex global `toscop_mutex`.

1.2 Thread refresh_th

A thread `refresh_th` é responsável por atualizar as informações contidas em cada uma das estruturas declaradas globalmente no arquivo `toscop/include/toscop.h`, a função `refresh_th` está definida no arquivo `toscop/src/toscop_thread.c`:

```
void* refresh_th(void* arg) {
    (void) arg;

    struct timespec st = {0}, ct = {0};
    clock_gettime(CLOCK_MONOTONIC, &st); // pega o tempo deis do boot
    cpu_stats last_stat = {0};

    while (k_p != 'q') {

        clock_gettime(CLOCK_MONOTONIC, &ct);
        // calcula o tempo atual - tempo do começo
        refresh_t = (ct.tv_sec - st.tv_sec) + ((ct.tv_nsec - st.tv_nsec) / N_TO_S);

        // caso tenha passado max_time sec da refresh
        if (refresh_t >= max_time) {
            // reseta o clock, start time recebe current time
            st = ct;
            // guarda o ultimo valor de cpu usage sem delta
            last_stat = th->cpu_stat;

            pthread_mutex_lock(&toscop_mutex);

            tp_free(tp); // limpa lista de procs anterior
            tp = create_term_procs(); // cria nova lista

            th_free(th); // limpa informacoes globais anterior
            th = create_term_header(); // cria o novo term header
            calc_cpu_stats(th, last_stat); // calcula delta de % cpu

            pthread_mutex_unlock(&toscop_mutex);

        }
    }
    pthread_exit(NULL);
}
```

A condição de parada é a mesma da thread `print_th`. A thread `refresh_th` atualiza as informações das estruturas globais de forma periódica com exclusão mútua utilizando as funções `pthread_mutex_lock` e `pthread_mutex_unlock`, sendo controlado os ciclos pela variável global `max_time`.

2 Telas do ncurses

O programa define um gerenciador de janelas `toscop_wm` que separa as informações coletadas por telas, em ordem, começando com informações globais, em seguida uma lista com todos os processos lidos e por fim uma tela com informações detalhadas de um processo:

1. `t_win th_win`
2. `t_win tp_win`
3. `t_win proc_win`

Seguindo a mesma enumeração, as estruturas utilizadas para guardar os dados são:

1. `term_header th`
2. `term_procs tp`
3. `w_proc proc`

A estrutura de tela utilizada está declarada no arquivo `toscop/include/toscop_win.h`:

```
// wrapper da window com suas informacoes basicas
typedef struct t_win {
    WINDOW* win;
    int width;
    int height;
    int x;
    int y;
    uint64_t starts_at;
} t_win;
// enum para alternar entre telas
typedef enum WIN_TABS {
    TH_WIN = 0,
    TP_WIN,
    PROC_WIN,
} WIN_TABS;
// estrutura para gerenciamento das telas do ncurses
typedef struct toscop_wm {
    t_win th_win;           // term_header window (info global)
    t_win tp_win;           // term_procs window  (info da lista de procs)
    t_win proc_win;         // window do proc      (info do processo)

    WIN_TABS c_win;         // window atual que esta com foco
} toscop_wm;
```

2.1 Tela th_win

A primeira tela do toscop mostra informações globais do sistema operacional. Todas as informações foram obtidas via *syscalls* do linux ou lendo o diretório */proc* e seus subdiretórios. A estrutura que guarda todas as informações da tela *th_win*, é *term_header* que está declarada em *toscop/include/term_header.h*

```
// cabeçalho do toscop com informações globais
typedef struct term_header {
    struct sysinfo si; // sysinfo contem informações do sistema
    struct tm* ti;     // timeinfo contem informacoes de tempo

    cpu_stats cpu_stat; // valores de tempo de uso do cpu, em secs e em %
    mem_stats mem_stat; // valores de memoria, em uso, livre, virtual etc...

    double lavg[3];     // loadavg do sistema

    uint64_t d_uptime;  // tempo em sec dos dias
    uint64_t h_uptime;  // tempo em sec das horas
    uint64_t m_uptime;  // tempo em sec dos minutos

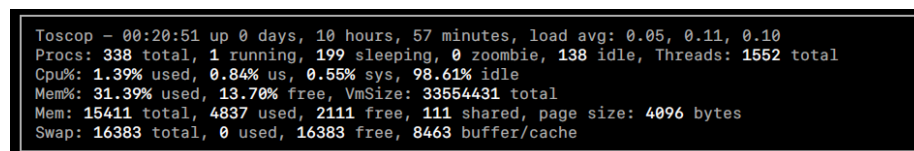
    uint64_t t_procs;   // total de processos listados no /proc/[pid]
    uint64_t t_threads; // total de tasks listados no /proc/[pid]/task/[tid]
} term_header;
```

nota: si é um struct utilizado na *syscall sysinfo*;

nota: ti é o struct retornado da função *localtime* utilizando a *syscall time*;

nota: *cpu_stats* é um struct que guarda informações do */proc/stat*;

nota: *mem_stats* é um struct que guarda informações do */proc/meminfo*;



The screenshot shows the output of the toscop th_win command. It displays system statistics including uptime, load average, process counts, CPU usage, memory usage, and swap usage. The text is as follows:

```
Toscop - 00:20:51 up 0 days, 10 hours, 57 minutes, load avg: 0.05, 0.11, 0.10
Procs: 338 total, 1 running, 199 sleeping, 0 zombie, 138 idle, Threads: 1552 total
Cpu%: 1.39% used, 0.84% us, 0.55% sys, 98.61% idle
Mem%: 31.39% used, 13.70% free, VmSize: 33554431 total
Mem: 15411 total, 4837 used, 2111 free, 111 shared, page size: 4096 bytes
Swap: 16383 total, 0 used, 16383 free, 8463 buffer/cache
```

Figure 1: exemplo tela *th_win*

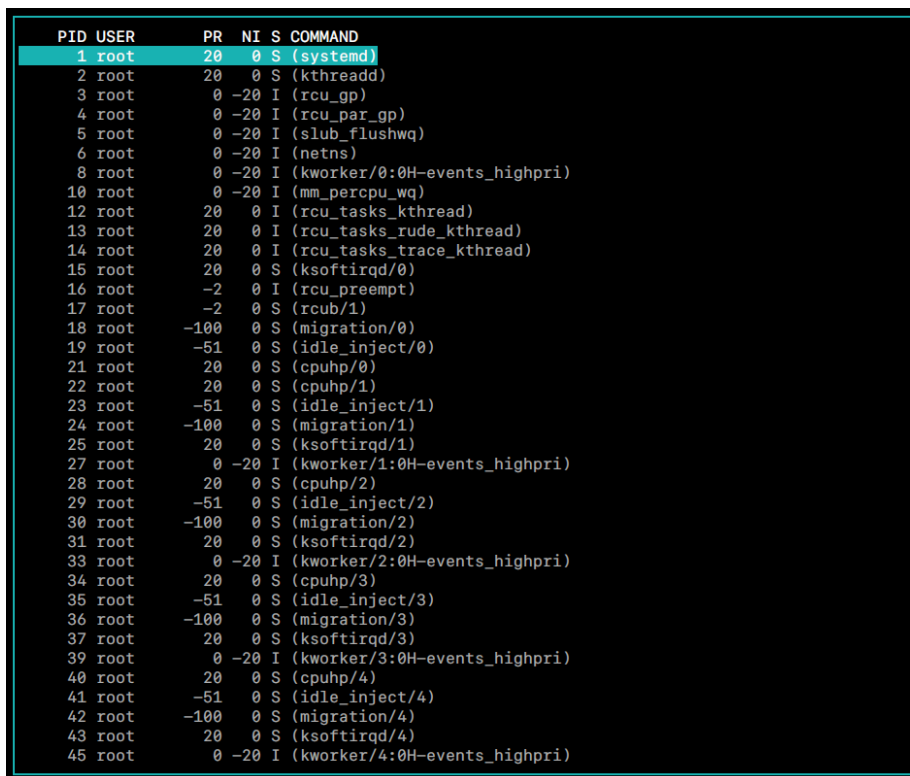
As informações do sistema operacional estão organizadas em linhas, contendo data e hora, número de processos e threads, estados dos processos, porcentagem de uso de CPU e de memória, memória RAM e memória virtual, memória swap e memória compartilhada, tamanho de página e memória em buffer/cache.

2.2 Tela tp_proc

A segunda tela do toscop mostra uma lista de processos. Todas as informações para criar a lista são lidas no */proc/[pid]* e seus subdiretórios, a estrutura que

guarda todos processos é `term_procs` declarada em `toscop/include/term_procs.h` que contém a lista duplamente encadeada declarada em `toscop/include/proc_list.h`:

```
// lista duplamente encadeada para guardar a struct do proc
typedef struct proc_list {
    w_proc *proc;
    struct proc_list *next;
    struct proc_list *prev;
} ProcList;
// estrutura que le o /proc e inicializa a lista de procs
typedef struct term_procs {
    ProcList *proc_list;
    ProcList *proc_list_tail;
} term_procs;
```



PID	USER	PR	NI	S	COMMAND
1	root	20	0	S	(systemd)
2	root	20	0	S	(kthreadd)
3	root	0	-20	I	(rcu_gp)
4	root	0	-20	I	(rcu_par_gp)
5	root	0	-20	I	(slub_flushwq)
6	root	0	-20	I	(netns)
8	root	0	-20	I	(kworker/0:0H-events_highpri)
10	root	0	-20	I	(mm_percpu_wq)
12	root	20	0	I	(rcu_tasks_kthread)
13	root	20	0	I	(rcu_tasks_rude_kthread)
14	root	20	0	I	(rcu_tasks_trace_kthread)
15	root	20	0	S	(ksoftirqd/0)
16	root	-2	0	I	(rcu_preempt)
17	root	-2	0	S	(rcub/1)
18	root	-100	0	S	(migration/0)
19	root	-51	0	S	(idle_inject/0)
21	root	20	0	S	(cpuhp/0)
22	root	20	0	S	(cpuhp/1)
23	root	-51	0	S	(idle_inject/1)
24	root	-100	0	S	(migration/1)
25	root	20	0	S	(ksoftirqd/1)
27	root	0	-20	I	(kworker/1:0H-events_highpri)
28	root	20	0	S	(cpuhp/2)
29	root	-51	0	S	(idle_inject/2)
30	root	-100	0	S	(migration/2)
31	root	20	0	S	(ksoftirqd/2)
33	root	0	-20	I	(kworker/2:0H-events_highpri)
34	root	20	0	S	(cpuhp/3)
35	root	-51	0	S	(idle_inject/3)
36	root	-100	0	S	(migration/3)
37	root	20	0	S	(ksoftirqd/3)
39	root	0	-20	I	(kworker/3:0H-events_highpri)
40	root	20	0	S	(cpuhp/4)
41	root	-51	0	S	(idle_inject/4)
42	root	-100	0	S	(migration/4)
43	root	20	0	S	(ksoftirqd/4)
45	root	0	-20	I	(kworker/4:0H-events_highpri)

Figure 2: exemplo tela `tp-win`

As informações dos processos estão separadas por colunas sendo: identificador do processo; nome do usuário dono do processo; prioridade real no escalonador; prioridade em espaço de usuário; estado do processo; arquivo executável que iniciou o processo.

2.3 Tela proc_win

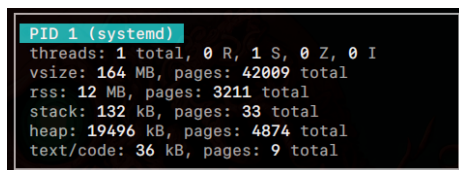
A terceira tela do toscop mostra informações detalhadas de um processo. Todas as informações para criar um processo são lidos no `/proc/[pid]` e seus subdiretórios, a estrutura que guarda um processo é `w_proc` declarada em `toscop/include/w_proc.h`:

```
// conteudo de um processo
typedef struct w_proc {
    token* ptokens;           // campos do /proc/[pid]/stat
    char* path;               // caminho absoluto -> /proc/[pid]
    char* owner_name;         // nome do usuario dono da proc
    int uid;                  // user id
    int gid;                  // grupo dono da proc

    uint64_t r_mem;           // (rss) ram total de memoria do processo
    uint64_t v_mem;           // virtual size total do processo
    uint64_t pr_mem;          // total de paginas de rss do processo
    uint64_t pv_mem;          // total de paginas de vm do processo
    uint64_t stack_pages;     // total de paginas da stack
    uint64_t stack_size;      // total em kB da stack
    uint64_t heap_pages;      // total de paginas da heap
    uint64_t heap_size;       // total em kB da heap
    uint64_t text_pages;      // total de paginas do text
    uint64_t text_size;       // total em kB do text

    uint64_t r_threads;       // running threads de um proc
    uint64_t s_threads;       // sleeping threads de um proc
    uint64_t z_threads;       // zombie threads de um proc
    uint64_t i_threads;       // idle threads de um proc
} w_proc;
```

nota: token `*ptokens` é um struct declarado em `toscop/include/proc-parser.h` que guarda os campos definidos no `/proc/[pid]/stat`;



```
PID 1 (systemd)
threads: 1 total, 0 R, 1 S, 0 Z, 0 I
vsize: 164 MB, pages: 42009 total
rss: 12 MB, pages: 3211 total
stack: 132 kB, pages: 33 total
heap: 19496 kB, pages: 4874 total
text/code: 36 kB, pages: 9 total
```

Figure 3: exemplo tela proc_win

A tela do processo está organizada em linhas, contendo identificador e nome do comando que inicializou o processo, número total de threads e seus estados, memória virtual, memória stack, memória heap e memória de text/code, tanto em kB quanto em número de páginas.