Санкт-Петербургский государственный политехнический университет Институт Информационных Технологий и Управления

Кафедра компьютерных систем и программных технологий

Отчёт по практической работе

по предмету «Системное программное обеспечение»

**Утилита top**

Работу выполнил студент гр. 53501/3 Мартынов С. А. Работу принял преподаватель Душутина Е. В.

Санкт-Петербург 2015

**Содержание**

[Постановка задачи 3](#_TOC_250015)

[Введение 4](#_TOC_250014)

1. [Виртуальная файловая система procfs 5](#_TOC_250013)
2. [Процессы 9](#_TOC_250012)
3. [Измерение уровня заряда батарейки 17](#_TOC_250011)
4. [Мониторинг времени 25](#_TOC_250010)
5. [Центральный процессор 26](#_TOC_250009)
6. [Имя устройства (хоста) 31](#_TOC_250008)
7. [Измерение средней загрузки 32](#_TOC_250007)
8. [Измерение уровня использования памяти 35](#_TOC_250006)
9. [Измерение уровня использования области подкачки 37](#_TOC_250005)
10. [Мониторинг процессов 39](#_TOC_250004)
11. [Измерение времени работы системы 41](#_TOC_250003)
12. [Модификация для работы с процессом 43](#_TOC_250002)

[Заключение 48](#_TOC_250001)

[Список литературы 49](#_TOC_250000)

# Постановка задачи

В рамках данной работы необходимо ознакомиться с работой утилиты top. Рассмотреть и описать механизмы сбора информации, используемые системные вызовы и общий порядок функционирования утилиты.

В работе необходимо указать источники и версии используемых программных продуктов. Показать пример доступа к proc в утилите.

Привести отдельные собственные программные примеры доступа к proc (по чтению и записи посредством утилит и напрямую);

Предложить модификацию утилиты дополнением информации о входящих потоках (нитях) в заданный в качестве параметра процесс, а также по ключу информации о потоках каждого процесса.

Описать возможные способы получения и источники информации о потоках процесса, привести отдельные программные примеры (вне модифицированной утилиты) получения информации о входящих потоках для процесса.

Полные исходные коды сделать доступными по адресу https://github.com/SemenMartynov/ SPbPU\_SystemProgramming.

# Введение

Работа выполняется под управлением Ubuntu 14.04.2 LTS. В качестве изучаемой утилиты используется реализация htop (автор Hisham H. Muhammad). Htop написан на языке Си и использует для отображения библиотеку Ncurses. Htop показывает динамический список системных процессов (рисунок 1), список обычно выравнивается по использованию ЦПУ. В отличие от top, htop показывает все процессы в системе. Также показывает время непрерывной работы, использование процессоров и памяти.

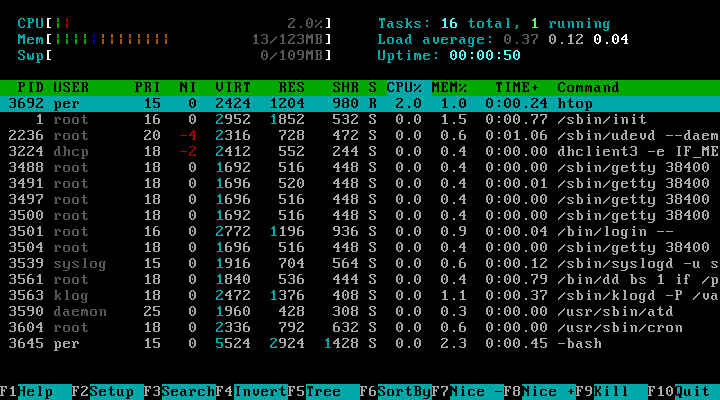


Рис. 1: Системный монитор htop

Рассматривается версия 1.0.3 (от 24 апреля 2014 года). Исходники доступны по лицензии GPL на сайте [http://hisham.hm/htop/.](http://hisham.hm/htop/) Для простоты изучения, в код системы были внесены не значительные изменения.

# 1 Виртуальная файловая система procfs

Файловая система /ргос содержит подробную информацию об активных процессах. Инфор- мация о процессе, сохраненная в файловой системе /ргос, изменяется по мере прохождения данным процессом его жизненного цикла.

Каждый элемент в каталоге /ргос - это десятичное число, соответствующее идентификатору какого-нибудь процесса. Каждый каталог в файловой системе /ргос содержит файлы с более подробной информацией о данном процессе. Владелец каждого файла в каталоге

/ргос и его подкаталогах устанавливается по номеру идентификатора пользователя данного процесса.

Первоначально procfs была разработана для свободного получения информации о состоянии процессов, теперь ее функции расширились, и через эту виртуальную файловую систему процессам можно передавать какие-то параметры (см. рис. 2).

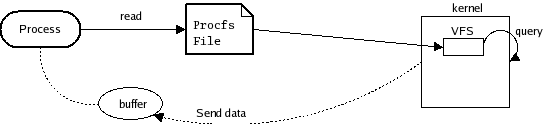


Рис. 2: Виртуальная файловая система procfs

Виртуальной она называется потому, что имеющиеся здесь файлы и каталоги на самом деле не находятся на жестком диске. После загрузки ядра они находятся в оперативной памяти. Для пользователя этот механизм полностью прозрачен. Многие программы, в том числе htop, собирают информацию из файлов в /рrос, форматируют их и выводят результат.

В /рrос можно найти также информацию об установленном оборудовании, разметке жест- кого диска, статистику и многое другое.

При работе с /proc есть важный нюанс: информацию из файла можно прочитать, но открыть его в текстовом редакторе не получится, т.к. его содержимое может измениться в любую секунду. Для записи и считывания данных в такой файл используются утилиты, вроде cat и echo.

Файлы в procfs могут иметь три варианта доступа:

* только для чтения — предназначены для получения информации об определенном

параметре, при попытке в них что-то записать будет выдано предупреждение;

* только для чтения пользователем root — такой же, как и предыдущий, но получить информацию может только администратор;
* только для записи пользователем root — позволяет не только считать данные, но и изменить параметр.

Возможны и некоторые комбинации этих трех вариантов. К тому же в некоторые файлы можно записать только строго определенное значение.[1]

Утилита htop производит считывание различных параметров процессов. Далее будет рассмотрено как это происходит.

Просмотреть информацию о процессоре можно командой cat из файла /proc/cpuinfo. В моём случае это дало следующий результат (для отчёта вывод обрезан, т.к. повторяется по всем ядрам):

sam@spb:~$ cat /proc/cpuinfo processor : 0

vendor\_id : GenuineIntel cpu family : 6

model : 23

model name : Intel(R) Core(TM)2 Quad CPU Q8300 @ 2.50GHz stepping : 10

microcode : 0xa0b cpu MHz : 1998.000

cache size : 2048 KB physical id : 0 siblings : 4

core id : 0 cpu cores : 4 apicid : 0

initial apicid : 0 fpu : yes fpu\_exception : yes cpuid level : 13

wp : yes

flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat

pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx lm constant\_tsc arch\_perfmon pebs bts rep\_good nopl aperfmperf pni dtes64 monitor ds\_cpl vmx est

tm2 ssse3 cx16 xtpr pdcm sse4\_1 xsave lahf\_lm dtherm tpr\_shadow vnmi flexpriority bogomips : 4991.93

clflush size : 64 cache\_alignment : 64

address sizes : 36 bits physical, 48 bits virtual power management:

А вот вывод информации о памяти

sam@spb:~$ cat /proc/meminfo MemTotal: 8176408 kB

MemFree: 391976 kB

Buffers: 39692 kB

Cached: 964508 kB

SwapCached: 1784 kB

Active: 1878368 kB

Inactive: 1060044 kB

Active(anon): 1528192 kB

Inactive(anon): 729880 kB

Active(file): 350176 kB

Inactive(file): 330164 kB

Unevictable: 32 kB

Mlocked: 32 kB

SwapTotal: 8387580 kB

SwapFree: 8359320 kB

Dirty: 0 kB

Writeback: 0 kB

AnonPages: 1932372 kB

Mapped: 4824228 kB

Shmem: 323860 kB

Slab: 176288 kB

SReclaimable: 100712 kB

SUnreclaim: 75576 kB

KernelStack: 4568 kB

PageTables: 39424 kB

NFS\_Unstable: 0 kB

Bounce: 0 kB

WritebackTmp: 0 kB

CommitLimit: 12475784 kB

Committed\_AS: 9617760 kB

VmallocTotal: 34359738367 kB

VmallocUsed: 327844 kB

VmallocChunk: 34359407416 kB

HardwareCorrupted: 0 kB

AnonHugePages: 475136 kB

HugePages\_Total: 0

HugePages\_Free: 0

HugePages\_Rsvd: 0

HugePages\_Surp: 0

Hugepagesize: 2048 kB

DirectMap4k: 1467968 kB

DirectMap2M: 6920192 kB

Следующий пример показывает как можно разрешить машине быть сетевым шлюзом для IPv6 соединений (параметр будет сброшен в 0 после перезагрузки):

sam@spb:~$ cat /proc/sys/net/ipv6/conf/all/forwarding 0

sam@spb:~$ echo "1" | sudo tee -a /proc/sys/net/ipv6/conf/all/forwarding [sudo] password for sam:

1

sam@spb:~$ cat /proc/sys/net/ipv6/conf/all/forwarding 1

sam@spb:~$

# 2 Процессы

Листинг 1 содержит отрывок файла Process.h, описывающий структуру Process. На самом деле этот отрезок короче, если учесть работу препроцессора, который содержит многие строки (отладочную информацию, данные OpenVZ и виртуального сервера).

Листинг 1: Клиент именованного каналов (src/top/Process.h)

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

typedef struct Process\_ {

Object super ;

struct Process List\_ \* pl;

bool updated ;

pid\_ t pid ; char \* comm ; int indent ; char

bool

bool bool

state ;

tag ; show Children ; show ;

pid\_ t ppid ;

unsigned unsigned unsigned

int

int int

pgrp ;

session ; tty\_ nr ;

pid\_ t tgid ;

int tpgid ; unsigned long

# ifdef DEBUG

int

flags ;

unsigned

unsigned unsigned unsigned

# endif unsigned unsigned unsigned

unsigned

long

long long long

int minflt ;

int cminflt ; int majflt ; int cmajflt ;

long

long long long

long

long long long

int

int int int

utime ;

stime ; cutime ; cstime ;

long int priority ;

long int nice ; long int nlwp ;

IOPriority io Priority ; char starttime\_ show [8]; time\_ t starttime\_ ctime ;

# ifdef DEBUG

1. long int itrealvalue ;
2. unsigned long int vsize ;
3. long int rss ;
4. unsigned long int rlim ;
5. unsigned long int startcode ;
6. unsigned long int endcode ;
7. unsigned long int startstack ;
8. unsigned long int kstkesp ;
9. unsigned long int kstkeip ;
10. unsigned long int signal ;
11. unsigned long int blocked ;
12. unsigned long int sigignore ;
13. unsigned long int sigcatch ;
14. unsigned long int wchan ;
15. unsigned long int nswap ;
16. unsigned long int cnswap ;
17. # endif
18. int exit\_ signal ;
19. int processor ;
20. int m\_ size ;
21. int m\_ resident ;
22. int m\_ share ;
23. int m\_trs ;
24. int m\_drs ;
25. int m\_lrs ;
26. int m\_dt ;
27. uid\_ t st\_ uid ;
28. float percent\_ cpu ;
29. float percent\_ mem ;
30. char \* user ;
31. # ifdef HAVE\_ OPENVZ
32. unsigned int ctid ;
33. unsigned int vpid ;
34. # endif
35. # ifdef HAVE\_ VSERVER
36. unsigned int vxid ;
37. # endif
38. # ifdef HAVE\_ TASKSTATS
39. unsigned long long io\_ rchar ;
40. unsigned long long io\_ wchar ;
41. unsigned long long io\_ syscr ;
42. unsigned long long io\_ syscw ;
43. unsigned long long io\_ read\_ bytes ;
44. unsigned long long io\_ write\_ bytes ;
45. unsigned long long io\_ cancelled\_ write\_ bytes ;
46. double io\_ rate\_ read\_ bps ;
47. unsigned long long io\_ rate\_ read\_ time ;
48. double io\_ rate\_ write\_ bps ;
49. unsigned long long io\_ rate\_ write\_ time ;
50. # endif
51. # ifdef HAVE\_ CGROUP
52. char \* cgroup ;
53. # endif
54. } Process ;

Назначение большинства полей понятно исходя из их названий, и они здесь представле- ны для дальнейшего перехода к системам, которые заполняют значение этих полей (к примеру, процент занятой памяти или процессорного времени). Процессы объединяются в списки (строка 4), а объект Object (строка 2) отвечает за отображение. Полный граф взаимодействия структура данных представлен на рисунке 3.

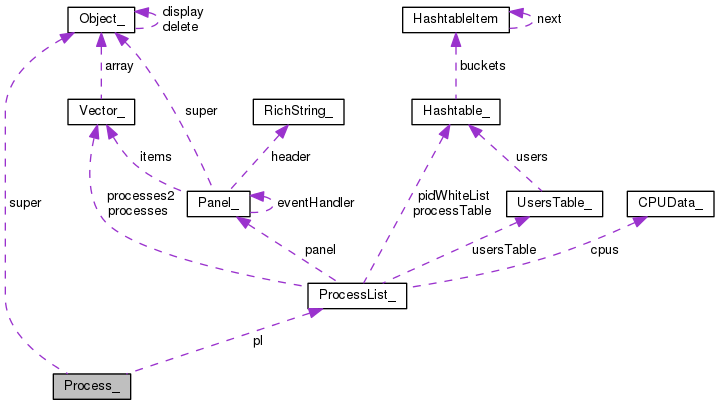


Рис. 3: Граф взаимодействия для структуры Process

На графе видно, что некоторые структуры, такие как CPUData\_ не связаны с конкретным процессом, и это логично, т.к. эта общесистемная информация, но к ней есть доступ через сам список. Структуры, такие как RitcString\_, Vector\_ и Hashtable\_ используются для служебных целей (сортировка списков, парсинг текстовой информации), т.к. программа написана на C (т.е. это реализация структур из C++, реализованная на языке С).

Все процессы помещаются в список, который, помимо самих процессоров, хранит ещё общесистемные показатели, такие как распределение памяти и ресурсов центрального процессора (листинг 2).

Получить показатели используемой памяти можно из файла /proc/meminfo (стр. 5 - макрос подставит правильный путь на место PROCMEMINFOFILE). В этой функции определяется следующие показатели памяти:

* + общий объём памяти (стр 17);
  + объём свободной памяти (стр. 19);
  + объём памяти, разделяемой между процессами (стр. 21);
  + размер буферов (стр. 25);
  + размер кешей (стр.29);
  + общий объём, используемый для хранения страниц памяти на диске (стр. 33);
  + свободное место в свапе (стр. 35);
  + объём используемой памяти, как разность между общим объёмом и свободной памя- тью (стр. 41);
  + объём используемого места в свапе, как разность между общим объёмом и свободным пространством (стр. 42);

Помимо информации о памяти, тут же происходит считывание информации о прерываниях и операциях ввода-вывода из файла /proc/stat (стр. 45 - замена имени файла будет произ- ведена препроцессором). из этого файла для каждого процессора (стр. 49) считываются следующие показатели:

* + время, проведённое процессором в пространстве пользователя (стр. 83);
  + время, на выделение которого повлиял приоритет процесса (стр. 84);
  + время, проведённое процессором в пространстве ядра (стр. 85);
  + время, проведённое процессором в пространстве ядра, включая обработку прерываний (стр. 86);
  + время, проведённое процессором в режиме бездействия (стр. 87);
  + время, проведённое процессором в режиме бездействия, включая время ожидания выполнения операций ввода-вывода (стр. 88);
  + время ожидания выполнения операций ввода-вывода (стр. 89);
    - объём обработки прерываний (стр. 90);
    - объём обработки программных прерываний (стр. 91);
    - украденное время - характерно для гипервизоров (стр. 92);
    - время в режиме гостя - характерно для гипервизоров (стр. 93);
    - общее время работы процессора (стр. 94);
    - объём используемого места в свапе, как разность между общим объёмом и свободным пространством (стр. 42);

Листинг 2: Считывание различных общесистемных показателей(src/top/ProcessList.c)

1. void Process List\_ scan ( Process List \* this ) {
2. unsigned long long int usertime , nicetime , systemtime , systemalltime , idlealltime , idletime , totaltime , virtalltime ;
3. unsigned long long int swap Free = 0;

4

1. FILE \* file = fopen ( PROCMEMINFOFILE , " r");
2. if ( file == NULL ) {
3. CRT\_ fatal Error (" Cannot open " PROCMEMINFOFILE );

8 }

9 int cpus = this -> cpu Count ;

10 {

1. char buffer [ 128 ];
2. while ( fgets ( buffer , 128 , file )) {

13

1. switch ( buffer [0]) {
2. case ’M’:
3. if ( String\_ starts With ( buffer , " Mem Total :"))
4. sscanf ( buffer , " Mem Total : % llu kB", & this -> total Mem );
5. else if ( String\_ starts With ( buffer , " Mem Free :"))
6. sscanf ( buffer , " Mem Free : % llu kB", & this -> free Mem );
7. else if ( String\_ starts With ( buffer , " Mem Shared :"))
8. sscanf ( buffer , " Mem Shared : % llu kB", & this -> shared Mem );
9. break ;
10. case ’B’:
11. if ( String\_ starts With ( buffer , " Buffers :"))
12. sscanf ( buffer , " Buffers : % llu kB", & this -> buffers Mem );
13. break ;
14. case ’C’:
15. if ( String\_ starts With ( buffer , " Cached :"))
16. sscanf ( buffer , " Cached : % llu kB", & this -> cached Mem );
17. break ;
18. case ’S’:
19. if ( String\_ starts With ( buffer , " Swap Total :"))
20. sscanf ( buffer , " Swap Total : % llu kB", & this -> total Swap );
21. if ( String\_ starts With ( buffer , " Swap Free :"))
22. sscanf ( buffer , " Swap Free : % llu kB", & swap Free );
23. break ;

37 }

38 }

39 }

40

1. this -> used Mem = this -> total Mem - this -> free Mem ;
2. this -> used Swap = this -> total Swap - swap Free ;
3. fclose ( file );

44

1. file = fopen ( PROCSTATFILE , " r");
2. if ( file == NULL ) {
3. CRT\_ fatal Error (" Cannot open " PROCSTATFILE );

48 }

1. for ( int i = 0; i <= cpus ; i ++) {
2. char buffer [ 256 ];
3. int cpuid ;
4. unsigned long long int ioWait , irq , softIrq , steal , guest ;
5. io Wait = irq = soft Irq = steal = guest = 0;
6. *// Dependending on your kernel version ,*
7. *// 5 , 7 or 8 of these fields will be set.*
8. *// The rest will remain at zero.*
9. fgets ( buffer , 255 , file );

58 if ( i == 0)

1. sscanf ( buffer , " cpu % llu % llu % llu % llu % llu % llu % llu % llu % llu ",

& usertime , & nicetime , & systemtime , & idletime , & ioWait , & irq , & softIrq , & steal , & guest );

1. else {
2. sscanf ( buffer , " cpu % d % llu % llu % llu % llu % llu % llu % llu % llu % llu "

, & cpuid , & usertime , & nicetime , & systemtime , & idletime , & ioWait ,

& irq , & softIrq , & steal , & guest );

1. *// assert( cpuid == i - 1);*

63 }

1. *// Fields existing on kernels >= 2.6*
2. *// ( and RHEL ’ s patched kernel 2 . 4 ...)*
3. idlealltime = idletime + io Wait ;
4. systemalltime = systemtime + irq + soft Irq ;
5. virtalltime = steal + guest ;
6. totaltime = usertime + nicetime + systemalltime + idlealltime + virtalltime ;
7. CPUData \* cpu Data = &( this -> cpus [ i]);
8. *// assert ( usertime >= cpuData -> user Time );*
9. *// assert ( nicetime >= cpuData -> nice Time );*
10. *// assert ( systemtime >= cpuData -> system Time );*
11. *// assert ( idletime >= cpuData -> idle Time );*
12. *// assert ( totaltime >= cpuData -> total Time );*
13. *// assert ( systemalltime >= cpuData -> system All Time );*
14. *// assert ( idlealltime >= cpuData -> idle All Time );*
15. *// assert ( io Wait >= cpuData -> io Wait Time );*
16. *// assert ( irq >= cpuData -> irq Time );*
17. *// assert ( soft Irq >= cpuData -> soft Irq Time );*
18. *// assert ( steal >= cpuData -> steal Time );*
19. *// assert ( guest >= cpuData -> guest Time );*
20. cpuData -> user Period = usertime - cpuData -> user Time ;
21. cpuData -> nice Period = nicetime - cpuData -> nice Time ;
22. cpuData -> system Period = systemtime - cpuData -> system Time ;
23. cpuData -> system All Period = systemalltime - cpuData -> system All Time ;
24. cpuData -> idle All Period = idlealltime - cpuData -> idle All Time ;
25. cpuData -> idle Period = idletime - cpuData -> idle Time ;
26. cpuData -> io Wait Period = io Wait - cpuData -> io Wait Time ;
27. cpuData -> irq Period = irq - cpuData -> irq Time ;
28. cpuData -> soft Irq Period = soft Irq - cpuData -> soft Irq Time ;
29. cpuData -> steal Period = steal - cpuData -> steal Time ;
30. cpuData -> guest Period = guest - cpuData -> guest Time ;
31. cpuData -> total Period = totaltime - cpuData -> total Time ;
32. cpuData -> user Time = usertime ;
33. cpuData -> nice Time = nicetime ;
34. cpuData -> system Time = systemtime ;
35. cpuData -> system All Time = systemalltime ;
36. cpuData -> idle All Time = idlealltime ;
37. cpuData -> idle Time = idletime ;
38. cpuData -> io Wait Time = io Wait ;
39. cpuData -> irq Time = irq ;
40. cpuData -> soft Irq Time = soft Irq ;
41. cpuData -> steal Time = steal ;
42. cpuData -> guest Time = guest ;
43. cpuData -> total Time = totaltime ;

107 }

108 double period = ( double ) this -> cpus [0]. total Period / cpus ; fclose ( file );

109

1. *// mark all process as " dirty"*
2. for ( int i = 0; i < Vector\_ size ( this -> processes ); i ++) {
3. Process \* p = ( Process \*) Vector\_ get ( this -> processes , i);
4. p-> updated = false ;

114 }

115

116 this -> total Tasks = 0;

|  |  |  |
| --- | --- | --- |
| 117 |  | this -> userland Threads = 0; |
| 118 |  | this -> kernel Threads = 0; |
| 119 |  | this -> running Tasks = 0; |
| 120 |  |  |
| 121 |  | Process List\_ process Entries ( this , PROCDIR , NULL , period ); |
| 122 |  |  |
| 123 |  | this -> showing Thread Names = this -> show Thread Names ; |
| 124 |  |  |
| 125 |  | for ( int i = Vector\_ size ( this -> processes ) - 1; i >= 0; i - -) { |
| 126 |  | Process \* p = ( Process \*) Vector\_ get ( this -> processes , i); |
| 127 |  | if ( p-> updated == false ) |
| 128 |  | Process List\_ remove ( this , p); |
| 129 |  | else |
| 130 |  | p-> updated = false ; |
| 131 |  | } |
| 132 |  |  |
| 133 | } |  |

# 3 Измерение уровня заряда батарейки

Листинг 3 содержит отрывок файла BatteryMeter.c, который содержит функционал для отображения информации об источнике питания.

Как видно в строке 18, информация о батарее находиться по пути /proc/acpi/battery. В этой директории должны находиться файлы, название которых начинается с букв BAT, и это сравнение производится в строке 38. Имена файлов складываются в односвязный список myList (стр. 44), для последующего перебора (стр. 48). В строке 53 формируется полный путь до файла, отображающего состояние батарейки и содержимое этого файла вычитывается в массив line (стр. 61). В этой строке производится смещение на количество символов, переданное в качестве параметра функции, а полученный результат провидится к числу и аккумулируется. Так собирается информация об общем объёме батареи и её состоянии (остатке заряда). Сбор осуществляет функция getProcBatData в стр. 186. Эта же функция определяет процент использования аккумулятора, путём деления остатка заряда на общий объём батареи (стр. 195).

Если источником питания является подключенный сетевой адаптер, то информация об этом может быть получена из /proc/acpi/ac\_adapter (стр. 84) или из /sys/class/power\_supply/ (стр. 132). В обоих случаях производится попытка чтения файла для определения актив- ности адаптера. Этот вариант применим для ноутбуков, в моём случае обе директории оказались пусты.

В этом же файле представлен функционал для отслеживания подключения адаптера (в этот момент источник питания изменяется с батарейки на питание от электросети) или завершение зарядки аккумулятора. События мониторятся по пути /sys/class/power\_supply/ (стр. 220). Код в строке 228 и строке 240 во многом дублируются, т.к. в некоторых системах событие окончание зарядки содержит слово CHARGE, а в некоторых ENERGY. Обработка события происходит в функции из строки 1.

Источник питания может быть электросетью, а может быть аккумуляторной батареей, выво- димый текст определяется в строке 275. Полный граф включения для файла BatteryMeter.c представлен на рисунке 4.

Листинг 3: Battery Meter - измерение уровня заряда(src/top/BatteryMeter.c)

1. static unsigned long int parse Uevent ( FILE \* file , const char \* key ) {
2. char line [ 100 ];
3. unsigned long int d Value = 0;

4

1. while ( fgets ( line , sizeof line , file )) {
2. if ( strncmp ( line , key , strlen ( key )) == 0) {
3. char \* value ;
4. strtok ( line , "=");
5. value = strtok ( NULL , "=");
6. d Value = atoi ( value );
7. break ;

12 }

13 }

14 return d Value ;

15 }

16

1. static unsigned long int parse Bat Info ( const char \* fileName , const unsigned short int lineNum , const unsigned short int word Num ) {
2. const char battery Path [] = PROCDIR "/ acpi / battery /";
3. DIR \* battery Dir = opendir ( battery Path );
4. if (! battery Dir )
5. return 0;

22

1. typedef struct list Lbl {
2. char \* content ;
3. struct list Lbl \* next ;
4. } list ;

27

1. list \* my List = NULL ;
2. list \* new Entry ;

30

31 */\**

1. *Some of this is based off of code found in kismet ( they claim it came from gkrellm).*
2. *Written for multi battery use ...*

34 *\*/*

1. for ( const struct dirent \* dir Entries = readdir (( DIR \*) battery Dir ); dir Entries ; dir Entries = readdir (( DIR \*) battery Dir )) {
2. char \* entry Name = ( char \*) dirEntries -> d\_ name ;

37

1. if ( strncmp ( entryName , " BAT ", 3))
2. continue ;

40

1. new Entry = calloc (1 , sizeof ( list ));
2. newEntry -> next = my List ;
3. newEntry -> content = entry Name ;
4. my List = new Entry ;

45 }

46

1. unsigned long int total = 0;
2. for ( new Entry = my List ; new Entry ; new Entry = newEntry -> next ) {
3. const char info Path [30];
4. const FILE \* file ;
5. char line [50];

52

53 snprintf (( char \*) infoPath , sizeof infoPath , "% s% s/% s", battery Path , newEntry -> content , file Name );

54

1. if (( file = fopen ( infoPath , " r")) == NULL ) {
2. closedir ( battery Dir );
3. return 0;

58 }

59

1. for ( unsigned short int i = 0; i < line Num ; i ++) {
2. fgets ( line , sizeof line , ( FILE \*) file );

62 }

63

64 fclose (( FILE \*) file );

65

1. const char \* found Num Tmp = String\_ get Token ( line , word Num );
2. const unsigned long int found Num = atoi ( found Num Tmp );
3. free (( char \*) found Num Tmp );

69

70 total += found Num ;

71 }

72

1. free ( my List );
2. free ( new Entry );
3. closedir ( battery Dir );
4. return total ;

77 }

78

1. static ACPresence chk Is Online () {
2. FILE \* file = NULL ;
3. ACPresence isOn = AC\_ ERROR ;

82

1. if ( access ( PROCDIR "/ acpi / ac\_ adapter ", F\_OK ) == 0) {
2. const char \* power\_ supply Path = PROCDIR "/ acpi / ac\_ adapter ";
3. DIR \* power\_ supply Dir = opendir ( power\_ supply Path );
4. if (! power\_ supply Dir )
5. return AC\_ ERROR ;

88

1. for ( const struct dirent \* dir Entries = readdir (( DIR \*) power\_ supply Dir

); dir Entries ; dir Entries = readdir (( DIR \*) power\_ supply Dir )) {

1. char \* entry Name = ( char \*) dirEntries -> d\_ name ;

91

1. if ( entry Name [0] != ’A’)
2. continue ;

94

1. char state Path [50];
2. snprintf (( char \*) statePath , sizeof statePath , "% s/% s/ state ", power\_ supply Path , entry Name );
3. file = fopen ( statePath , " r");

98

99 if (! file ) {

1. isOn = AC\_ ERROR ;
2. continue ;

102 }

103

1. char line [ 100 ];
2. fgets ( line , sizeof line , file );
3. line [ sizeof ( line ) - 1] = ’\0 ’;

107

1. if ( file ) {
2. fclose ( file );
3. file = NULL ;

111 }

112

113 const char \* is Online = String\_ get Token ( line , 2);

114

1. if ( strcmp ( isOnline , " on - line ") == 0) {
2. free (( char \*) is Online );
3. isOn = AC\_ PRESENT ;
4. *// If any AC adapter is being used then stop*
5. break ;

120

121 } else {

|  |  |  |
| --- | --- | --- |
| 122 |  | isOn = AC\_ ABSENT ; |
| 123 |  | } |
| 124 |  | free (( char \*) is Online ); |
| 125 | } |  |
| 126 |  |  |
| 127 | if | ( power\_ supply Dir ) |
| 128 |  | closedir ( power\_ supply Dir ); |

|  |  |  |
| --- | --- | --- |
| 129 |  | |
| 130 | } | else { |
| 131 |  |  |
| 132 |  | const char \* power\_ supply Path = "/ sys / class / power\_ supply "; |
| 133 |  |  |
| 134  135 |  | if ( access ("/ sys / class / power\_ supply ", F\_OK ) == 0) {  const struct dirent \* dir Entries ; |
| 136 |  | DIR \* power\_ supply Dir = opendir ( power\_ supply Path ); |
| 137 |  | char \* entry Name ; |
| 138 |  |  |
| 139  140 |  | if (! power\_ supply Dir ) {  return AC\_ ERROR ; |
| 141 |  | } |
| 142 |  |  |
| 143 |  | for ( dir Entries = readdir (( DIR \*) power\_ supply Dir ); dir Entries ; |
| 144 |  | dir Entries = readdir (( DIR \*) power\_ supply Dir )) {  entry Name = ( char \*) dirEntries -> d\_ name ; |
| 145 |  |  |
| 146 |  | if ( strncmp ( entryName , " A", 1)) { |
| 147 |  | continue ; |
| 148 |  | } |
| 149 |  |  |
| 150 |  | char online Path [50]; |
| 151 |  | snprintf (( char \*) online Path , sizeof online Path , "% s/% s/ online ", |

power\_ supply Path , entry Name );

152 file = fopen ( onlinePath , " r");

153

1. if (! file ) {
2. isOn = AC\_ ERROR ;
3. continue ;

157 }

158

159 isOn = ( fgetc ( file ) - ’0 ’);

160

1. if ( file ) {
2. fclose ( file );
3. file = NULL ;

164 }

165

1. if ( isOn == AC\_ PRESENT ) {
2. *// If any AC adapter is being used then stop*
3. break ;
4. } else {
5. continue ;

171 }

172 }

173

1. if ( power\_ supply Dir )
2. closedir ( power\_ supply Dir );

176 }

177 }

178

1. *// Just in case :-)*
2. if ( file )
3. fclose ( file );

182

183 return isOn ;

184 }

185

1. static double get Proc Bat Data () {
2. const unsigned long int total Full = parse Bat Info (" info ", 3 , 4);
3. if ( total Full == 0)
4. return 0;

190

1. const unsigned long int total Remain = parse Bat Info (" state ", 5 , 3);
2. if ( total Remain == 0)
3. return 0;

194

195 double percent = total Full > 0 ? (( double ) total Remain \* 100) / ( double ) total Full : 0;

|  |  |  |
| --- | --- | --- |
| 196 |  | return percent ; |
| 197 | } |  |
| 198 |  |  |
| 199 | sta | tic double get Sys Bat Data () { |
| 200 |  | const struct dirent \* dir Entries ; |
| 201 |  | const char \* power\_ supply Path = "/ sys / class / power\_ supply /"; |
| 202 |  | DIR \* power\_ supply Dir = opendir ( power\_ supply Path ); |
| 203 |  | if (! power\_ supply Dir ) |
| 204 |  | return 0; |
| 205 |  |  |
| 206  207 |  | char \* entry Name ; |
| 208 |  | unsigned long int total Full = 0; |
| 209 |  | unsigned long int total Remain = 0; |
| 210 |  |  |
| 211 |  | for ( dir Entries = readdir (( DIR \*) power\_ supply Dir ); dir Entries ; |
| 212 |  | dir Entries = readdir (( DIR \*) power\_ supply Dir )) {  entry Name = ( char \*) dirEntries -> d\_ name ; |
| 213 |  |  |
| 214 |  | if ( strncmp ( entryName , " BAT ", 3)) { |

|  |  |  |
| --- | --- | --- |
| 215  216 | continue ;  } |  |
| 217 |  |
| 218 | const char uevent Path [50]; |
| 219 |  |
| 220 | snprintf (( char \*) uevent Path , sizeof uevent Path , "% s% s/ uevent ", |
| 221 | power\_ supply Path , entry Name ); |
| 222 | FILE \* file ; |
| 223  224 | if (( file = fopen ( ueventPath , " r")) == NULL ) {  closedir ( power\_ supply Dir ); |
| 225 | return 0; |
| 226 | } |
| 227 |  |
| 228 | if (( total Full += parse Uevent ( file , " POWER\_ SUPPLY\_ ENERGY\_ FULL ="))) | { |
| 229 | total Remain += parse Uevent ( file , " POWER\_ SUPPLY\_ ENERGY\_ NOW ="); |  |
| 230 | } else { |  |
| 231 | *// reset file pointer* |  |
| 232  233 | if ( fseek ( file , 0 , SEEK\_ SET ) < 0) {  closedir ( power\_ supply Dir ); |  |
| 234 | fclose ( file ); |  |
| 235 | return 0; |  |
| 236 | } |  |
| 237 | } |  |
| 238 |  |  |
| 239 | *// Some systems have it as CHARGE instead of ENERGY.* |  |
| 240 | if (( total Full += parse Uevent ( file , " POWER\_ SUPPLY\_ CHARGE\_ FULL ="))) | { |
| 241 | total Remain += parse Uevent ( file , " POWER\_ SUPPLY\_ CHARGE\_ NOW ="); |  |
| 242  243 | } else {  *// reset file pointer* |  |
| 244  245 | if ( fseek ( file , 0 , SEEK\_ SET ) < 0) {  closedir ( power\_ supply Dir ); |  |
| 246 | fclose ( file ); |  |
| 247 | return 0; |  |
| 248 | } |  |
| 249 | } |  |
| 250 |  |  |

251 fclose ( file );

252 }

253

254 const double percent = total Full > 0 ? (( double ) total Remain \* 100) / ( double ) total Full : 0;

|  |  |  |
| --- | --- | --- |
| 255 |  | closedir ( power\_ supply Dir ); |
| 256 |  | return percent ; |
| 257 | } |  |

258

1. static void Battery Meter\_ set Values ( Meter \* this , char \* buffer , int len ) {
2. double percent = get Proc Bat Data ();

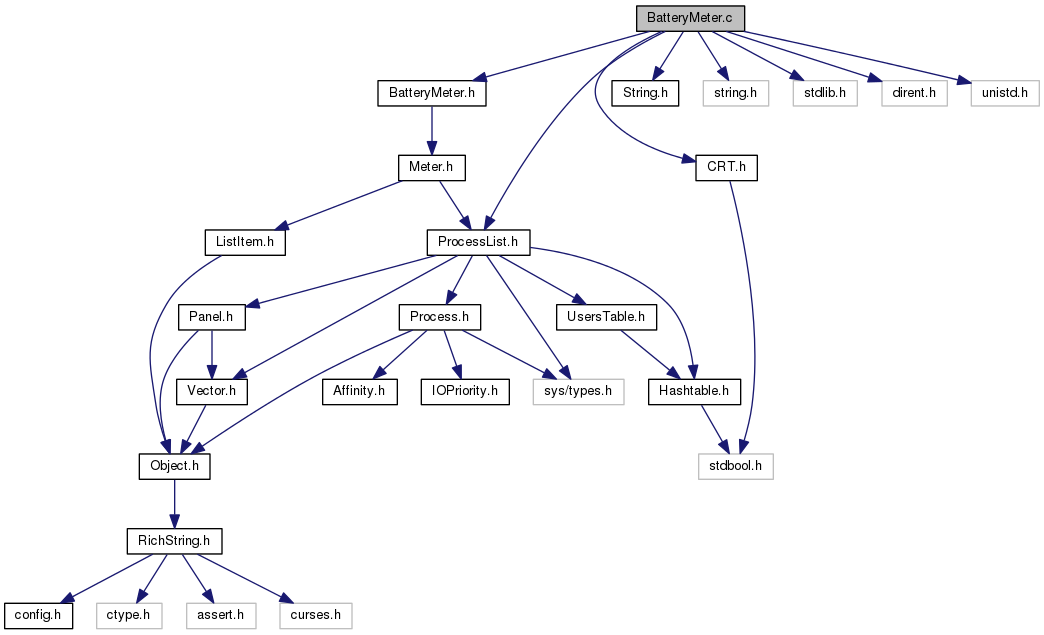


Рис. 4: Граф включения для файла BatteryMeter.c

# 4 Мониторинг времени

Монитор времени отвечает за отображение текущего времени пользователя. Отрывок файла ClockMeter.c представлен в листинге 4 а граф включения на рисунке 5.

Интерес представляет только одна функция. Она берёт локальное время (стр. 3) и пре- образует его к привычному виду (часы:минуты:секунды), сохраняя это преобразование в буфере, который является аргументом функции.

Листинг 4: Clock Meter - мониторинг времени(src/top/ClockMeter.c)

1. static void Clock Meter\_ set Values ( Meter \* this , char \* buffer , int size ) {
2. time\_ t t = time ( NULL );
3. struct tm \* lt = localtime (& t);
4. this -> values [0] = lt -> tm\_ hour \* 60 + lt -> tm\_ min ;
5. strftime ( buffer , size , "% H:% M:% S", lt);

6 }

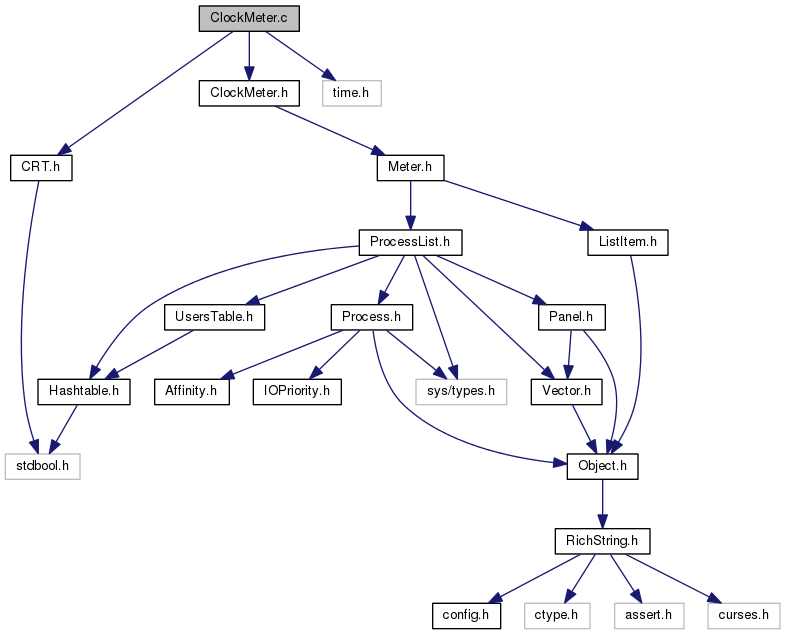


Рис. 5: Граф включения для файла ClockMeter.c

# 5 Центральный процессор

С оценкой потребления процессора всегда встает сложность разделения физических и логических ядер. В одном реальном процессоре может быть несколько ядер, каждое из которых может иметь дополнительный набор регистров, обеспечивающих hyper-threading.

Ранее мы рассматривали как происходит чтение информации об использовании процессора приложениями. Функция CPUMeter\_setValues из листинга 5 занимается подготовкой следующих полей[2]:

* nicePeriod (стр. 11) - означает процент CPU, используемого пользовательскими про- цессами, на которые повлияло использование команд nice или renice, т.е. по существу их приоритет был изменен по сравнению с приоритетом по умолчанию, назначаемому планировщиком, на более высокий или низкий. При назначении какому-либо процессу команды nice, положительное число означает более низкий приоритет (1 = 1 шаг ниже нормального), а отрицательное число означает более высокий приоритет. 0 – значение по умолчанию, что означает, что решение о приоритете принимает планировщик. Можно установить, какой планировщик используется системой.
* userPeriod (стр. 12) - оказывает использование отдельного процессора (пользователь- скими процессами, такими, как apache, mysql и т.д.) до максимального значения, составляющего 100%. Таким образом, если в четырехъядерном процессоре 1 про- цесс использует 100% CPU, это даст значение %us, равное 25%. Значение 12,5% для 8-ядерного процессора означает, что занято одно ядро.
* systemPeriod (стр 14) - означает использование CPU системой. Обычно это значение невысоко, высокие его значения могут свидетельствовать о проблеме с конфигами ядра, проблему со стороны драйвера, или целый ряд других вещей.
* irqPeriod (стр 15) - означает прерывания на уровне железа; на плате электроны движутся по микросхемам предсказуемым образом. Например, когда сетевая карта получает пакет, перед передачей информации, содержащейся в пакете в процессор через ядро, она запросит прерывание в канале прерывания материнской платы. Про- цессор сообщает ядру, что у сетевой карты для него есть информация, а ядро имеет возможность решить, как поступить. Высокое значение времени, тратящегося на обработку прерываний на уровне железа встречается на виртуальной машине до- вольно редко, но по мере того, как гипервизоры предоставляют в распоряжение виртуальных машин все больше «железа», эта ситуация может измениться. Чрез- вычайно высокая пропускная способность сети, использование USB, вычисления на графических процессорах, – все это может привести к росту этого параметра на

величину, превышающую несколько процентов.

* softIrqPeriod (стр 16) - прерывание на уровне софта; начиная с ядра linux версии 2.4 реализована возможность запроса прерывания программным обеспечением (прило- жениями), а не элементом аппаратного обеспечения или устройством (драйвером), запрашивающим прерывание в канале прерывания материнской платы; запрос об- служивается ядром посредством его обработчика прерываний. Это означает, что приложение может запросить приоритетный статус, ядро может подтвердить получе- ние команда, а программное обеспечение будет терпеливо ждать, пока прерывание не будет обслужено. Если мы применим утилиту tcpdump к гигабитному каналу с высоким трафиком, то значение может измениться примерно на 10%, – по мере заполнения выделенной памяти tcpdump, утилита посылает зарос на прерывание, чтобы переместить данные со стека на диск, экран и т.д.
* ioWaitPeriod (стр 17) - процент времени (циклов, секунд), в течение которого процессор простаивал, ожидая завершения операции ввода-вывода. Когда какой-либо процесс или программа запрашивает данные, он сначала проверяет кэш процессора (в нем имеется 2 или 3 кэша), затем проверяет память и, наконец, доходит до диска. Дойдя до диска, процессу или программе обычно приходится ждать, пока поток ввода- вывода передаст информацию в оперативную память, прежде чем иметь возможность снова на нем работать. Чем медленнее диск, тем выше будет значение IO Wait % для каждого процесса. Это происходит также с процессами записи на диск, если системный буфер заполнен и его необходимо прочистить при помощи ядра – обычно это наблюдается на серверах баз данных с высокой нагрузкой. Если значение IO Wait стабильно превышает 100 / (кол-во CPU \* кол-во процессов)%, это означает, что, возможно, имеется проблема хранения, с которой необходимо разобраться. Если вы наблюдаете высокую среднюю нагрузку, прежде всего, проверьте этот параметр. Если он высок, тогда узкое место в процессах, скапливающихся на диске, а не в чем-либо еще.
* stealPeriod (стр 18) - в виртуализированной среде множество логических серверов могут работать под одним фактическим гипервизором. Каждой виртуальной ма- шине(VM) присваивается 4-8 "виртуальных"CPU; хотя сами гипервизоры могут не иметь (кол-во VM \* кол-во виртуальных CPU на одну VM). Причина этого заключа- ется в том, что мы не перегружаем CPU использованием виртуальных машин, так что если дать одной-двум VM возможность изредка использовать 8 процессоров, это не будет негативно влиять на весь пул в целом. Однако если виртуальными процессо- рами VM используется количество CPU, превышающее количество физических (или логических, в случае с гиперпотоковыми процессорами Xeon), тогда значение iosteal

будет расти.

* + guestPeriod (стр 19) - параметр похож на предыдущий, но со стороны виртуальной машины.

Функция CPUMeter\_display (стр. 33) занимается оформленным выводом информации, которую мы описали выше. Граф включения представлен на рисунке 6.

Листинг 5: CPU Meter - центральный процессор(src/top/CPUMeter.c)

1. static void CPUMeter\_ set Values ( Meter \* this , char \* buffer , int size ) {
2. Process List \* pl = this -> pl;
3. int cpu = this -> param ;
4. if ( cpu > this -> pl -> cpu Count ) {
5. snprintf ( buffer , size , " absent ");
6. return ;

7 }

1. CPUData \* cpu Data = &( pl -> cpus [ cpu ]);
2. double total = ( double ) ( cpuData -> total Period == 0 ? 1 : cpuData -> total Period );
3. double percent ;
4. this -> values [0] = cpuData -> nice Period / total \* 100 . 0 ;
5. this -> values [1] = cpuData -> user Period / total \* 100 . 0 ;
6. if ( pl -> detailed CPUTime ) {
7. this -> values [2] = cpuData -> system Period / total \* 100 . 0 ;
8. this -> values [3] = cpuData -> irq Period / total \* 100 . 0 ;
9. this -> values [4] = cpuData -> soft Irq Period / total \* 100 . 0 ;
10. this -> values [5] = cpuData -> io Wait Period / total \* 100 . 0 ;
11. this -> values [6] = cpuData -> steal Period / total \* 100 . 0 ;
12. this -> values [7] = cpuData -> guest Period / total \* 100 . 0 ;
13. this -> type -> items = 8;
14. percent = MIN (100.0 , MAX (0.0 , ( this -> values [0]+ this -> values [1]+ this -> values [2]+
15. this -> values [3]+ this -> values [4]) ));
16. } else {
17. this -> values [2] = cpuData -> system All Period / total \* 100 . 0 ;
18. this -> values [3] = ( cpuData -> steal Period + cpuData -> guest Period ) / total \* 100 . 0 ;
19. this -> type -> items = 4;
20. percent = MIN (100.0 , MAX (0.0 , ( this -> values [0]+ this -> values [1]+ this -> values [2]+ this -> values [3]) ));

28 }

1. if ( isnan ( percent )) percent = 0.0;
2. snprintf ( buffer , size , " %5.1 f%%", percent );

31 }

32

1. static void CPUMeter\_ display ( Object \* cast , Rich String \* out ) {
2. char buffer [50];
3. Meter \* this = ( Meter \*) cast ;
4. Rich String\_ prune ( out );
5. if ( this -> param > this -> pl -> cpu Count ) {
6. Rich String\_ append ( out , CRT\_ colors [ METER\_ TEXT ], " absent ");
7. return ;

40 }

1. sprintf ( buffer , " %5.1 f%% ", this -> values [1]) ;
2. Rich String\_ append ( out , CRT\_ colors [ METER\_ TEXT ], ":");
3. Rich String\_ append ( out , CRT\_ colors [ CPU\_ NORMAL ], buffer );
4. if ( this -> pl -> detailed CPUTime ) {
5. sprintf ( buffer , " %5.1 f%% ", this -> values [2]) ;
6. Rich String\_ append ( out , CRT\_ colors [ METER\_ TEXT ], " sy:");
7. Rich String\_ append ( out , CRT\_ colors [ CPU\_ KERNEL ], buffer );
8. sprintf ( buffer , " %5.1 f%% ", this -> values [0]) ;
9. Rich String\_ append ( out , CRT\_ colors [ METER\_ TEXT ], " ni:");
10. Rich String\_ append ( out , CRT\_ colors [ CPU\_ NICE ], buffer );
11. sprintf ( buffer , " %5.1 f%% ", this -> values [3]) ;
12. Rich String\_ append ( out , CRT\_ colors [ METER\_ TEXT ], " hi:");
13. Rich String\_ append ( out , CRT\_ colors [ CPU\_ IRQ ], buffer );
14. sprintf ( buffer , " %5.1 f%% ", this -> values [4]) ;
15. Rich String\_ append ( out , CRT\_ colors [ METER\_ TEXT ], " si:");
16. Rich String\_ append ( out , CRT\_ colors [ CPU\_ SOFTIRQ ], buffer );
17. sprintf ( buffer , " %5.1 f%% ", this -> values [5]) ;
18. Rich String\_ append ( out , CRT\_ colors [ METER\_ TEXT ], " wa:");
19. Rich String\_ append ( out , CRT\_ colors [ CPU\_ IOWAIT ], buffer );
20. sprintf ( buffer , " %5.1 f%% ", this -> values [6]) ;
21. Rich String\_ append ( out , CRT\_ colors [ METER\_ TEXT ], " st:");
22. Rich String\_ append ( out , CRT\_ colors [ CPU\_ STEAL ], buffer );
23. if ( this -> values [7]) {
24. sprintf ( buffer , " %5.1 f%% ", this -> values [7]) ;
25. Rich String\_ append ( out , CRT\_ colors [ METER\_ TEXT ], " gu:");
26. Rich String\_ append ( out , CRT\_ colors [ CPU\_ GUEST ], buffer );

67 }

1. } else {
2. sprintf ( buffer , " %5.1 f%% ", this -> values [2]) ;
3. Rich String\_ append ( out , CRT\_ colors [ METER\_ TEXT ], " sys :");
4. Rich String\_ append ( out , CRT\_ colors [ CPU\_ KERNEL ], buffer );
5. sprintf ( buffer , " %5.1 f%% ", this -> values [0]) ;
6. Rich String\_ append ( out , CRT\_ colors [ METER\_ TEXT ], " low :");
7. Rich String\_ append ( out , CRT\_ colors [ CPU\_ NICE ], buffer );
8. if ( this -> values [3]) {
9. sprintf ( buffer , " %5.1 f%% ", this -> values [3]) ;
10. Rich String\_ append ( out , CRT\_ colors [ METER\_ TEXT ], " vir :");

|  |  |  |  |
| --- | --- | --- | --- |
| 78 |  |  | Rich String\_ append ( out , CRT\_ colors [ CPU\_ GUEST ], buffer ); |
| 79 |  |  | } |
| 80 |  | } |  |
| 81 | } |  |  |

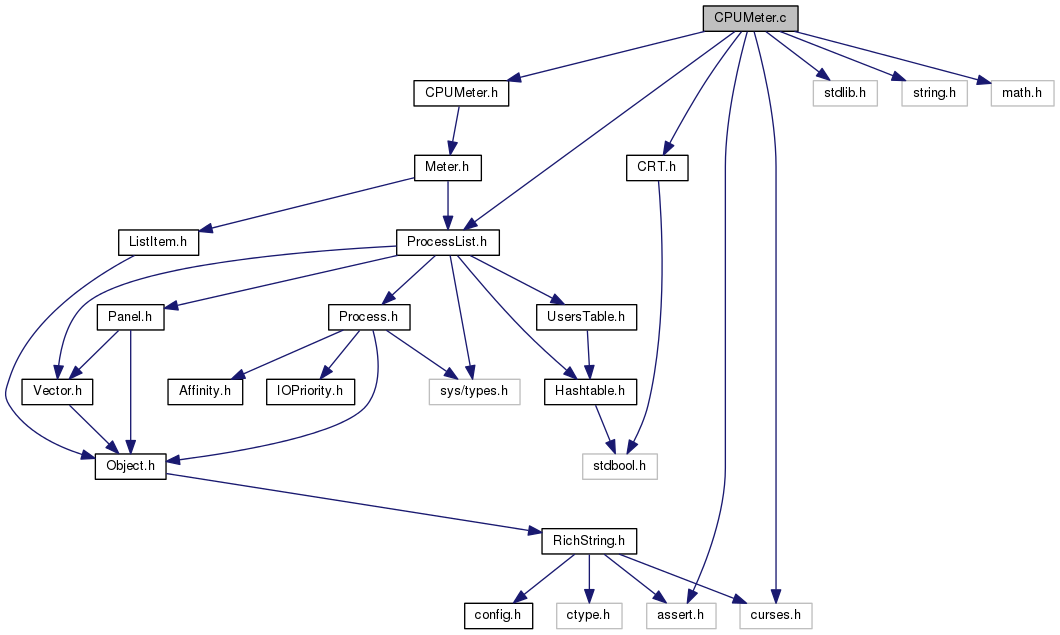


Рис. 6: Граф включения для файла CPUMeter.c

# 6 Имя устройства (хоста)

Этот модуль является самым простым. Его работа сводится фактически к одной строке, которая записывает имя хоста в переданный буфер. Функция gethostname является стан- дартной (POSIX), и она определена в заголовочном файле unistd.h. Вызов этой функции обёрнут другой функцией HostnameMeter\_setValues, которая показана в листинге 5. Про- стота этой функции не отменяет её значимости, которая показана в графе включения на рисунке 7.

Листинг 6: Hostname Meter - имя устройства (хоста) (src/top/HostnameMeter.c)

1. static void Hostname Meter\_ set Values ( Meter \* this , char \* buffer , int size ) {
2. ( void ) this ;
3. gethostname ( buffer , size -1) ;

4 }

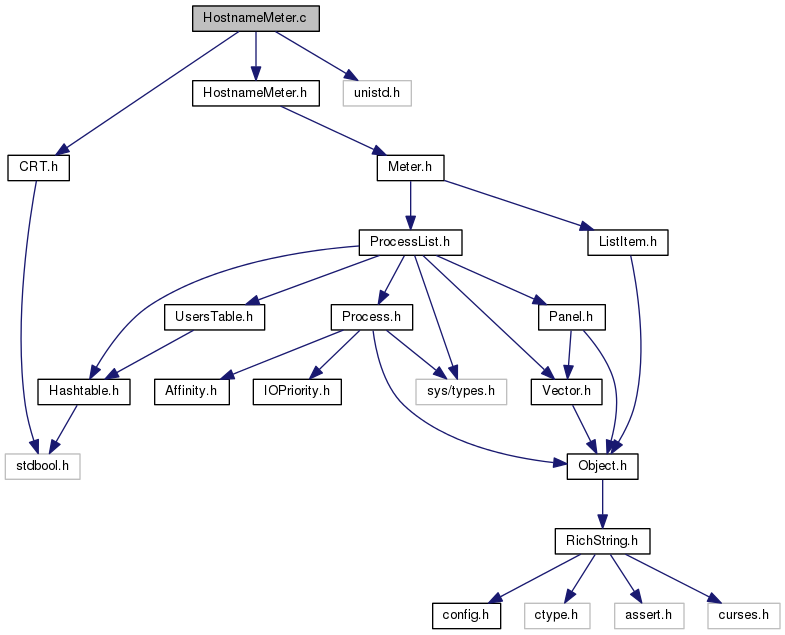


Рис. 7: Граф включения для файла HostnameMeter.c

# 7 Измерение средней загрузки

Большую часть задачи, как и раньше, берёт на себя ядро. В файловой системе proc оно создаёт файл loadavg. Этот файл состоит из пяти групп[2]:

* первое поле (число) — показывает использование процессора за последнюю минуту;
* второе поле (число) — показывает использование процессора за последние пять минут;
* третье поле (число) — показывает использование процессора за последние десять минут;
* четвёртое поле (число/число) — состоит из двух значений, разделённых слешем, первая часть значения поля показывает количество выполняющихся в данный момент процессов/потоков (это значение не может быть больше количества присутствующих в системе CPU), вторая часть поля отображает количество процессов присутствующих в системе;
* пятое поле (число) — хранит ID последнего запущенного в системе процесса (если для просмотра содержимого /proc/loadavg была использована команда cat, то будет значение PID именно программы cat).

В листинге 6 представлен отрывок файла LoadAverageMeter.c. Функция LoadAverageMeter\_scan (стр. 1) как раз обеспечивает открытие файла loadavg (стр. 4) и чтение пяти рассмотренных выше полей (стр. 6), при этом четвёртое поле читается как два различных числа.

Функция LoadAverageMeter\_setValues (стр. 14) обращается к LoadAverageMeter\_scan и сохраняет полученные значения по загрузке процессора за последнюю минуту, пять минут и десять минут в буфер (стр. 16), переданный в качестве аргумента.

Для отображения значений используется функция LoadAverageMeter\_display (стр 19). Она обеспечивает форматированный и цветной вывод для отображения загрузки процессора за последнюю минуту (стр. 22), пять минут (стр. 24) и десять минут (стр. 26).

Функции LoadMeter\_setValues (стр. 30) и LoadMeter\_display (стр 39) также обращаются к функции LoadAverageMeter\_scan, но они отображают только загрузку за последнюю минуту, для просмотра изменения загрузки в режиме (относительно) реального времени.

Листинг 7: Load Average Meter - измерение средней загрузки (src/top/LoadAverageMeter.c)

1. static inline void Load Average Meter\_ scan ( double \* one , double \* five , double \* fifteen ) {
2. int active Procs , totalProcs , last Proc ;
3. \* one = 0; \* five = 0; \* fifteen = 0;
4. FILE \* fd = fopen ( PROCDIR "/ loadavg ", " r");
5. if ( fd) {
6. int total = fscanf ( fd , " %32 lf %32 lf %32 lf %32 d /%32 d %32 d", one , five , fifteen ,
7. & active Procs , & total Procs , & last Proc );
8. ( void ) total ;
9. assert ( total == 6);
10. fclose ( fd);

11 }

12 }

13

1. static void Load Average Meter\_ set Values ( Meter \* this , char \* buffer , int size )

{

1. Load Average Meter\_ scan (& this -> values [2] , & this -> values [1] , & this -> values [0]) ;
2. snprintf ( buffer , size , " %.2 f /%.2 f /%.2 f", this -> values [2] , this -> values

[1] , this -> values [0]) ;

17 }

18

1. static void Load Average Meter\_ display ( Object \* cast , Rich String \* out ) {
2. Meter \* this = ( Meter \*) cast ;
3. char buffer [20];
4. sprintf ( buffer , " %.2 f ", this -> values [2]) ;
5. Rich String\_ write ( out , CRT\_ colors [ LOAD\_ AVERAGE\_ FIFTEEN ], buffer );
6. sprintf ( buffer , " %.2 f ", this -> values [1]) ;
7. Rich String\_ append ( out , CRT\_ colors [ LOAD\_ AVERAGE\_ FIVE ], buffer );
8. sprintf ( buffer , " %.2 f ", this -> values [0]) ;
9. Rich String\_ append ( out , CRT\_ colors [ LOAD\_ AVERAGE\_ ONE ], buffer );

28 }

29

1. static void Load Meter\_ set Values ( Meter \* this , char \* buffer , int size ) {
2. double five , fifteen ;
3. Load Average Meter\_ scan (& this -> values [0] , & five , & fifteen );
4. if ( this -> values [0] > this -> total ) {
5. this -> total = this -> values [0];

35 }

36 snprintf ( buffer , size , " %.2 f", this -> values [0]) ;

37 }

38

39 static void Load Meter\_ display ( Object \* cast , Rich String \* out ) {

40

41

42

43

44

Meter \* this = ( Meter \*) cast ;

char buffer [20];

sprintf ( buffer , " %.2 f ", (( Meter \*) this ) -> values [0]) ; Rich String\_ write ( out , CRT\_ colors [ LOAD ], buffer );

}

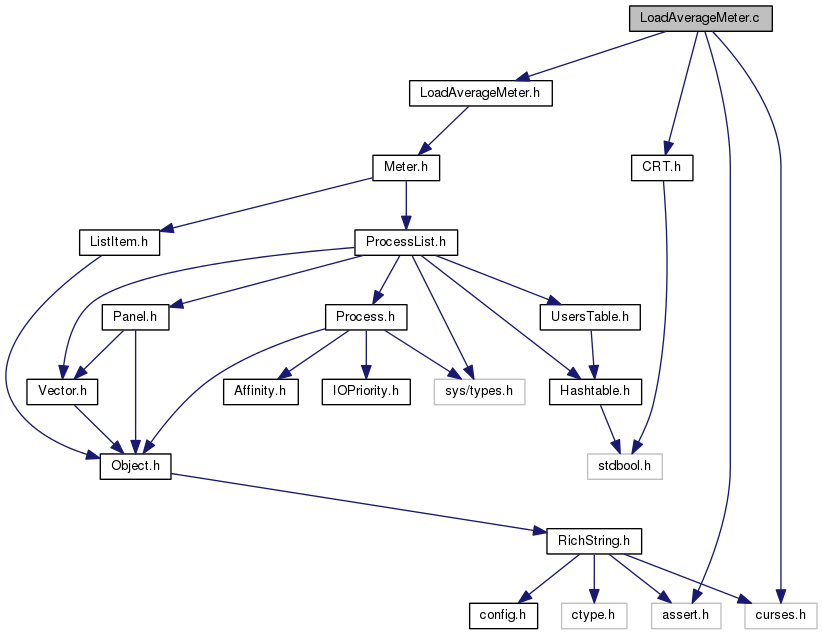


Рис. 8: Граф включения для файла LoadAverageMeter.c

# 8 Измерение уровня использования памяти

Сбор различных характеристик системы уже был показан в листинге 2. В листинге 8 представлены две функции.

Функция MemoryMeter\_setValues (стр. 1) выводит в буфер следующую информацию:

* usedMem (стр. 2) - реально использующая в данный момент и зарезервированная системой память;
* buffersMem (стр. 3) - буферы в памяти это страницы памяти, зарезервированные системой для выделения их процессам, когда они затребуют этого, так же известна как heap-memory;
* cachedMem (стр. 4) - файлы, которые недавно были использованы системой/процес- сами и хранящиеся в памяти на случай если вскоре они снова потребуются.

Функция MemoryMeter\_display (стр. 13) занимается оформленным выводом собранной информации. Граф включения представлен на рисунке 9.

Листинг 8: Memory Meter - измерение уровня использования памяти (src/top/MemoryMeter.c)

1. static void Memory Meter\_ set Values ( Meter \* this , char \* buffer , int size ) {
2. long int used Mem = this -> pl -> used Mem ;
3. long int buffers Mem = this -> pl -> buffers Mem ;
4. long int cached Mem = this -> pl -> cached Mem ;
5. used Mem -= buffers Mem + cached Mem ;
6. this -> total = this -> pl -> total Mem ;
7. this -> values [0] = used Mem ;
8. this -> values [1] = buffers Mem ;
9. this -> values [2] = cached Mem ;
10. snprintf ( buffer , size , "% ld /% ldMB ", ( long int ) used Mem / 1024 , ( long int ) this -> total / 1024) ;

11 }

12

1. static void Memory Meter\_ display ( Object \* cast , Rich String \* out ) {
2. char buffer [50];
3. Meter \* this = ( Meter \*) cast ;
4. int k = 1024 ; const char \* format = "% ldM ";
5. long int total Mem = this -> total / k;
6. long int used Mem = this -> values [0] / k;
7. long int buffers Mem = this -> values [1] / k;
8. long int cached Mem = this -> values [2] / k;
9. Rich String\_ write ( out , CRT\_ colors [ METER\_ TEXT ], ":");
10. sprintf ( buffer , format , total Mem );

23

24

25

26

27

28

29

30

31

32

33

Rich String\_ append ( out , CRT\_ colors [ METER\_ VALUE ], buffer );

sprintf ( buffer , format , used Mem );

Rich String\_ append ( out , CRT\_ colors [ METER\_ TEXT ], " used :"); Rich String\_ append ( out , CRT\_ colors [ MEMORY\_ USED ], buffer ); sprintf ( buffer , format , buffers Mem );

Rich String\_ append ( out , CRT\_ colors [ METER\_ TEXT ], " buffers :"); Rich String\_ append ( out , CRT\_ colors [ MEMORY\_ BUFFERS ], buffer ); sprintf ( buffer , format , cached Mem );

Rich String\_ append ( out , CRT\_ colors [ METER\_ TEXT ], " cache :");

Rich String\_ append ( out , CRT\_ colors [ MEMORY\_ CACHE ], buffer );

}

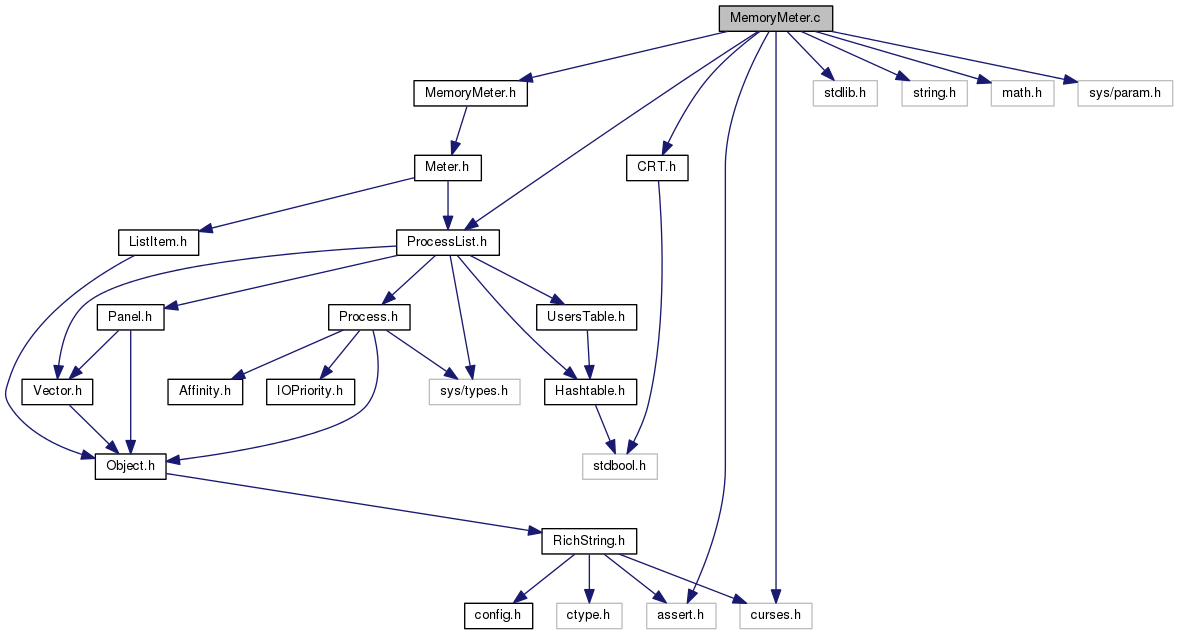


Рис. 9: Граф включения для файла MemoryMeter.c

# 9 Измерение уровня использования области подкачки

Если система приступает к запуску программу, которая требует больше оперативной памяти, чем доступно, то для решения этой задачи, используется технология swapping (”подкачка”). Суть этой технологии заключается в том, что некоторый объем данных (который не “помещается” в оперативную память) временно хранится на жестком диске, в то время как другая часть данных обрабатывается.

В Linux оперативная память делится на разделы, называемые страницами. Swapping (подкачка) – это процесс во время которого страницы памяти копируются на специально сконфигурированный для этого раздел диска, называемый swap space (раздел подкачки, может быть как и файлом, так и разделом жесткого диска), для освобождения ОЗУ. Совокупные размеры физической памяти и раздела подкачки – это объем имеющийся виртуальной памяти.

В листинге 2 мы рассматривали процедуру сбора системной информации, в листинге 9 нас интересуют три функции работы со свапом.

В функции SwapMeter\_humanNumber (стр. 2) размер раздела подкачки приводится к виду, легко воспринимаемому человеком. Если объём превышает 10 гигабайт, то число отображается как соответствующее количество гигабайт (стр 4). Если число превышает 10 мегабайт, то оно отображается как соответствующее количество мегабайт (стр 6). Если ни одно из предыдущих правил не сработало, то объём отображается в килобайтах (стр. 8).

Функция SwapMeter\_setValues (стр. 11) выводит информацию о подкачке в буфер, пе- реданный в качестве параметра. Для этого используется общий объём свапа (стр. 13) и занятый объём (стр. 12).

Функция SwapMeter\_display (стр 18) обеспечивает форматированный вывод всей собранной информации. Граф включения представлен на рисунке 10.

Листинг 9: Swap Meter - измерение уровня использования области подкачки (src/top/SwapMeter.c)

1

1

2

3

4

5

6

7

8

9

0

*/\* NOTE: Value is in kilobytes \*/*

static void Swap Meter\_ human Number ( char \* if (\* value >= 10\* GIGABYTE )

buffer , const long int \* value ) {

sprintf ( buffer ,

else if (\* value >= sprintf ( buffer ,

else

sprintf ( buffer ,

"% ldG ", \* value /

10\* MEGABYTE )

"% ldM ", \* value /

GIGABYTE );

MEGABYTE );

"% ldK ", \* value );

}

1. static void Swap Meter\_ set Values ( Meter \* this , char \* buffer , int len ) {
2. long int used Swap = this -> pl -> used Swap ;
3. this -> total = this -> pl -> total Swap ;
4. this -> values [0] = used Swap ;
5. snprintf ( buffer , len , "% ld /% ldMB ", ( long int ) used Swap / MEGABYTE , ( long int ) this -> total / MEGABYTE );

16 }

17

1. static void Swap Meter\_ display ( Object \* cast , Rich String \* out ) {
2. char buffer [50];
3. Meter \* this = ( Meter \*) cast ;
4. long int swap = ( long int ) this -> values [0];
5. long int total = ( long int ) this -> total ;
6. Rich String\_ write ( out , CRT\_ colors [ METER\_ TEXT ], ":");
7. Swap Meter\_ human Number ( buffer , & total );
8. Rich String\_ append ( out , CRT\_ colors [ METER\_ VALUE ], buffer );
9. Swap Meter\_ human Number ( buffer , & swap );
10. Rich String\_ append ( out , CRT\_ colors [ METER\_ TEXT ], " used :");
11. Rich String\_ append ( out , CRT\_ colors [ METER\_ VALUE ], buffer );

29 }

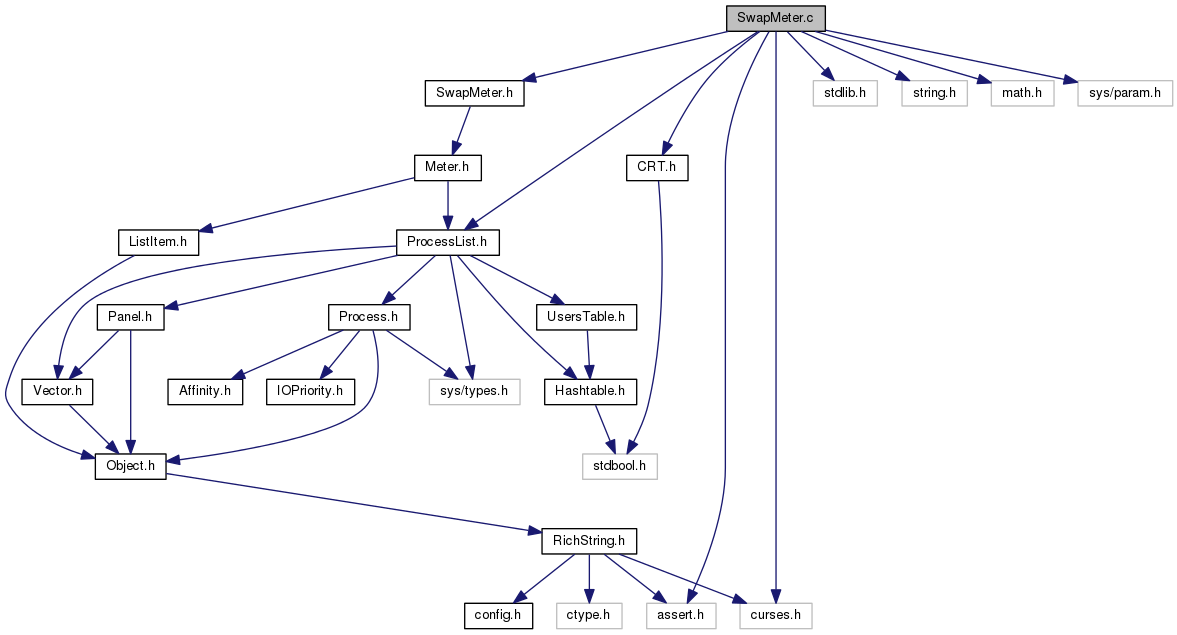


Рис. 10: Граф включения для файла SwapMeter.c

# 10 Мониторинг процессов

Как мы уже говорили, все процессы представлены своей директорией в файловой системе proc. вывод этой информации можно изучить в листинге 10.

Функция TasksMeter\_setValues (стр. 1) занимается в выводом в буфер (стр. 5) информации о процессах. Этой информацией является общее количество процессов (стр. 3) и количество запущенных в данный момент процессов (стр. 4). Как мы говорили выше, количество запущенных процессов не может превышать количество процессоров.

Функция TasksMeter\_display (стр. 8) обеспечивает форматированный вывод информации о процессах. При этом стоит обратить внимание, что она выводит не только процессы пользователя (стр. 22) но и процессы ядра (стр. 28), которые могут заниматься, к примеру, управлением кэшами.

Граф включений представлен на рисунке 11.

Листинг 10: Tasks Meter - мониторинг процессов (src/top/TasksMeter.c)

* 1. static void Tasks Meter\_ set Values ( Meter \* this , char \* buffer , int len ) {
  2. Process List \* pl = this -> pl;
  3. this -> total = pl -> total Tasks ;
  4. this -> values [0] = pl -> running Tasks ;
  5. snprintf ( buffer , len , "% d/% d", ( int ) this -> values [0] , ( int ) this -> total );

6 }

7

1. static void Tasks Meter\_ display ( Object \* cast , Rich String \* out ) {
2. Meter \* this = ( Meter \*) cast ;
3. Process List \* pl = this -> pl;
4. char buffer [20];
5. sprintf ( buffer , "% d", ( int )( this -> total - pl -> userland Threads - pl -> kernel Threads ));
6. Rich String\_ write ( out , CRT\_ colors [ METER\_ VALUE ], buffer );
7. int thread Value Color = CRT\_ colors [ METER\_ VALUE ];
8. int thread Caption Color = CRT\_ colors [ METER\_ TEXT ];
9. if ( pl -> highlight Threads ) {
10. thread Value Color = CRT\_ colors [ PROCESS\_ THREAD\_ BASENAME ];
11. thread Caption Color = CRT\_ colors [ PROCESS\_ THREAD ];

19 }

1. if (! pl -> hide Userland Threads ) {
2. Rich String\_ append ( out , CRT\_ colors [ METER\_ TEXT ], ", ");
3. sprintf ( buffer , "% d", ( int ) pl -> userland Threads );
4. Rich String\_ append ( out , thread Value Color , buffer );
5. Rich String\_ append ( out , thread Caption Color , " thr ");

25 }

26

27

28

29

30

31

32

33

34

35

36

if (! pl -> hide Kernel Threads ) {

Rich String\_ append ( out , CRT\_ colors [ METER\_ TEXT ], ", "); sprintf ( buffer , "% d", ( int ) pl -> kernel Threads ); Rich String\_ append ( out , thread Value Color , buffer ); Rich String\_ append ( out , thread Caption Color , " kthr ");

}

Rich String\_ append ( out , CRT\_ colors [ METER\_ TEXT ], "; "); sprintf ( buffer , "% d", ( int ) this -> values [0]) ; Rich String\_ append ( out , CRT\_ colors [ TASKS\_ RUNNING ], buffer ); Rich String\_ append ( out , CRT\_ colors [ METER\_ TEXT ], " running ");

}

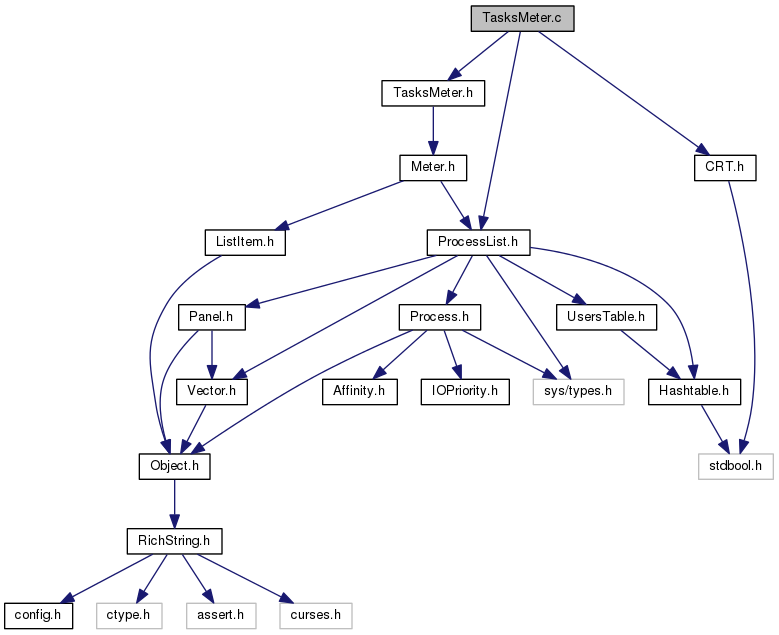


Рис. 11: Граф включения для файла TasksMeter.c

# 11 Измерение времени работы системы

Время работы системы также хранится в файловой системе proc в файле uptime. В листинге 11 представлена функция UptimeMeter\_setValues (стр. 1), которая открывает этот файл (стр. 3) и считывает оттуда значение (стр. 5). Стоит отметить, что на самом деле в этом файле хранится два числа.

* + первое показывает количество секунд, прошедших с момента включения компьютера (это значение используется в дальнейшем)
  + второе показывает количество времени (тоже в секундах), которое система провела в бездействии (в т.ч. ожидая завершения операций ввода-вывода); на многоядерных системах это число складывается из времени бездействия каждого ядра, так что второе число по значению может обогнать первое.

Число секунд с момента включения приводится к виду, удобному для человеческого восприятия - в буфер выводится (стр. 27) количество дней, часов, минут и секунд работы.

Листинг 11: Uptime Meter - измерение времени работы системы (src/top/UptimeMeter.c)

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

static void Uptime Meter\_ set Values ( Meter \* this ,

double uptime = 0;

FILE \* fd = fopen ( PROCDIR "/ uptime ", " r");

char \* buffer , int len ) {

if

( fd) {

fscanf ( fd , " %64 lf", & uptime ); fclose ( fd);

}

int int int int int

totalseconds = ( int ) ceil ( uptime );

seconds

minutes hours =

= totalseconds % 60;

= ( totalseconds /60) % 60; ( totalseconds / 3600 ) % 24;

days = ( totalseconds / 86400 ) ;

this -> values [0] = days ; if ( days > this -> total )

this -> total = days ;

}

char daysbuf [15]; if ( days > 100) {

sprintf ( daysbuf , "% d

{

days (!) , ", days );

}

}

else if ( days > 1) {

sprintf ( daysbuf , "% d days , ", days ); else if ( days == 1) {

sprintf ( daysbuf , " 1 day , ");

} else {

daysbuf [0] = ’\0 ’;

26

27

}

snprintf ( buffer , len , "% s %02 d :%02 d :%02 d", daysbuf , hours , minutes , seconds );

28 }

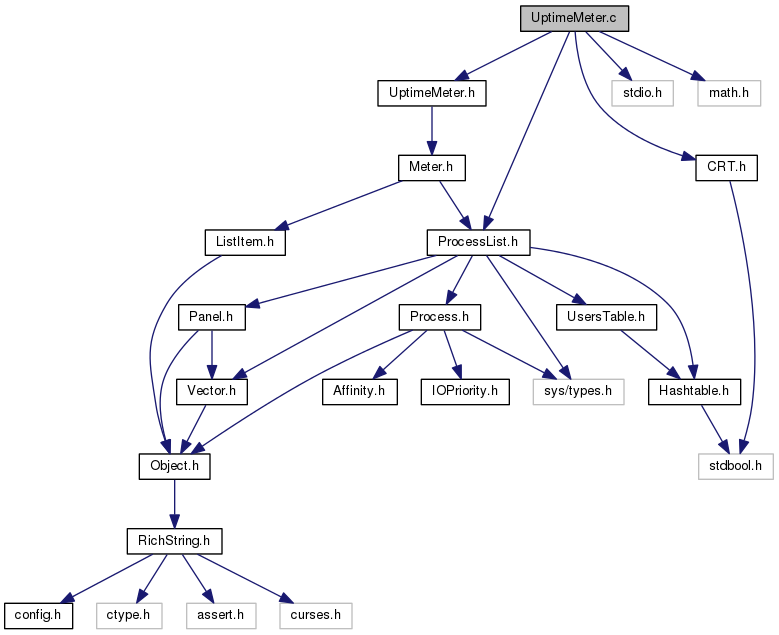


Рис. 12: Граф включения для файла UptimeMeter.c

# 12 Модификация для работы с процессом

Как уже говорилось во введении, на верхнем уровне процессы представляют собой ди- ректории, именованные в соответствии с их pid. Рассмотрим, какую информацию можно получить из файловой системы proc, а потом добавим рассматриваемой утилите возмож- ность работать с процессом, переданным в качестве параметра по его pid.

Некоторые файлы и директории из ProcFS[2]:

* /proc/PID/cmdline – аргументы командной строки (где PID – идентификатор процесса или self);
* /proc/PID/environ – переменные окружения для данного процесса;
* /proc/PID/status – статус процесса;
* /proc/PID/fd – директория, содержащая символьные ссылки на каждый открытый файловый дескриптор;
* /proc/cpuinfo – информация о процессоре (производитель, модель, поколение и т.п.);
* /proc/cmdline – параметры, передаваемые ядру при загрузке;
* /proc/uptime – количество секунд, прошедших с момента загрузки ядра и проведенных в режиме бездействия;
* /proc/version – содержит информацию о версии ядра, компилятора и другую инфор- мацию, связанную с загруженным ядром.

Приведём несколько примеров использования procfs.

Следующая команда демонстрирует, как из procfs можно получить информацию о текущей рабочей директории процесса (3165 — номер pid’a процесса)

# ls -la /proc/3165/cwd

lrwxrwxrwx 1 clamav clamav 0 Авг 18 16:07 /proc/3165/cwd -> /var/lib/clamav

Далее, можно вывести все переменные процесса

# cat /proc/3165/environ | strings ReceiveTimeout=30 CONSOLE=/dev/console SELINUX\_INIT=YES

TERM=linux rootmnt=/root

PidFile=/var/run/clamav/freshclam.pid

NotifyClamd=/etc/clamav/clamd.conf LogTime=no

INIT\_VERSION=sysvinit-2.86 init=/sbin/init DNSDatabaseInfo=current.cvd.clamav.net AllowSupplementaryGroups=false PATH=/sbin:/usr/sbin:/bin:/usr/bin LogSyslog=false

DatabaseMirror=database.clamav.net db.local.clamav.net runlevel=2

RUNLEVEL=2 PWD=/ VERBOSE=no

DatabaseOwner=clamav CompressLocalDatabase=no previous=N

PREVLEVEL=N

LogVerbose=false MaxAttempts=5 ScriptedUpdates=yes Foreground=false Checks=24

SHLVL=3 HOME=/

DatabaseDirectory=/var/lib/clamav/ LogFacility=LOG\_LOCAL6 UpdateLogFile=/var/log/clamav/freshclam.log LogFileMaxSize=0

ConnectTimeout=30 Debug=false

\_=/sbin/start-stop-daemon

Теперь можно получить статистику процесса

$ cat /proc/1742/status Name: bash

State: S (sleeping) Tgid: 3515

Pid: 3515

PPid: 3452

|  |  |  |  |
| --- | --- | --- | --- |
| TracerPid: | 0 |  | |
| Uid: 1000 | 1000 | 1000 | 1000 |
| Gid: 100 | 100 | 100 | 100 |
| FDSize: 256 |  |  |  |
| Groups: 16 33 | 100 |  |  |

VmPeak: 9136 kB

VmSize: 7896 kB

VmLck: 0 kB

VmHWM: 7572 kB

VmRSS: 6316 kB

VmData: 5224 kB

VmStk: 88 kB

VmExe: 572 kB

VmLib: 1708 kB

VmPTE: 20 kB

Threads: 1

SigQ: 0/3067

SigPnd: 0000000000000000

ShdPnd: 0000000000000000

SigBlk: 0000000000010000

SigIgn: 0000000000384004

SigCgt: 000000004b813efb CapInh: 0000000000000000

CapPrm: 0000000000000000

CapEff: 0000000000000000

CapBnd: ffffffffffffffff Cpus\_allowed: 00000001

Cpus\_allowed\_list: 0

Mems\_allowed: 1

Mems\_allowed\_list: 0

voluntary\_ctxt\_switches: 150

nonvoluntary\_ctxt\_switches: 545

Самое полезное в данном выводе, это поле Threads. Оно показывает количество потоков у процесса (ещё эту информацию можно получить через утилиту ps)[3]. Теперь остаётся внести изменения в исследуемую утилиту.

Поле с потоками называется NLWP (Number of Light-Weight Processes). Это название пошло со времён ОС Солярис.

Для работы этого функционала в структуре ProcessList\_ (объявлена в заголовочном файле Process.h) объявлена переменная nlwp типа long int (см. стр. 33 в листинге 1). Чтение из status-файла происходит в функции ProcessList\_readStatFile, листинг которой представлен в листинге 12.

Листинг 12: Чтение stat-файла процесса (src/top/ProcessList.c)

1. static bool Process List\_ read Stat File ( Process \* process , const char \* dirname , const char \* name , char \* command ) {
2. char filename [ MAX\_ NAME +1];
3. snprintf ( filename , MAX\_NAME , "% s/% s/ stat ", dirname , name );
4. FILE \* file = fopen ( filename , " r");
5. if (! file )
6. return false ;

7

8 static char buf [ MAX\_ READ ];

9

1. int size = fread ( buf , 1 , MAX\_READ , file );
2. if (! size ) { fclose ( file ); return false ; }

12

1. *// assert( process -> pid == atoi( buf));*
2. char \* location = strchr ( buf , ’ ’);
3. if (! location ) { fclose ( file ); return false ; }

16

1. location += 2;
2. char \* end = strrchr ( location , ’)’);
3. if (! end ) { fclose ( file ); return false ; }

20

1. int commsize = end - location ;
2. memcpy ( command , location , commsize );
3. command [ commsize ] = ’\0 ’;
4. location = end + 2;

25

26 int num = sscanf ( location ,

27 "% c % d % u % u % u "

28 "% d % lu "

29 "%\* u %\* u %\* u %\* u "

1. "% llu % llu % llu % llu "
2. "% ld % ld % ld "

32 "%\* d %\* u %\* u %\* d %\* u %\* u %\* u %\* u %\* u %\* u %\* u %\* u %\* u %\* u %\* u %\* u %\* u "

33 "% d % d",

34 & process -> state , & process -> ppid , & process -> pgrp , & process -> session , & process -> tty\_nr ,

35

36

37

38

39

40

41

& process -> tpgid , & process -> flags ,

& process -> utime , & process -> stime , & process -> cutime , & process -> cstime ,

& process -> priority , & process -> nice , & process -> nlwp ,

& process -> exit\_ signal , & process -> processor ); fclose ( file );

return ( num == 16) ;

}

Функция получает путь к proc (dirname) и номер процесса (name) в качестве параметров, из которых складывается полный путь к stat-файлу (стр 3). Этот файл открывается на чтение (стр стр. 4) и после некоторых проверок из него выбираются значения, в т.ч. количество процессов (стр. 37).

Запуск модифицированной версии, для отслеживания потоков firefox

sam@spb:~/tmp/top$ ./top --pid 19771

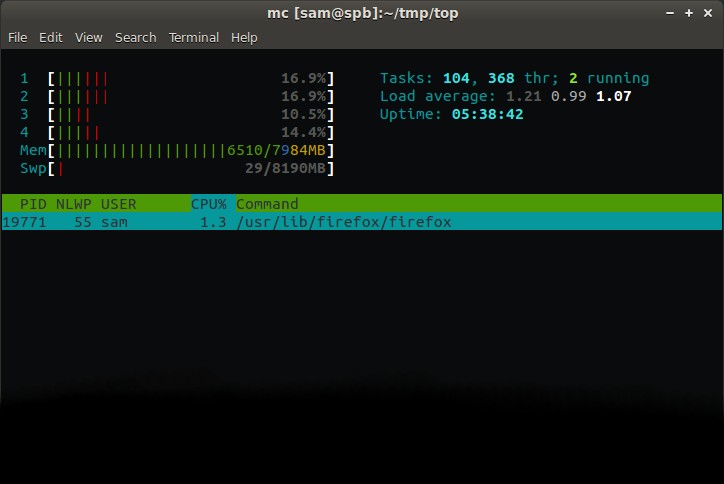


Рис. 13: Количество потоков у процесса с pid 19771

# Заключение

В данной работе нами была изучена программа htop, являющееся расширенной верси- ей стандартной утилиты top. Мы выпустили из рассмотрения особенности, связанные с графическим выводом (на базе библиотеки ncurces), сосредоточившись на основных функциональных возможностях.

Изучив исходный код, мы убедились, что никакие специальные системные вызовы утилита не использует, только самые широко распространённы, вроде чтения из файла и вывод на экран (но они используются так часто, что их систематизация для данного отчёта оказалась крайне затруднительной).

Основным выводом является тот факт, что в отличии от Windows, Linux предоставляет удобный механизм сбора системной информации через файловую систему proc, который активно используется утилитами вроде ps, htop (стр. 5 листинг 2; стр. 18 листинг 3; стр. 136 листинг 3; стр. 201 листинг 3; стр. 4 листинг 7; стр. 3 листинг 11; стр. 26 листинг 13).

Файловая система proc обладает обширными возможностями по конфигурированию linux, в то же время её использование требует предельной осторожности, так как попытка записи в некоторые файлы может повредить файловую систему или привести к краху системы.

# Список литературы

1. Яремчук С. А. Linux Mint на 100 %. – СПб.: Питер, 2011. – 240 е.: ил. — (Серия «На 100 %»). ISBN: 978-5-49807-803-8.
2. HowTo: Troubleshoot with linux ’top’ command. David Van Rood (dowdandassociates.com).
3. Арнольд Роббинс. Linux. Программирование в примерах – СПб.: КУДИЦ-Пресс, 2006 – 256 стр.