# Утилиты:

1. evtest –монитор событий устройств ввода, то же что и /dev/input/event\*

(для кнопки вкл\выкл OrPi: event\*(axp20x-pek) ); Важно снимать блокировку при чтении функцией read(),для этого fd=open(“..”,O\_RDONLY|O\_NONBLOCK)

1. man [utility] –справка по “utility”
2. xmodmap(-pk)- маппер (GUI)

для кнопки вкл\выкл OrPi: keycode:124, name:XF86PowerOff

1. xev- отображение событий (GUI)
2. tty- в каком терминальном окне находимся
3. chvt N- переключиться на терминальное окно N или используй сочетание клавиш Ctrl+Alt+Fn(F1, F2 и т.п., F7-графический)
4. fgconsole- кол-во активных терминальных окон
5. deallocvt- отключить неиспользуемые терминальные окна
6. dmesg
7. less- постраничный вывод текста в терминал(выйти ‘q’, перемещаться ‘keyUP/keyDOWN’); так же см. head и tail. less –N вывод с нумерацией строк
8. lsof- узнать кто использует файл (например: lsof | grep /dev/ | less)
9. showkey
10. fdisk
11. cowsay
12. figlet
13. mount/umount
14. find
15. uname- информация о системе
16. jobs- отображение фоновых процессов
17. kill- завершить процесс
18. trap- перехват сигнала
19. top- постоянный вывод списка работающих процессов, ps- одиночный “snapshot”
20. cp, rm, mv, cd, pwd, date, ping, ls, tree
21. lsmod- список модулей(драйверов) загруженных в систему (только “динамические” т.е. не built in), rmmod- выгрузить модуль, modprobe- загрузка модулей ядра, insmod
22. type –a “команда” –путь к команде. Пути к исполняемым программам можно посмотреть так: echo “${PATH//:/$’\n’}”, command- краткая информация по команде
23. which- найти команду, whereis- расположение исходника команды, бинарного файла, мануала

# Сторонние библиотеки для работы с GPIO:

1. Libgpiod (gpioinfo и т.п.)
2. wiringOP (gpio readall и т.п)
3. acpid (acpi\_listen и т.п.)- через эту утилиту можно создать обработчик событий гугли: “acpi orangepi”

# Файловая система Linux

Про файловую систему гугли “FHS”  
/proc/- генерируется системой “на лету”, содержит информацию о процессах PID (папки с номерами) ядре, памяти и т.п. Sysfs можно посмотреть в /sys/…

# Общие заметки

ОСНОВНОЕ ПРАВИЛО LINUX - SOURCE CODE ТВОЯ ДОКУМЕНТАЦИЯ!!!! А-А-А-А-А-А-А-А

cat /proc/bus/input/devices- узнать процессы input  
Получить карту памяти устройства(CPU): cat /proc/iome. Номера аппаратных устройств типа 7081400.i2c к которым обращаются драйвера, файловые дескрипторы и т.п соответствуют начальному адресу в памяти.

## Терминология

SPL- Secondary Program Loader

# GPIO

cat /sys/kernel/debug/gpio- узнать системные GPIO. Более детально, с указанием функций и драйверов для каждого вывода: cat /sys/kernel/debug/pinctrl/\*/pinmux-pins  
Принцип расчета номера GPIO: (position of letter in alphabet - 1) \* 32 + pin number, например PH18=(8-1)\*32+18=242. Это же считается как offset для gpiochip\*.

# Кнопка SW4 OrangePi 3 LTS

*Про КНОПКУ(SW4) на OrangePi 3 LTS:* я предполагаю, что кнопка подключена к микросхеме управления питанием (axp805, вывод EN/PWRON), эта микросхема “общается” с процессором по последовательному протоколу (вероятно по i2c), выводами: SKC(AXP)<--(PMU-SCK)-->PL0(CPU) и SDA(AXP)<--(PMU-SDA)-->PL1(CPU). Эта микросхема(регулятор) достаточно сложная и имеет много режимов работы, сильно углубляться не стал, однако, в linux-драйвере (см. далее) она сконфигурирована как Self-work-mode.   
Устройство в ОС, формирующее событие (см. evtest) по нажатию кнопки- i2c(axp20x-pek): …/platform/soc/7081400.i2c, драйвер: mv64xxx\_i2c --> /i2c-2/0036: 0036 -что-то вроде адреса устройства на шине (его драйвер axp20x-i2c) в нем два под устройства:

1. --> axp20x-regulator (его драйвер: axp20x-regulator)
2. -->axp221-pek (его драйвер: axp20x-pek)

Сами драйвера можно найти в “linux-kernel”/drivers/…; bindings для devicetree в “linux-kernel”/Documentation/devicetree/bindings/mfd/... Как получить структуру дерева устройств читай ниже. В ноде devicetree (i2c@7081400-->pmic@36-->regulators) есть прототип “sw{};” для работы с кнопкой, теоретически его можно заполнить через overlay.

; пины PL0, PL1 (gpiochip0): …/platform/soc/7022000.pinctrl, драйвер: sun50i-h6-r-pinctrl.   
Событие при нажатии на кнопку определяется как EV\_SYN и сопутствующее ему EV\_KEY (гугли input/event-codes), которое имеет код KEY\_POWER (см. …/linux/input.h). Это событие можно обработать по типу нажатия на клавиатуру (читай про ioctl ниже). Про отключение кнопки в системе читай ниже.

# Светодиоды на OrangePi 3 LTS

*Про светодиоды* *на OrangePi 3 LTS:* PWR\_LED(красный)- PL4, STATUS\_LED(зеленый)- PL7. Устройство в linux: platform/leds. Драйвер: “leds-gpio”.  
; пины PL4, PL7 (gpiochip0): …/platform/soc/7022000.pinctrl, драйвер: sun50i-h6-r-pinctrl.  
Внимание!!! Здесь я указал PWR\_LED(красный)- PL4, STATUS\_LED(зеленый)- PL7 согласно принципиальной схеме, однако, в сборке Linux(5.10.75-sun50iw6) через GPIO\_GET\_LINEINFO\_IOCTL(gpiochip0) они указаны наоборот- и это неправильно.  
Как я указал, светодиоды находятся под контролем драйвера “leds-gpio”(built in), относятся к классу leds (/sys/class/leds/…) => должны управляться из предоставленного API и при попытке обращения к ним напрямую через:   
echo 356 > /sys/class/gpio/export  
или через GPIO\_GET\_LINEHANDLE\_IOCTL  
Будет выводиться ошибка по типу: Device or resource busy.   
Так как модуль “leds-gpio” встроен в ядро отключить его не получиться, как вариант можно использовать overlay для dts и переопределить устройство как gpio, надо проверить.

# Отключение кнопки питания

Отключить кнопку вкл\выкл можно через:

/etc/system/logind.conf, присвоив HandlePowerKey=ignore; PowerKeyIgnoreInhibited=yes;

затем перезапустив:

systemctl restart system-logind

или перезагрузив систему.

Однако, где-то все равно остается механизм выключения, запускающий shutdown

**Верный способ** отключить кнопку- выключить сервис system-logind.service:

systemctl mask (--runtime, для этой сессии?) systemd-logind.service

pam-auth-update (не проверял)

После перезагрузки кнопка не будет выключать систему (как минимум, ОС не будет реагировать на event\* axp20..., процесс system-l отсутствует)

Важно!! При этом способе невозможно запустить дополнительное терминальное окно

Для включения:

systemctl unmask systemd-logind.service

systemctl restart systemd-logind.service

Кроме того, можно попробовать отлавливать сигнал по типу SIGPWR (через trap) и его модифицировать (например, отменять выключение: shutdown –c. Возможный конфликт с system-logind), не проверял.

# Установка пакетов

Установить что-либо- apt install “имя пакета”;  
удалить- apt remove(autoremove) “имя пакета”, после удаления пакета лучше перезагрузиться;  
dpkg –l –список установленных пакетов;

# IOCTL

Я так понял, что все управление устройствами ввода\вывода происходит через утилиту ioctl… с ней необходимо разобраться!

Макросы и структуры для работы с GPIO см. в <…/linux/gpio.h>, например псевдокод:  
struct gpiochip\_info info;

int fd = open(“/dev/gpiochip0”, O\_RDONLY);

ioctl(fd, GPIO\_GET\_CHIPINFO\_IOCTL, &info);

положит в структуру info данные о “/dev/gpiochip0”

Макросы и структуры для работы с устройствами ввода\вывода см. в <…/linux/input.h>..больше информации [тут](https://www.linuxjournal.com/article/6429).

Прочитать состояние клавиш (всю карту) можно при помощи директивы EVIOCGKEY.

Либо через чтение события (в /dev/input/event\*), псевдокод:

struct input\_event ie;

int fd = open(“/dev/input/event3”, O\_RDONLY);

tmp=read(fd, &ie, sizeof(struct input\_event)), однако, функция чтения события- блокирующая(контрится через таймаут функции select(……,timeout))

# Power Management

Управление питанием и т.д. происходит через сервис system-logind с ним необходимо разобраться, и почитать про pam\_systemd!

# Devicetree

Структуру дерева устройств можно посмотреть в /sys/firmware/devicetree/base, “распаковать” ее можно так: (из /sys/firmware/devicetree) dtc –I fs base/. Исходный файл дерева устройств можно получить так: (из /boot/dtb/allwinner) dtc –O dts ‘название платы’, сохранить его можно с помощью ключа –o ‘название файла’. Однако, при этом подходе все ссылочные секции и символьные метки (по типу &pio) будут представлены в шестнадцатеричном виде (крч, лучше исходники искать в репозиториях производителей или типо того).

В более старых версиях ядра (not mainline) для конфигурации устройств использовался файл FEX/script.bin

Каждый узел (node) дерева устройств имеет набор пар: property name – property value, часть property являются стандартными (compatible, #address-cells, например), а часть определяется драйвером устройства (список этих характеристик и их возможные значения содержаться в “Linux\Documentation\devicetree\bindings”).

Больше информации про Devicetree см. в *devicetree-specification-v0.4.pdf*

Заметки:

1) Свойство status (определяющее рабочее состояние устройства) будучи не определено, т.е. отсутствуя, обрабатывается так, как если бы свойство существовало со значением “okay”.

2) Узел \_\_symbols\_\_ необходим для метода target=<phandle>.

Возможно несколько вариантов внесения изменений в DT, один из них- использовать overlay. ВАЖНО!- Как пример, про overlay для armbian, можно почитать [тут](https://docs.armbian.com/User-Guide_Armbian_overlays/) . Статьи на тему [1](https://devdotnet.org/post/rabota-s-gpio-na-primere-banana-pi-bpi-m64-chast-2-device-tree-overlays/); [2](https://irq5.io/2018/07/24/boot-time-device-tree-overlays-with-u-boot/). Загрузчику необходимо сообщить про overlay, для U-boot читай [тут](https://u-boot.readthedocs.io/en/latest/usage/fdt_overlays.html). Кроме того, существует еще одна абстракция (как я понял над overlay)- …\_fixup.scr, это некий скрипт, взаимодействующий с u-boot, переменными окружения и командами “fdt”, заменяющий функциональность аргументов overlay. Подробнее читай в [заголовке](#_Das_U-boot)

# Демон Linux

Демон- компьютерная программа. Запускается самой системой и работает в фоновом режиме, без взаимодействия с пользователем и привязки к терминалу.

Как вариант, можно создать так:  
nohup ./myScript & (nohup- отвязывает ссылки процесса на STDOUT и STDERR, тем самым, процесс не зависит от терминала пользователя; &- делает процесс фоновым)

# Драйвера(модули)

Модули ядра (драйвера) могут быть встроенными (Built-in kernel modules), т.е. являются частью ядра и интегрируются в него на этапе компиляции. Либо встраиваемыми (LKM- loadable kernel module), т.е. могут загружаться\выгружаться пользователем.

Заблокировать загрузку LKM- модуля системой можно добавив его в /etc/modprobe.d/, далее указываем наименование модуля в новой строке файла blacklist.conf. Либо создаем свой файл .conf  
Просмотреть загруженные модули можно с помощью lsmod, либо cat /lib/modules/$(uname -r)/modules.builtin (выведет все built-in модули для текущего ядра)  
Про модули(драйвера) их разработку и т.п можно почитать [тут](https://embetronicx.com/linux-device-driver-tutorials/)

# Log- файлы

Про лог-файлы см. [habr](https://habr.com/ru/post/332502/)

# Das U-boot

[База](https://u-boot.readthedocs.io/en/latest/index.html). В CLI u-boot’a во время загрузки можно войти через serial- console (ttyS0, например. Это UART) путем нажатия … Ctrl-C, например. Туда же выводиться лог (echo) скрипта u-boot. Shell- команды типа “boot” завершат сессию CLI и продолжат загрузку. Имеется вариант конфигурирования сценария загрузки u-boot’a путем создания скриптов подробнее- гугли. В качестве примера (годно!) изучай boot.cmd или boot.scr (в них внизу есть команда сборки через mkimage), а так же orangepiEnv.txt из Ubuntu- образа (/boot/…). Для останова во время скрипта используй команду “pause” (не проверял, поддерживается не во всех сборках). Так же, в качестве документации можно использовать source code проекта (каталог u-boot/doc/…). Про fixup, например, можно почитать в …/doc/develop/driver-model/fdt-fixup.rst. ВАЖНО!- что бы понять алгоритм загрузки изучай printenv, там прописаны все скрипты. Запуск скрипта в CLI через команду run. Запись **‘${name}’** означает значение переменной с именем **name**. т.е.:  
setenv test 12345  
setenv tmp ${test}  
printenv tmp  
tmp=12345  
Про команды загрузчика читай [тут](http://nskhuman.ru/allwinner/uboot/dasuboot.php)  
Про fdt. U-boot по умолчанию пытается работать (fdt print, fdt set и т.п.) с “fdtcontorladdr” это может быть предопределенное/базовое/любое дерево устройств, которое не участвует в загрузке. Соответственно нету смысла с ним взаимодействовать. Вероятная последовательность действий (действует до перезагрузки):   
1) Устанавливаем адрес дерева устройств (если не определен, уточни через printenv)   
setenv fdt\_addr\_r <0x00000000, очевидно не этот..>  
2) Загружаем в этот адрес локальное дерево устройств   
load ${devtype} ${devnum} ${fdt\_addr\_r} <path, например /boot/dtb/fdt\_name.dtb>  
3) Указываем контрольное дерево устройств  
fdt addr ${fdt\_addr\_r}  
4) Далее можно с ним работать…  
Environment variables и возможно что то еще, u-boot выгружает в RAM, соответственно после редактирования их нужно записать в ROM для этого используется команда saveenv.  
Необходимо разобраться с config- файлом u-boot.  
Про config- файл. Как я понял, u-boot обращается к файлу конфигурации по типу <board>\_defconfig. В сборке для orangePi, которую я использовал, он находился в директории /usr/lib/u-boot/… Этот файл нету смысла менять, так как он является вроде как справочным. Конфигурацию можно поменять только перекомпилировав исходник((. Вероятно, скомпилированный загрузчик можно подгрузить на место старого с помощью команды ‘dd’, не проверял.