

«Научно-технический центр Метротек»

Лабораторная работа №1

Тема: Сбор статистики по сетевому трафику

Стажёр: Бронников Егор Игоревич

[<bronnikov.40@mail.ru>](mailto:bronnikov.40@mail.ru)

[<t.me/endygamedev>](https://t.me/endygamedev)

Санкт-Петербург

2022

Содержание

Постановка задачи	3
1. Описание	6
1.1. Первый вариант	6
1.2. Второй вариант	6
1.3. Структура исходников	7
2. Сборка	9
2.1. Исходники	9
2.2. deb-пакет	9
3. Запуск	11
3.1. ps-scanner-1 и ps-scanner-2	11
3.2. ps-stats	12
3.3. Скрипты	13
3.4. Тесты	15
4. Профилирование	17
5. Авторство и лицензия	19
Приложение А	21

Постановка задачи

Цель задачи: создать набор программного обеспечения, который мог бы собирать и отображать статистику по трафику на заданном сетевом интерфейсе.

Требования:

1. ПО должно работать на ПК под управлением Debian GNU/Linux (версии 10 и новее).
2. Для реализации использовать язык программирования C.
3. Сборка должна осуществляться GNU Toolchain.
4. Дистрибуция должна осуществляться при помощи deb-пакета.
5. Сбор статистики должен вестись только по входящим UDP пакетам.
6. Должна быть реализована возможность указывать конкретные параметры учитываемых в статистике пакетов:
 1. IP-адрес источника
 2. IP-адрес назначения
 3. Порт источника
 4. Порт назначения
7. В статистике должно присутствовать количество принятых пакетов и суммарное количество байт в этих пакетах.

ПО организовать в виде двух отдельных утилит: первая читает данные с сетевого интерфейса и собирает статистику по пакетам, вторая при запуске получает собранную статистику у первой утилиты и выводит её на экран.

Утилита для сбора статистики. Нужно реализовать 2 варианта данной утилиты:

1. Два потока (pthread): первый читает пакеты при помощи Raw Socket (OSI L2) с интерфейса, проверяет параметры пакета и для

подходящих по заданным параметрам, передаёт статистику во второй поток. Второй суммирует статистику и отдаёт её по запросу извне.

2. Два потока (pthread): первый читает пакеты при помощи Raw Socket (OSI L2) с интерфейса, проверяет параметры пакета и для подходящих по заданным параметрам, суммирует статистику. Второй отдаёт её по запросу извне.

Самостоятельно провести профилирование обоих вариантов, оценить какой вариант эффективнее, с каким вариантом можно обеспечить большую пропускную способность.

В обоих вариантах: передача данных между потоками осуществляется любым способом, на усмотрение разработчика.

Утилита для вывода статистики на экран: запрашивает статистику у первой утилиты через `ibus` или через `POSIX Message Queues`. Рекомендация: попробовать реализовать оба варианта.

Программное обеспечение должно сопровождаться документацией, содержащей следующие разделы:

1. Описание - общая информация, что и как делает ПО.
2. Сборка - инструкции по сборке ПО из исходников: что установить в систему, какой командой запустить сборку, что должно получиться в итоге.
3. Запуск - как запустить ПО, как подать трафик на интерфейс, чтобы убедиться в корректности работы, что пользователь программ должен увидеть на экране.
4. Результаты профилирования двух реализованных вариантов утилиты для сбора статистики.
5. Авторство и лицензия - указать имя и электронную почту автора, указать лицензию.

Результат работы: архив `git` репозитория, содержащего исходники ПО и сопроводительную документацию.

Вспомогательная информация:

- The Linux Programming Interface: <https://man7.org/tlpi/>
- GCC: <https://gcc.gnu.org>
- pthread: <https://man7.org/linux/man-pages/man7/pthreads.7.html>
- Message Queues: https://man7.org/linux/man-pages/man7/mq_overview.7.html
- Raw Sockets: <https://man7.org/linux/man-pages/man7/raw.7.html>
- ubus: <https://openwrt.org/docs/techref/ubus>
- Сборка deb-пакета: <https://www.debian.org/doc/devel-manuals#debmake-doc>
- Профилирование: <https://www.brendangregg.com/>
- Git: <https://git-scm.com/book/en/v2>

1. Описание

Данная лабораторная работа была разделена на два варианта реализации утилит.

1.1 Первый вариант

Данный вариант предусматривает, что создаются два потока (*pthread*): первый читает пакеты при помощи Raw Socket (OSI L2) с интерфейса, проверяет параметры пакета и для подходящих по заданным параметрам, передаёт статистику во второй поток. Второй суммирует статистику и отдаёт её по запросу извне.

Передача статистики пользователю осуществляется через *POSIX Message Queues*.

Передача статистики между потоками осуществляется через каналы (*pipe*).

1.2 Второй вариант

Эта утилита создаёт два потока (*pthread*). Первый поток читает пакеты при помощи Raw Socket (OSI L2) с интерфейса, проверяет параметры пакета и для подходящих по заданным параметрам, суммирует статистику. Второй поток отдаёт её по запросу извне.

Передача статистики пользователю осуществляется через *POSIX Message Queues*.

Для работы захвата пакетов обязательно нужно указать сетевой интерфейс, с которого будут считываться пакеты.

В статистике отражается количество принятых пакетов и суммарное количество байт в этих пакетах. Также существует возможность фильтрации пакетов по следующим параметрам:

1. IP-адрес источника
2. IP-адрес назначения
3. Порт источника
4. Порт назначения

1.3 Структура исходников

Весь исходный код данного проекта содержится в папке **src/**:

```
src/  
    basic.c  
    colors.h  
    Makefile  
    ps-scanner-1.c  
    ps-scanner-2.c  
    ps-scanner.h  
    ps-stats.c
```

В паке **src/** содержатся два варианта реализации утилиты для сбора статистики: **ps-scanner-1.c** — первый вариант реализации, **ps-scanner-2.c** — второй вариант реализации.

Описание файлов

ps-scanner-1.c:

Данный файл содержит системную утилиту, которая собирает информацию о пакетах (первым вариантом реализации) и передаёт её пользователю через *POSIX Message Queues*.

`ps-scanner-2.c`:

Данный файл содержит системную утилиту, которая собирает информацию о пакетах (вторым вариантом реализации) и передаёт её пользователю через *POSIX Message Queues*.

`ps-scanner.h`:

Заголовочный файл для `ps-scanner-1.c` и `ps-scanner-2.c`, который содержит прототипы функций.

`basic.c`

В данный файл вынесены общие функции, которые не связаны логикой с `ps-scanner-1.c` и `ps-scanner-2.c`.

`ps-stats.c`:

В данном файле содержатся функции, которые выводят статистику для пользователя. Данные приходят от `ps-scanner` через *POSIX Message Queues*.

`colors.h`:

Данный файл содержит константы цветов для более приятного пользовательского интерфейса.

2. Сборка

Зависимости

- Build Essential

```
$ sudo apt-get install build-essential
```

- (*optional*) Linux Tools для профилирования

```
$ sudo apt-get install linux-tools-common linux-tools-generic
```

```
$ sudo apt-get install linux-tools-`uname -r`
```

2.1 Исходники

Для сборки проекта можно использовать Makefile.

```
$ git clone https://github.com/andygamedev/network-traffic.git
```

```
$ cd ./network-traffic/src/
```

```
$ make
```

2.2 deb-пакет

Для сборки проекта можно воспользоваться deb-пакетом. В пакете будет содержаться два варианта реализации сбора статистики (ps-scanner-1 и ps-scanner-2).

Установка

1. Клонировать репозиторий с проектом:

```
$ git clone https://github.com/andygamedev/network-traffic.git
```

2. Переходим в директорию для создания deb-пакета:

```
$ cd ./network-traffic/deb/
```

3. Собираем запускаемые файлы (ps-scanner-1, ps-scanner-2 и ps-stats):

```
$ make
```

4. Собираем и устанавливаем deb-пакет:

```
$ make install
```

Проверка установки пакета

```
$ dpkg -l | grep packet-sniffer
```

Если всё установилось корректно, то должно получиться следующее сообщение:

```
agor@ubuntu:~$ dpkg -l | grep packet-sniffer
ii packet-sniffer 1.0 amd64 Utilities for sniffing packages by specified parameters.
```

Удаление

```
$ make clean
```

ИЛИ

```
$ sudo dpkg -r packet-sniffer
```

3. Запуск

Установленный deb-пакет содержит три утилиты:

1. `ps-scanner-1` – утилита для сбора статистики (первый вариант реализации);
2. `ps-scanner-2` – утилита для сбора статистики (второй вариант реализации);
3. `ps-stats` – пользовательская утилита, которая предоставляет собранную статистику.

3.1 `ps-scanner-1` и `ps-scanner-2`

`ps-scanner-1` и `ps-scanner-2` получают на вход 1 обязательный аргумент и 4 опциональных аргумента, которые помогают отфильтровать нужные пакеты:

- `-ni` или `--interface` – *!(обязательный аргумент)* сетевой интерфейс, с которого будут считываться пакеты.
- `-ips` или `--ip_source` – IP-адрес источника отслеживаемых пакетов;
- `-ipd` или `--ip_dest` – IP-адрес назначения отслеживаемых пакетов;
- `-ps` или `--port_source` – порт источника отслеживаемых пакетов;
- `-pd` или `--port_dest` – порт назначения отслеживаемых пакетов.

Запуск `ps-scanner-1` и `ps-scanner-2`:

```
$ sudo ps-scanner-1 -ni <interface name> -ips <IP source> -ipd <IP  
dest> -ps <port source> -pd <port dest>
```

```
$ sudo ps-scanner-2 -ni <interface name> -ips <IP source> -ipd <IP  
dest> -ps <port source> -pd <port dest>
```

Пример запуска `ps-scanner-1` (аналогично для `ps-scanner-2`):

```
$ sudo ps-scanner-1 -ni lo -ipd 192.168.1.2 -pd 9999
```

В данном примере отбираются только те пакеты с сетевого интерфейса `lo`, у которых IP-адрес назначения – `192.168.1.2` и порт назначения – `9999`.

Также в файл `/var/log/ps-scanner.log` записываются все пакеты, которые прошли этап отбора, чтобы можно было проверить результат работы программы.

Список всех доступных сетевых интерфейсов можно узнать следующим способом:

```
$ netstat -i
```

или

```
$ ip link show
```

3.2 ps-stats

Данная утилита идёт без аргументов и связывается с утилитой `ps-scanner-1` или `ps-scanner-2` по названию (идентификатору) очереди (Message Queue Name).

`ps-stats` запускается после `ps-scanner-1` или `ps-scanner-2`, если попытаться выводить статистику без предварительного сбора, то будет напечатано сообщение с ошибкой.

Запуск `ps-stats`:

```
$ sudo ps-stats
```

3.3 Скрипты

Для проверки работоспособности утилит можно воспользоваться файлами из папки `tests/`. Они создают локальный UDP-сервер и отправляют сообщения (пакеты) пользователя.

```
tests/  
    Makefile  
    nc-client  
    nc-server  
    ps-scanner-test  
    ps-stats-test  
    py-client.py
```

`ps-scanner-test:`

```
#!/bin/bash  
sudo ps-scanner-$1 -ni lo -ipd $(hostname -I | awk '{print $1}') -pd  
    9999
```

Данный скрипт принимает 1 обязательный позиционный аргумент, который определяет какую реализацию сбора статистики необходимо запустить (1 или 2). Запускается сканер на сетевом интерфейсе `lo` (Loopback interface), который отбирает только те пакеты, у которых IP-адрес назначения `localhost`, а порт назначения `9999`.

Также если не собирать `deb`-пакет, то можно протестировать данную утилиту из исходников:

```
#!/bin/bash  
sudo ../src/ps-scanner-$1 -ni lo -ipd $(hostname -I | awk '{print  
    $1}') -pd 9999
```

`ps-stats:`

```
#!/bin/bash
sudo ps-stats
```

В данном скрипте запускается вывод собранной статистики `ps-scanner`.

Также если не собирать `deb`-пакет, то можно протестировать данную утилиту из исходников:

```
#!/bin/bash
sudo ../src/ps-stats
```

`nc-server:`

```
#!/bin/bash
nc -u -l 9999
```

С помощью утилиты `netcat` запускается UDP-сервер на порте 9999, который ожидает сообщений от клиентов.

`nc-client:`

Данная утилита принимает 4 опциональных аргумента:

- `-c` – количество посланных пользователем сообщений;
- `-m` – текст сообщения, которое будет послано;
- `-s` – IP-адрес сервера;
- `-p` – порт сервера.

То есть после запуска данной программы на сервер `$count`-раз отправится сообщение `$message`.

`py-client.py`:

Также вместо утилиты `nc-client` можно воспользоваться данной утилитой. Данная утилита принимает 4 опциональных аргумента:

- `-c` – количество посланных пользователем сообщений;
- `-m` – текст сообщения, которое будет послано;
- `-s` – IP-адрес сервера;
- `-p` – порт сервера.

То есть после запуска данной программы на сервер `count`-раз отправится сообщение `message`.

3.4 Тесты

Итак, для тестирования, с помощью `netcat`, создаётся локальный UDP-сервер, на который посылаются UDP-пакеты (сообщения) клиентов.

1. Для начала стоит перейти в папку с тестом:

```
$ cd ./tests/
```

2. Затем в запускается UDP-сервер:

```
$ ./nc-server
```

3. После чего можно запустить `ps-scanner-1`:

```
$ ./ps-scanner-test 1 # first implementation test
```

или `ps-scanner-2`:

```
$ ./ps-scanner-test 2 # second implementation test
```

4. Далее стоит запустить `ps-stats`:

```
$ ./ps-stats
```

5. После чего можно отправлять различные пакеты на сервер и смотреть как отображается статистика:

```
$ ./nc-client
```

или

```
$ ./py-client
```

6. Также собранные пакеты можно посмотреть:

```
$ cat /var/log/ps-scanner.log
```

Видеопример последовательного запуска программ представлен в репозитории GitHub.

4. Профилирование

Профилирование проводилось с помощью утилиты perf.

Профилировались утилиты ps-scanner-1 и ps-scanner-2. Была собрана информация о (%) загрузженности CPU, функциях программ в пользовательском пространстве и функциях ядра.

С помощью perf record была собрана данная статистика и сохранена в директории ./profiling/ver1/perf.data для ps-scanner-1 и ./profiling/ver2/perf.data для ps-scanner-2.

Оба сборщика статистик запускались при следующем тесте: отправлялись 10000 пакетов, 1000 пакетов и затем снова 10000 пакетов с сообщением "Hello".

Samples: 25K of event 'cycles', Event count (approx.): 3414381966				Samples: 17K of event 'cycles', Event count (approx.): 2189833718			
Overhead	Command	Shared Object	Symbol	Overhead	Command	Shared Object	Symbol
5.85%	ps-scanner-1	[kernel.kallsyms]	[k] syscall_exit_to_user_mo	5.26%	ps-scanner-2	libc-2.31.so	[.] __vfprintf_internal
3.35%	ps-scanner-1	libc-2.31.so	[.] __vfprintf_internal	4.03%	ps-scanner-2	[kernel.kallsyms]	[k] syscall_exit_to_user_mo
3.13%	ps-scanner-1	[kernel.kallsyms]	[k] syscall_return_via_sysr	1.97%	ps-scanner-2	[kernel.kallsyms]	[k] __entry_text_start
3.10%	ps-scanner-1	[kernel.kallsyms]	[k] __entry_text_start	1.87%	ps-scanner-2	[kernel.kallsyms]	[k] psi_group_change
2.54%	ps-scanner-1	[kernel.kallsyms]	[k] __fget_light	1.72%	ps-scanner-2	[kernel.kallsyms]	[k] syscall_return_via_sysr
2.28%	ps-scanner-1	[kernel.kallsyms]	[k] psi_group_change	1.66%	ps-scanner-2	[kernel.kallsyms]	[k] fsnotify
2.11%	ps-scanner-1	[kernel.kallsyms]	[k] fsnotify	1.63%	ps-scanner-2	[kernel.kallsyms]	[k] crc32c_pcl_intel_update
1.75%	ps-scanner-1	[kernel.kallsyms]	[k] copy_user_generic_unrol	1.37%	ps-scanner-2	libc-2.31.so	[.] __strchrnul_avx2
1.65%	ps-scanner-1	[kernel.kallsyms]	[k] __kmalloc	1.32%	ps-scanner-2	[kernel.kallsyms]	[k] ext4_do_update_inode
1.44%	ps-scanner-1	[kernel.kallsyms]	[k] get_obj_cgroup_from_cur	1.30%	ps-scanner-2	[kernel.kallsyms]	[k] __kmalloc
1.10%	ps-scanner-1	[kernel.kallsyms]	[k] pipe_read	1.29%	ps-scanner-2	[kernel.kallsyms]	[k] memcg_slab_free_hook
1.09%	ps-scanner-1	[kernel.kallsyms]	[k] __sched_text_start	1.23%	ps-scanner-2	[kernel.kallsyms]	[k] copy_user_generic_unrol
1.08%	ps-scanner-1	[kernel.kallsyms]	[k] common_file_perm	1.21%	ps-scanner-2	[kernel.kallsyms]	[k] __fget_light
1.03%	ps-scanner-1	[kernel.kallsyms]	[k] vfs_write	1.12%	ps-scanner-2	[kernel.kallsyms]	[k] kmem_cache_free
1.03%	ps-scanner-1	[kernel.kallsyms]	[k] mutex_lock	1.11%	ps-scanner-2	libc-2.31.so	[.] __IO_file_xspuon@GLIBC_
0.99%	ps-scanner-1	ps-scanner-1	[.] send_stats	1.05%	ps-scanner-2	libc-2.31.so	[.] __IO_fwrite
0.92%	ps-scanner-1	[kernel.kallsyms]	[k] crc32c_pcl_intel_update	1.01%	ps-scanner-2	libc-2.31.so	[.] __IO_default_xspuon
0.89%	ps-scanner-1	[kernel.kallsyms]	[k] do_mq_timedsend	0.99%	ps-scanner-2	[kernel.kallsyms]	[k] xas_load
0.87%	ps-scanner-1	[kernel.kallsyms]	[k] fput_many	0.99%	ps-scanner-2	libc-2.31.so	[.] __memmove_avx_unaligned
0.85%	ps-scanner-1	[kernel.kallsyms]	[k] memcg_slab_free_hook	0.94%	ps-scanner-2	libc-2.31.so	[.] __itoa_word
0.83%	ps-scanner-1	libc-2.31.so	[.] __GI___mq_timedsend	0.92%	ps-scanner-2	[kernel.kallsyms]	[k] page_counter_cancel
0.81%	ps-scanner-1	libc-2.31.so	[.] __IO_default_xspuon	0.91%	ps-scanner-2	[kernel.kallsyms]	[k] vfs_write
0.77%	ps-scanner-1	[kernel.kallsyms]	[k] page_counter_try_charge	0.90%	ps-scanner-2	[kernel.kallsyms]	[k] page_counter_try_charge
0.75%	ps-scanner-1	[kernel.kallsyms]	[k] switch_fpu_return	0.89%	ps-scanner-2	[kernel.kallsyms]	[k] __sched_text_start
0.73%	ps-scanner-1	[kernel.kallsyms]	[k] kmem_cache_free	0.89%	ps-scanner-2	[kernel.kallsyms]	[k] kmem_cache_alloc
0.73%	ps-scanner-1	[kernel.kallsyms]	[k] update_curr	0.88%	ps-scanner-2	libc-2.31.so	[.] __IO_fflush
0.69%	ps-scanner-1	[kernel.kallsyms]	[k] newidle_balance	0.84%	ps-scanner-2	[kernel.kallsyms]	[k] get_obj_cgroup_from_cur
0.69%	ps-scanner-1	[kernel.kallsyms]	[k] pipe_write	0.81%	ps-scanner-2	ps-scanner-2	[.] packet_information
0.68%	ps-scanner-1	[kernel.kallsyms]	[k] copy_user_enhanced_fast	0.81%	ps-scanner-2	[kernel.kallsyms]	[k] copy_user_enhanced_fast
0.68%	ps-scanner-1	[kernel.kallsyms]	[k] page_counter_cancel	0.73%	ps-scanner-2	[kernel.kallsyms]	[k] __find_get_block
0.67%	ps-scanner-1	libc-2.31.so	[.] __strchrnul_avx2	0.70%	ps-scanner-2	[kernel.kallsyms]	[k] ext4_inode_csum.isra.0
0.66%	ps-scanner-1	[kernel.kallsyms]	[k] kmem_cache_alloc	0.70%	ps-scanner-2	[kernel.kallsyms]	[k] packet_recvmg
0.65%	ps-scanner-1	[kernel.kallsyms]	[k] cpuacct_charge	0.69%	ps-scanner-2	[kernel.kallsyms]	[k] __mod_memcg_state
0.62%	ps-scanner-1	ps-scanner-1	[.] packet_information	0.68%	ps-scanner-2	[kernel.kallsyms]	[k] pagecache_get_page
0.62%	ps-scanner-1	libc-2.31.so	[.] __IO_fwrite	0.67%	ps-scanner-2	[kernel.kallsyms]	[k] __skb_wait_for_more_pac
0.61%	ps-scanner-1	libpthread-2.31.so	[.] __libc_read	0.62%	ps-scanner-2	[kernel.kallsyms]	[k] ext4_journal_get_writ

На данном рисунке продемонстрированы отчёты утилиты perf. Слева – отчёт утилиты ps-scanner-1, справа – отчёт утилиты ps-scanner-2.

Выводы

Из данных отчётов можно заметить, что эффективнее процессорное время расходует вторая реализация, которая сразу суммирует статистику. То есть `ps-scanner-2` работает лучше, чем `ps-scanner-1`. Также стоит отметить, что `ps-scanner-2` обрабатывает пакеты быстрее, чем `ps-scanner-1`.

5. Авторство и лицензия

Автор: Егор Бронников <bronnikov.40@mail.ru>

network-traffic лицензирован под GPL-3.0 License.

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each sourcefile to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

Utilities for sniffing packages by specified parameters.
Copyright (C) 2022 Egor Bronnikov

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <<https://www.gnu.org/licenses/>>.

Also add information on how to contact you by electronic and paper

mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
network-traffic Copyright (C) 2022 Egor Bronnikov
This program comes with ABSOLUTELY NO WARRANTY; for details type
  'show w'.
```

```
This is free software, and you are welcome to redistribute it
under certain conditions; type 'show c' for details.
```

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary.

For more information on this, and how to apply and follow the GNU GPL, see [<https://www.gnu.org/licenses/>](https://www.gnu.org/licenses/).

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read [<https://www.gnu.org/licenses/why-not-lgpl.html>](https://www.gnu.org/licenses/why-not-lgpl.html).

Приложение А

Coding Style

С

В сторонних проектах с собственным описанным стилем оформления кода следует придерживаться правил этого стиля:

- U-Boot: <https://www.denx.de/wiki/U-Boot/CodingStyle>
- Linux: <https://www.kernel.org/doc/html/v4.10/process/coding-style.html>

В сторонних проектах без описанного стиля следует оформлять наш код по аналогии с остальными исходниками проекта.

В наших собственных проектах за основу взят стиль, принятый в linux с некоторыми изменениями.

Отступы

В качестве отступа используется один таб шириной в четыре пробела.

```
int main(void)
{
    int a = 0;
    return a;
}
```

Лишние табы и пробелы в конце строк следует удалять.

Длина строк

Следует избегать превышения ограничения в 80 символов на строку. Если выражение не помещается в 80 символов, его следует разделить

на части. При этом желательно, чтобы каждый аргумент функции находился в отдельной строке. Пример:

```
return_code = my_structure->callback_with_arguments(argument_number_1,  
                                                    argument_number_2);
```

Строки, выводимые программой на экран, не следует сокращать при превышении ими ограничения.

Скобки

В условиях, циклах и объявлениях структур открывающая фигурная скобка не переносится на следующую строку. Закрывающая переносится:

```
while (a > b) {  
    a--;  
    b++;  
    if (a == b) {  
        do_something();  
    }  
}
```

Закрывающая фигурная скобка располагается на том же уровне отступов, что и начало всей конструкции.

В объявлениях функций открывающая фигурная скобка переносится на следующую строку:

```
int main(int argc, char **argv)  
{  
    return 0;  
}
```

В условиях и циклах перед открывающей круглой скобкой ставится пробел. После закрывающей скобки ставится пробел:

```
if (a == 0) {  
    return a;  
}
```

Фигурные скобки должны присутствовать даже если в блоке всего одно выражение.

Пробелы

Бинарные и тернарные операторы окружаются пробелами:

= + - < > * / % | & ^ <= >= == != ? :

После унарных операторов не ставится пробел:

& * + - ~ ! sizeof typeof alignof __attribute__ defined

Пробел не ставится после префиксных инкремента и декремента:

++i

--i

и перед постфиксными:

i++

i--

Пробелы не ставятся вокруг операторов структур:

my_struct.element

my_struct->element

При объявлении указателя оператор * идет перед именем указателя, а не после его типа:

void *buffer

Имена и объявления

В именовании функций, переменных и типов используется *snake_case*.

Примеры:

```
void my_function(int my_argument);  
struct my_struct *s;
```

В именах желательно стараться избегать сокращений, из-за которых становится не ясно назначение сущности. Например, вместо **dctl()** использовать более полное имя **device_control()**.

Объявлять переменные в функции желательно рядом с выражениями, их использующими, если функция достаточно длинная (от 10 строк), либо в начале функции, если короткая (до 10 строк). Например, переменную-счётчик для цикла `for` следует объявлять так:

```
for (int i = 0; i < 3; ++i) {  
    actions();  
}
```

Также нужно следить за количеством переменных в одной функции. Если их число превышает пять, то следует задуматься о декомпозиции функции.

Функции

Функции должны быть как можно короче и выполнять как можно меньше действий. Длинные функции следует разбивать на подфункции, действия внутри которых связаны по смыслу. Пример:

```
int collect_data_and_calculate_result(struct program_context *c)  
{  
    void *data = collect_data(c);  
  
    return calculate_result(data);  
}  
  
int init_and_run_program(struct arguments *a){  
    struct program_context *c = init_context(a);  
  
    return collect_data_and_calculate_result(c);  
}
```



```
}
```

Структуры перечисления

Их объявления должны выглядеть следующим образом.

Структуры только с именем:

```
struct my_struct_name {  
    int i;  
    int j;  
    void *buffer;  
};
```

Структуры с typedef должны иметь суффикс `_t` в имени типа:

```
typedef struct my_struct {  
    int i;  
    int j;  
    void *buffer;  
} my_struct_t;
```

Перечисление объявляются аналогичным образом. Все варианты перечисления должны именоваться в верхнем регистре:

```
typedef enum {  
    ADDRESS_7BIT,  
    ADDRESS_10BIT,  
} addressing_mode_t;
```

Макросы

Желательно избегать написания макросов. Особенно вложенных, так как это приводит к сложностям в отладке.

Макросы именуются с использованием только верхнего регистра. Макрос и отдельно его входные параметры должны быть окружены круглыми скобками:

```
#define MIN(x, y)      ((x) < (y) ? (x) : (y))
```

Макросы с несколькими выражениями должны быть заключены в блок `do {} while(0)`:

```
#define DO_ACTION(a, b) \
    do { \
        if (do_first(a) >= 0) { \
            do_next(b); \
        } \
    } while(0)
```

Использование goto

`goto` можно использовать для обработки ошибок. Переходя должны осуществляться только в пределах одной функции. Пример:

```
int configure_device(void)
{
    int ret = 0;
    struct my_device *d = malloc(sizeof(my_device));
    if (!d) {
        return -ENOMEM;
    }

    ret = init_stage_first(d);
    if (ret != 0) {
        goto err_free;
    }

    ret = init_stage_last(d);
    if (ret != 0) {
        goto err_free;
    }

    return 0;

err_deinit:
```

```

    deinit_stage_first(d);
err_free:
    free(d);
    return ret;
}

```

Комментарии

Комментарии следует использовать только в качестве документации и для пояснения каких-то не очевидных специфичных случаев.

Функции

Документировать следует функции, поведение, входные и выходные параметры которые не очевидны. Пример:

```

/*
 * Configures an I2C bus.
 *
 * addr_lenght should be ADDRESS_7BIT or ADDRESS_10BIT.
 * speed should be SPEED_100MHZ or SPEED_400MHZ.*
 * Returns 0 on success, -1 otherwise.
 */
int configure_i2c_bus(int bus_number, int addr_length, int speed)

```

Не следует документировать очевидные функции. Например, в достаточно очевидны назначение и принцип работы функции. Пример:

```

int sum_int(int a, int b)
{
    return a + b;
}

```

Магические числа

Если назначение числового значения не очевидно из имени переменной или макроса, которым оно присваивается, то это значение следу-

ет снабдить комментарием. Пример:

```
#define DEVICE_CONTROL_REGISTER_ADDR 0x4
/* Set required device mode on startup. See device datasheet page
   100, table 20. */
#define DONTROL_REGISTER_DATA 0x4f7a
```

Заголовочные файлы

Должны обязательно содержать защиту от повторного включения:

```
#ifndef MY_HEADER_H
#define MY_HEADER_H
All the contents of the header file here
#endif
```

либо

```
#pragma once
Content
```

Содержимое может включать:

- дополнительные `include`
- объявление макросов
- объявление констант
- прототипы функций

но реализации функций должны содержаться только в файлах `.c`

Python

C_M. <https://peps.python.org/pep-0008/>