

КОРИШТЕЊЕ ДЕБАГЕРА

Обавезно крос-компајлирати са укљученим дебаг информацијама (опција `-g3`)

- `$arm-linux-gnueabihf-gcc ./src/conway.c -g3 -o ./bin/conway`

Уобичајено је да се на циљну платформу не пребацују непотребне дебаг информације, па се оне прије пребацивања на циљну платформу избацују.

- `arm-linux-gnueabihf-strip ./bin/conway -s -o ./bin/conway4target`

Пребацити фајл на РПИ

- `scp -P 5022 ./bin/conway4target pi@ip_address:~`

Инсталирати дебагер сервер на РПИ, ако већ није инстлиран (ово највјероватније неће радити ако је РПИ накачен на домаћина директно каблом):

- `sudo apt-get install gdbserver`

Покренути сервер (на РПИ, наравно)

- `gdbserver :8000 ./conway4target`

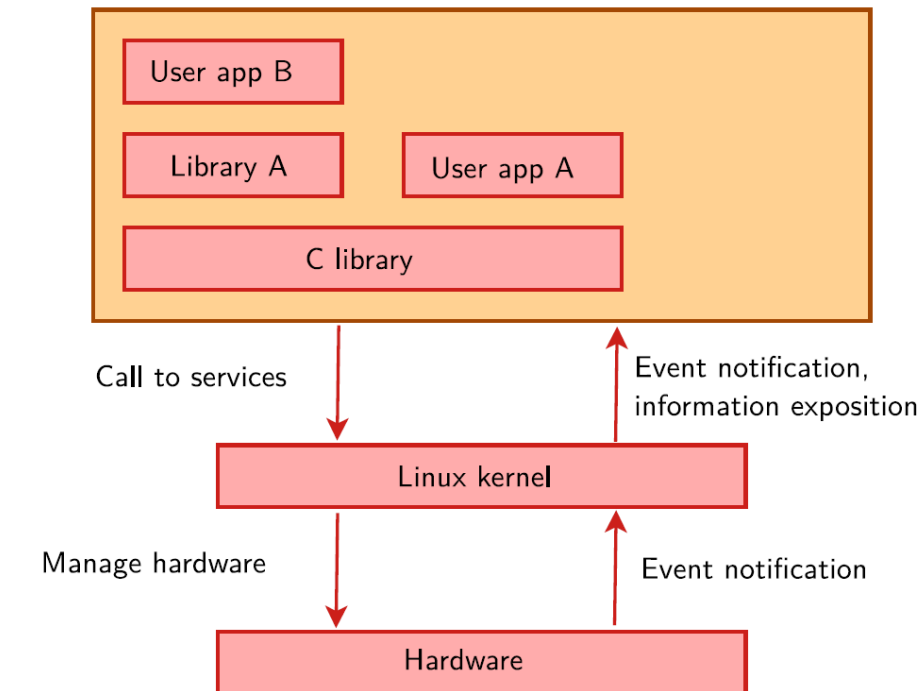
На домаћину (значи, не на РПИ) покренути дебагер

- `arm-linux-gnueabihf-gdb -q ./Conway`
- `(gdb) target remote rpi_address:8000`

Остале команде су идентичне са наредбама које су се већ користиле, једина нова наредба је „силазак“ са сервера

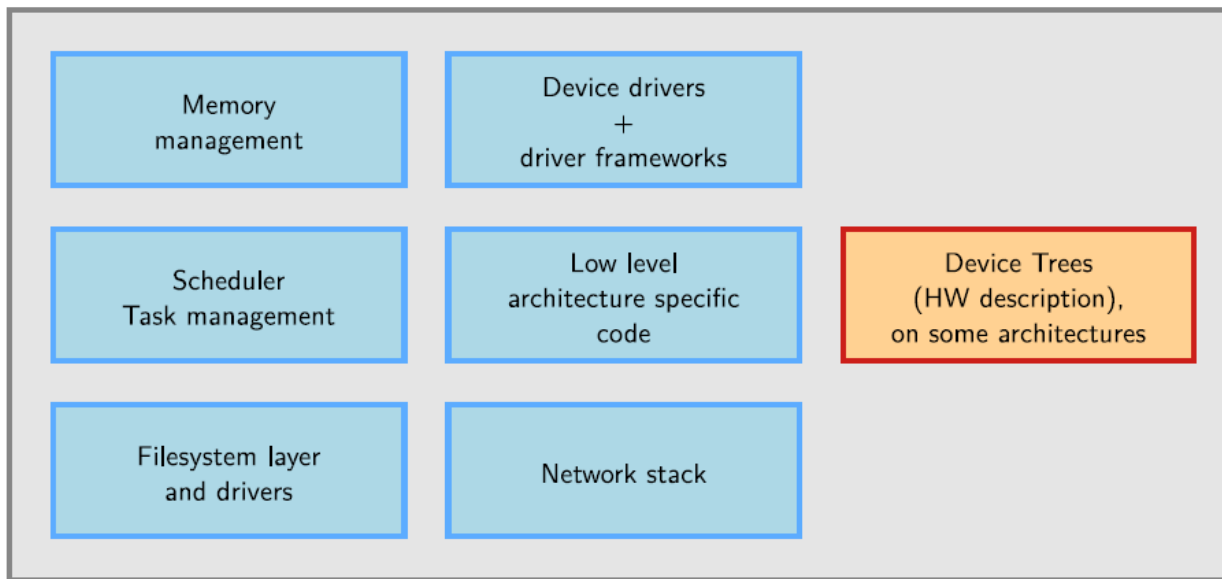
- `(gdb) monitor exit`


МІЄСТО LINUX KERNEL-А У СИСТЕМІ




ИЗНУТРА

Linux Kernel



 Implemented mainly in C, a little bit of assembly.

 Written in a Device Tree specific language.

КРОС КОМПАЈЛИРАЊЕ ОПЕРАТИВНОГ СИСТЕМА

Измјенити фајл `.bashrc` тако што ће се на крај фајла додати линија:

- `export KERNEL=kernel7`
- `. ~/.bashrc` (извршити скрипт)

Инсталирати потребне алате (инсталирати и гит, ако није инсталиран на некој од прошлих вјежби):

- `sudo apt-get install bison flex libssl-dev`

Спустити изворни код:

- `git clone --depth=1 https://github.com/raspberrypi/linux <mozda_drugi_folder>`
- `git clone --depth=1 --branch rpi-5.6.y https://github.com/raspberrypi/linux`

Најбољи начин претраге изворног кода:

- <https://elixir.bootlin.com>

ОГРАНИЧЕЊА ЛИНУКСОВОГ КЕРНЕЛА

Заборавити све из Це библиотеке

Кернел има своје функције и библиотеке (као што смо видјели)

Не постоји *floating point* у кернелу.

Интерни API се може промијенити између два *release*-а

- `Documentation/process/stable_api_nonsense.rst`
- Екстерни API је стабилан (како би радили програми који су и прије радили)

Не постоја никаква заштита меморије

- Оно што је у корисничком простору *segmentation fault* овдје, у кернелу, убија оперативни систем

Стек сегмент константне величине (8 KB или 4 KB). Амин

КОНФИГУРАЦИЈА

Кернел садржи на хиљаде руковаоца фајл системима, уређајима, мрежним протоколима, а осим тога посједује могућност конфигурисања многих параметара.

Хиљаде опција су у игри и од њих зависи на који начин ће бити компајлиран који дио изворног кода. Ово је наука само за себе!

Оно што је очигледно јесте да конфигурација зависи од циљане архитектуре и способности које се дају кернелу (нпр. Нећемо мрежне уређаје укључивати уопште)

Уобичајена конфигурација се налази у основном директоријуму, под називом `.config` (може бити да га неће бити прије позива прве конфигурације)

Стварна конфигурација која нама треба, налази се у фолдеру

- `/arch/<arhitektura>/configs`

Конфигурација се најлакше прегледа и мијења помоћу команде (један од алата)

- `make xconfig`

Али је за покретање потребно инсталирати:

- `sudo apt-get install qt5-default g++ pkg-config`

Свако снимање промјена, снима промјене у `.config`, али и бекапује стари фајл у `.config.old`

Команда којом се архитектурна конфигурација пребацује у основни директоријум је

- `make bcm2709_defconfig`

НАПОКОН, КОМПАЈЛИРАЊЕ!

Обавезно помоћу `-j 4` (чиме користимо 4 паралелна процеса за компајлирање) који се покреће из основног фолдера (јер смо прије тога намјестили конфигурацију и снимили је у `.config` фајл)

- `make -j4 zImage modules dtbs`

Ово ће потрајати толико да ће љетњи дан изгледати кратак као трептај ока!

Оно што ће бити генерисано овим процесом је сљедеће:

- `vmlinux` – сиров, некомпресован кернел имиџ у ELF формату. Не може се бутати, али може послужити за дебаговање
- `xyzImage` – компресован имиџ који се служи за бутовање, и који се налази у `arch/<arhitektura>/boot`
- `*.dtb` – избилдани `*.dts` фајлови у `arch/<arhitektura>/boot/dts`. Детаљније о овоме у *Device Tree for Dummies* линку на Пијаци.
- `*.ko` – избилдани модули, просути по фолдерима

МИСТЕР ПРОПЕР

Брисање свих генерисаних фајлова, али остајеш на истој архитектури

- `make clean`

Брисање свих генерисаних фајлова укључујући и `.config` како би се прешло на компајлирање за другу архитектуру

- `make mrproper`

Брисање за поновно печовање

- `make distclean`

Брисање свих промјена на изворном коду

- `git clean -fdx`

ЗАДАТАК

Направити conway игру живота на такав начин да свака тачка/ћелија је представљена програмском нити. Еволуирање може да почне ако:

- Систем је исписао старо стање свих ћелија
- Нови циклус еволуције завршен код ћелија које имају већи приоритет (ћелије лијево и изнад имају већи приоритет од ћелија испод и десно)

Еволуција се одвија по правилима игре, у односу на старо стање сусједних ћелија (уколико су ћелије са лијеве или горње стране, пошто су већ завршиле еволуциони циклус) или тренутно стање сусједних ћелија испод и десно.

Када све ћелије заврше један еволуциони циклус, стање се исписује на обрисан екран, а систем се успављује на одређено вријеме.

Величина табеле је 10 x 10 ћелија, а трајање спавања главне нити (система) се уноси као улазни параметар.

Као примјер се могу користити кодови

- https://rosettacode.org/wiki/Conway%27s_Game_of_Life#C
- <https://github.com/StephanieKeck/conway-game-of-life>
- http://www.science.smith.edu/dftwiki/index.php/Game_of_Life_in_C_and_MPI
- Или било који други...

Провјерити да ли семафори и/или произвођач-потрошач концепт са претходног курса могу бити од помоћи