



**MATEMATICKO-FYZIKÁLNÍ  
FAKULTA**  
Univerzita Karlova

## **BAKALÁŘSKÁ PRÁCE**

David Jaromír Šebánek

# **Vizualizace trasování procesů v Linuxu**

Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: RNDr. Jan Kára, Ph.D.

Studijní program: Informatika (B0613A140006)

Praha 2025

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne .....

Podpis autora

**Poděkování.** iLoveImg upscaling, VS Code, ChatGPT, Qt dokumentace, hwloc dokumentace, Linux kernel dokumentace, WSL, Yordan K., J. Kára, atd.

Název práce: Vizualizace trasování procesů v Linuxu

Autor: David Jaromír Šebánek

Katedra: Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: RNDr. Jan Kára, Ph.D., Katedra distribuovaných a spolehlivých systémů

Abstrakt: **Abstrakt.**

Klíčová slova: Linux, trasování

Title: Visualization of tracing of processes in Linux

Author: David Jaromír Šebánek

Department: Department of Distributed and Dependable Systems

Supervisor: RNDr. Jan Kára, Ph.D., Department of Distributed and Dependable Systems

Abstract: **Abstract.**

Keywords: Linux, tracing

# Obsah

<b>Úvod</b>	<b>8</b>
<b>1 Trasování v Linuxu</b>	<b>10</b>
1.1 Základy v kernelu . . . . .	10
1.2 Nástroje pro sběr trasovacích dat . . . . .	10
<b>2 Vizualizace trasovacích dat</b>	<b>11</b>
2.1 Grafy a jejich obsahy . . . . .	11
2.2 Nástroje k vizualizaci dat . . . . .	11
<b>3 KernelShark</b>	<b>12</b>
3.1 Historie . . . . .	12
3.2 Data k vizualizaci . . . . .	12
3.3 GUI . . . . .	12
3.4 Architektura/moduly/dělení . . . . .	12
3.5 Limity/nedostatky . . . . .	12
<b>4 Obecná analýza a stanovení požadavků</b>	<b>13</b>
4.1 Pluginy a modifikace . . . . .	13
4.1.1 Označení modifikací v kódu . . . . .	14
4.2 Výběr vylepšení . . . . .	14
4.2.1 Lepší analýza zásobníku kernelu . . . . .	14
4.2.2 Dělení vlastnictví událostí souvisejících se dvěma procesy .	16
4.2.3 Vizualizace nečinnosti procesů . . . . .	18
4.2.4 Vizualizace NUMA topologie CPU vedle grafu . . . . .	19
4.2.5 Dodatečná vylepšení . . . . .	20
<b>5 Record Kstack</b>	<b>23</b>
5.1 Cíl . . . . .	23
5.2 Analýza . . . . .	23
5.2.1 Návrh . . . . .	23
5.2.2 Implementace . . . . .	23
5.3 Vývojová dokumentace . . . . .	23
5.4 Uživatelská dokumentace . . . . .	24
5.4.1 Uživatel GUI . . . . .	24
5.5 Rozšíření . . . . .	24
5.6 Zhodnocení splněných požadavků . . . . .	24
<b>6 Stacklook</b>	<b>26</b>
6.1 Cíle . . . . .	26
6.2 Analýza . . . . .	27
6.3 Vývojová dokumentace . . . . .	29
6.3.1 Modifikované soubory . . . . .	29
6.3.2 Struktura pluginu . . . . .	29
6.4 Uživatelská dokumentace . . . . .	29
6.4.1 Instalace . . . . .	34

6.4.2	Uživatel GUI . . . . .	34
6.4.3	Vývojář pluginů . . . . .	34
6.5	Rozšíření . . . . .	34
6.6	Zhodnocení splněných požadavků . . . . .	34
<b>7</b>	<b>Couplebreak</b>	<b>35</b>
7.1	Cíle . . . . .	35
7.2	Analýza . . . . .	35
7.3	Vývojová dokumentace . . . . .	36
7.3.1	Modifikované soubory . . . . .	36
7.3.2	Struktura modifikace . . . . .	36
7.4	Uživatelská dokumentace . . . . .	37
7.4.1	Uživatel GUI . . . . .	37
7.4.2	Vývojář pluginů . . . . .	37
7.4.3	Vývojář KernelSharku . . . . .	37
7.5	Rozšíření . . . . .	37
7.6	Zhodnocení splněných požadavků . . . . .	37
<b>8</b>	<b>Naps</b>	<b>38</b>
8.1	Cíle . . . . .	38
8.2	Analýza . . . . .	38
8.3	Vývojová dokumentace . . . . .	38
8.3.1	Modifikované soubory . . . . .	38
8.3.2	Struktura pluginu . . . . .	38
8.4	Uživatelská dokumentace . . . . .	38
8.4.1	Instalace . . . . .	38
8.4.2	Uživatel GUI . . . . .	38
8.4.3	Vývojář pluginů . . . . .	38
8.5	Rozšíření . . . . .	38
8.6	Zhodnocení splněných požadavků . . . . .	38
<b>9</b>	<b>NUMA Topology Views</b>	<b>39</b>
9.1	Cíle . . . . .	39
9.2	Analýza . . . . .	39
9.2.1	Terminologie . . . . .	39
9.3	Vývojová dokumentace . . . . .	39
9.3.1	Modifikované soubory . . . . .	39
9.3.2	Struktura modifikace . . . . .	39
9.4	Uživatelská dokumentace . . . . .	39
9.4.1	Nové závislosti KernelSharku . . . . .	39
9.4.2	Uživatel GUI . . . . .	39
9.4.3	Vývojář pluginů . . . . .	39
9.4.4	Vývojář KernelSharku . . . . .	39
9.5	Rozšíření . . . . .	39
9.6	Zhodnocení splněných požadavků . . . . .	39

<b>10</b>	<b>Dodatečná vylepšení</b>	<b>40</b>
10.1	Aktualizace kódu zaškrťovacích políček . . . . .	40
10.2	Zpřístupnění barev užívaných v grafu . . . . .	40
10.3	Reakce objektů v grafu na přejetí myší . . . . .	41
10.4	Měnitelné nápisy v hlavičce grafu . . . . .	41
10.5	NoBoxes . . . . .	42
	<b>Závěr</b>	<b>44</b>
10.6	Shrnutí práce . . . . .	44
	<b>Seznam obrázků</b>	<b>45</b>
	<b>Seznam tabulek</b>	<b>46</b>
	<b>Seznam použitých zkratk</b>	<b>47</b>
<b>A</b>	<b>Přílohy</b>	<b>48</b>
A.1	První příloha . . . . .	48

# Úvod

Na světě je mnoho počítačů, mnoho operačních systémů a mnohem více programů pro tyto systémy. Moderní systémy hojně využívají přepínání mezi programy k zefektivnění využívání existujících zdrojů, snaží se tak být co nejeftivnější. Mnoho větších systémů je dnes distribuovaných, s mnoha procesory a technologiemi, které se snaží využít tyto výpočetní jednotky v maximální možné míře.

Ovšem systém není vždy schopen sám zvýšit výkon. V tom případě je nutné začít zkoumat, kde se práce zdržuje, na co se nejvíce čeká a proč se na to čeká. Nalezený problém se pak může hlouběji zanalyzovat a výkon tak s vyřešeným problémem navýšit.

Na hledání problémů existuje mnoho metod, nicméně tato práce se bude zabývat pouze jednou z nich - trasování systému. Trasování systému je ve zkratce úkon, při kterém se na systému spustí nějaká práce společně s programem, který zaznamenává, co přesně systém během práce dělá. Program pak zaznamenaná data uloží. Tato data jsou ovšem často zakódována tak, aby se šetřilo místem - uživatel z těchto dat mnohem více spíš nic nevyčte než naopak.

Na záchranu přicházejí interpreti těchto dat, často s grafickým prostředím, jejichž cílem je vizualizace trasování. Vizualizace pak představí data uživateli v takovém formátu, že v nich již lze hledat místa, kde výkon systému nebyl dostatečně vysoký. Jedním z takových vizualizačních nástrojů je program KernelShark pro Linux. Avšak žádný program není dokonalý, KernelShark nevyjímaje.

KernelShark dokáže efektivně zobrazit rozhodnutí systémového plánovače úloh, na jakém CPU proces pracoval, než šel spát, na jakém CPU práci obnoví, jak dlouho období nečinnosti trvá a podobně. KernelShark ale neumožňuje snadno získat důvod usnutí procesu, na kterou událost nebo proces čeká. K tomuto je nutné využít další nástroje. Analýza problémů pak musí spoléhat na dalších několik nástrojů pro získání celé představy o systému a událostech v něm, což je v praxi obtížné, často až nemožné.

## Cíle práce

Cíle práce jsou primárně dva: hlouběji představit trasování, jeho vizualizaci a KernelShark, než jak je pouze nastiňuje úvod, a vylepšit KernelShark tak, aby analýza trasovacích dat byla informativnější a uživatelsky příjemnější.

## Struktura práce

Kapitoly práce lze rozdělit na dvě části. První část je teoretická a její součástí jsou kapitoly jedna až tři. V nich se hlouběji popisuje trasování v Linuxu, nástroje pro sběr a vizualizaci trasovacích dat a speciálně věnuje jednu kapitolu KernelSharku. Druhá část je zaměřená na vylepšení KernelSharku a pokrývá kapitoly čtyři až deset. Zde se analyzují požadavky na vylepšení, vytvářejí se technická rozhodnutí pro implementaci, součástí jsou i vývojové a uživatelské dokumentace,



spolu s rozšířeními pro každé vylepšení, příklady využití a zhodnocení splnění podmínek. Každé vylepšení má vlastní kapitolu, dodatečná vylepšení jsou seskupena v jedné kapitole a jejich popisy jsou stručnější. Závěr práce shrnuje.

# 1 Trasování v Linuxu

Cíle/úvod této kapitoly...

## 1.1 Základy v kernelu

## 1.2 Nástroje pro sběr trasovacích dat

## 2 Vizualizace trasovacích dat

Cíle/úvod této kapitoly...

### 2.1 Grafy a jejich obsahy

### 2.2 Nástroje k vizualizaci dat

## 3 KernelShark

Cíle/úvod této kapitoly...

### 3.1 Historie

Steven Rostedt, Yordan & open-source, ...

### 3.2 Data k vizualizaci

trace-cmd, vlastní sběr s trace-cmd  
perf, ftrace

### 3.3 GUI

Qt6, desktopová aplikace  
co to tam vlastně vidím?

### 3.4 Architektura/moduly/dělení

Jako subsekce: sessions, main window, libkshark, trace graph...

### 3.5 Limity/nedostatky

ignorujeme typos, ale ne deprecated warnings (UPDATE CBOX STATES)  
Ne-topologie - ale proč by ne? - SRP vs využití  
Nehezke stacky - jde to lépe - ftrace cool, ale vizualizace nerada (NoBoxes)  
Plugin depenency hell - sched\_events vs naps, modifikace dat & couplebreak  
k záchraně

## 4 Obecná analýza a stanovení požadavků

V této kapitole se podíváme na možná vylepšení schopností KernelSharku. Vytvoříme pro ně požadavky a sepíšeme jejich cíle a nutné vlastnosti. Dále obecně analyzujeme, jak se dají vylepšení vytvořit. Hlubší, technická analýza každého z vylepšení pak bude součástí detailnějších pohledů na návrh a implementace vylepšení v jejich samostatných kapitolách.

### 4.1 Pluginy a modifikace

První možnou cestou rozšíření KernelSharku je přidání pluginů, které upraví zobrazované informace, nebo nějaké přidají. KernelShark již obsahuje oficiální pluginy a je možné se jejich strukturou inspirovat. Pluginy dokážou vykreslovat uživatelem definované tvary (s nimiž lze interagovat) do grafu, pozměňovat data záznamů událostí a přidávat nová menu tlačítka do hlavního okna. Pluginy tak zřejmě nejsou schopné všeho. Například nelze skrze pluginy vytvářet další záznamy, které by se pak zobrazily v grafu.

Abychom mohli docílit všech vylepšení, tak vylepšení nemohou být jen pluginy, ale i modifikace kódu KernelSharku. Přímo u zdroje je pak možné manipulovat s celým programem, nicméně na oplátku bude potřeba nerozbít to, co již funguje, zajistit podobný styl s předchozím kódem a vyznačovat místa změn, aby byla respektována licence. Obecně lze od modifikací požadovat následující:

- *Minimální vliv*, tj. vypnutá modifikace musí mít buď žádný, nebo minimální vliv na chod KernelSharku a jeho oficiálních pluginů. Pokud takový vliv má, musí být navržena tak, že se lze dostat ke starému chování.
- *Stylová podobnost*, tj. kód modifikací by měl být podobný ostatnímu kódu v KernelSharku pro zachování jednotného stylu.
- *Chovat se jako rozšíření*, tj. kód modifikací by se také měl snažit měnit existující kód co nejméně, chovat se jako rozšíření co možná nejvíce. Změny existujícího kódu musí být označeny a pokud nejsou triviální, popsány.

Od pluginů se bude obecně očekávat tento seznam:

- *Vlastní adresář*, tj. pluginy budou mít vždy vlastní adresář s vlastními instrukcemi pro sestavení, kódem a dokumentací, vše mimo adresář s KernelSharkem, jeho modifikacemi a oficiálními pluginy. Struktura repozitáře tímto požadavkem prospěje, pluginy budou lépe navigovatelné a kontrola nad sestavením a dokumentací pevnější. Samozřejmě je pak nutné napsat vlastní CMake instrukce k sestavení, ale to je přijatelná práce navíc.
- *Samostatnost*, tj. pluginy by se měly snažit být co nejsamostatnější, tj. nebyť závislé na ostatních pluginech, ať už vztahem „plugin A potřebuje k fungování plugin B“ nebo vztahem „plugin A zakazuje plugin B“. Zejména druhý ze vztahů by mohl být nepříjemný, jelikož by mohl snižovat efektivitu

analýzy, některé pluginy by prostě nemohly být aktivní. První ze vztahů je mírnější, existuje hlavně pro snížení potenciálního chaosu závislostí (tzv. „dependency hell“). Žádný z pluginů v této práci nebude právě z tohoto důvodu ani cílit na to být využitelný jinými pluginy jako knihovna.

#### 4.1.1 Označení modifikací v kódu

Licence KernelSharku, LGPL-2.1, vynucuje u provedených změn v softwaru s touto licencí jasné vyznačení změněných míst společně s datem změny. Každá změna spojená s modifikací tak bude ohraničena dvojicí komentářů:

```
//NOTE: Changed here. ([TAG]) ([DATE])  
...  
// END of change
```

[TAG] je zkratkovité označení po typ modifikace, se kterou změna souvisí - každá modifikace musí mít takovou značku. [DATE] je datum napsání změny ve formátu YYYY-MM-DD. Pokud změna vznikne během vývoje nějaké modifikace, ale není na ni nutně vázána (například je to pomocná funkce s širším využitím), pak stačí ohraničit [TAG] uvozovkami.

## 4.2 Výběr vylepšení

Následující vylepšení umožňují v KernelSharku zobrazovat dodatečné informace, které usnadňují analýzu trasovacích událostí. Součástí jejich popisu budou i názvy těchto vylepšení, které se použijí v dalších kapitolách. Názvy budou anglické pro zachování jednotného jazyka programu. Vylepšení jsou buď pluginy nebo modifikace zdrojového kódu KernelSharku - tato informace bude také součástí jejich popisu.

### 4.2.1 Lepší analýza zásobníku kernelu

#### Identifikace problémů k vyřešení

Zaznamenávací funkcionality KernelSharku, tzv. Record okénko, dovoluje spustit program `trace-cmd record` pro nastartování trasování běžícího systému skrze GUI. Ačkoliv to není ihned zřejmé, tato funkcionality dovoluje uživateli dodat argumenty `trace-cmd`, kterými upraví chování zaznamenávání. Pokud uživatel nezná tyto argumenty, nebude je schopen využít. Jedním z argumentů je `-T`, který zapne zaznamenávání kernel zásobníku do události typu `ftrace/kernel_stack`. Zásobník je zaznamenán po každé události, kromě události zaznamenání zásobníku.

KernelShark již umí zobrazit události typu `ftrace/kernel_stack`. Bere je jako každou jinou událost v trasovacích datech a tak je zobrazí jak v grafu, tak v seznamu událostí. Nicméně nás většinou nezajímá, že se nějaký zásobník trasoval, nýbrž spíš co v něm během události bylo. Z grafu toto nevyčteme, informační řádek nám nedokáže dát celou informaci, jelikož na ní nemá prostor. Seznam událostí je na tom trochu lépe, ale informace zásobníku je v textové formě víceřádková a často má řádků tolik, že většina prostoru seznamu je zabrána jen touto událostí. Celý zásobník je ovšem viditelný pouze, pokud je daná událost v seznamu vybrána,

jinak se zobrazí pouze jedna řádka ze zásobníku. To nám analýzu zpomalí o vybrání zásobníkové události.

## Extrakce požadavků

Zřejmě je zde co vylepšovat. Bylo by jistě příjemnější, kdyby bylo možné zobrazit obsah zásobníku přes záznam této události v grafu, nebo vidět nějakou podstatnou část v informačním řádku, namísto klikání mezi seznamem a grafem. Toto by nemělo být realizováno přes klikání na záznam samotný, jelikož takto uživatel záznamy zvýrazňuje. V tom případě by bylo vhodnější vytvořit tlačítko nad záznamy, jejichž zásobník si chceme zobrazit. To mohou být buď přímo záznamy se zásobníkem, nebo záznamy událostí po kterých byl zásobník zaznamenán. Abychom měli záznamy kde zobrazit, bylo by nejlepší vytvořit nějaké vyskakovací okénko. V tomto okénku by mohla být vedle zásobníku i další data, například po jakém typu události byl zásobník zaznamenán, nebo jakému procesu událost patřila. Přejetím kurzoru myši přes tlačítko bychom mohli donutit informační řádek k zobrazení nějaké zajímavé části zásobníku. Abychom toho byli schopni, bude nutno KernelShark dodatečně vylepšit o akce vyvolané při přejetí kurzorem myši přes objekty v graf. Nakonec by bylo vhodné umět toto chování nějak konfigurovat, třeba nezobrazovat tlačítka nad nějakým typem události, nebo měnit barvu tlačítek. K tomu by mohlo stačit konfigurační okénko vyvolané nějakým tlačítkem v hlavním okně.

Co se zaznamenávání zásobníku týče, ačkoliv je možné vytvořit soubor s událostmi záznamů zásobníku, bylo by uživatelsky přívětivější, kdyby se dala tato možnost zapínat pomocí nějakého GUI prvku. Nejpřirozenějším výběrem by bylo zaškrtačací políčko, umístěné v Record okénku KernelSharku. Obsah okénka nemá se zbytkem programu další vazby, tedy není třeba řešit další kompatibilitu.

Vylepšení Record okénka je možné pouze jako modifikace, pluginy nemají k tomuto okénku přístup. Modifikaci budeme nazývat stručně jako *Record Kstack*. Ovšem tlačítka nad záznamy a vyskakovací okénka je možné definovat čistě v pluginu. Jelikož bude plugin zaměřen na analýzu zásobníku, bude se nazývat *Stacklook*.

Plugin má požadavků více. proto jsou shrnuty v seznamu pro přehlednost:

- Nad podporovanými záznamy bude klikatelný objekt - po dvojitém kliknutí se zobrazí vyskakovací okénko.
- Tlačítka se budou zobrazovat nad záznamy jak v CPU grafech, tak v grafech procesů.
- Ve vyskakovacím okénku se zobrazí záznam zásobníku, název procesu, kterému událost záznamu patří a nějaká bližší informace specifická pro událost, nad kterou bylo tlačítko zobrazeno.
- Plugin bude mít konfigurační okénko, kde se bude moci některé chování upravit. Minimální součástí musí být zapínání a vypínání kreslení tlačítek nad podporovanými záznamy, úprava barev tlačítek a nastavování maximálního počtu viditelných záznamů v grafu, při jehož překročení se tlačítka nebudou zobrazovat.

- Informační řádek bude schopen zobrazit část zásobníku při přejetí kurzoru myši přes tlačítko. Která část zásobníku se zobrazí bude nastavitelné v konfiguraci pluginu. Jelikož informační řádek nemůže být změněn přes rozhraní pluginů a grafové objekty pluginů nereagují na přejetí myši, toto budou dodatečné modifikace, resp. modifikace *Preview Labels Changeable* a *Mouse Hover Plot Objects*.
- Tlačítka mohou být barevná stejně jako je zabarven proces, kterému záznam pod tlačítkem náleží. Získání barev procesů, CPU a streamů není v KernelSharku možné, KernelShark dovoluje barevné tabulky jenom vytvořit nanovo, které ovšem nebudou synchronizovány s tabulkami využívanými KernelSharkem. Zpřístupnění těchto tabulek bude dodatečná modifikace s názvem *Get Colors*.

Podporovanými událostmi budou `sched_switch` a `sched_waking`. Vybrány jsou proto, že budou podstatné i u dalších vylepšení. Tím se dá testovat izolace různých vylepšení, případně i jejich kompatibilita. Zároveň jsou často zajímavé k analýze. `Sched_switch` nám ukáže přepnutí z jednoho procesu na jiný. Zde lze zkoumat, proč byl proces přepnut, například zdali na něco začal čekat. `Sched_waking` nám pak ukazuje změnu spícího procesu na proces připravený k běhu. Tato událost a její záznam zásobníku mlže odhalit proč lze proces opět nechat běžet, například na co se přestalo čekat.

## 4.2.2 Dělení vlastnictví událostí souvisejících se dvěma procesy

### Identifikace problémů k vyřešení

Trasování ukládá mnoho různých událostí, přičemž některé jsou svázané se dvěma procesy. Tyto procesy jsou pak touto událostí spárované. Klasickým příkladem je událost `sched_switch`, tedy přepnutí se z kontextu jednoho procesu na kontext procesu jiného na stejném CPU. `Trace-cmd` dokáže tuto informaci uchovat, nicméně přepnutí je vyvoláno jen přepínajícím procesem. Zaznamená se tak jen jedna událost a to pro tento proces. KernelShark dokáže zobrazit ve svých CPU grafech tyto události v čase, dá se tedy zjistit, který proces byl přepnut do kterého na daném CPU. Nicméně tato informace není tak snadno zjistitelná v grafech procesů. Graf druhého procesu o přepnutí neví. Jediné, co ví, je že jeho proces začal pracovat. Pouze graf procesu přepínajícího bude o události přepnutí vědět a mít ji jako svou součást.

Tento systém, ač většinou funkční, nutí procesy se dělit o informaci, která se týká obou. Informaci pak jeden z nich ztrácí ve svém procesovém grafu. To nutí i některé pluginy upravovat vlastníky událostí, aby dosáhli svých funkcionalit. To je ovšem problém - některé pluginy mohou vyžadovat jiná data, než která jim KernelShark po upravení události dokáže poskytnout. Tedy pluginy měnící vlastníky musejí být načteny buď někdy jindy, nebo jiné pluginy zakazovat. Příkladem mohou být oficiální plugin `sched_events` a další z vylepšení, *Naps*. *Naps* plugin potřebuje mít `sched_waking` události v grafu procesu, který je probouzen. Vedle toho `sched_events` plugin potřebuje mít `sched_switch` události v grafech procesů, na které se přepíná. Dále pak zobrazuje své tvary mezi záznamy událostí



typu `sched_switch` a `sched_switch`, nebo mezi záznamy událostí `sched_waking` a `sched_switch`. Tyto dva pluginy si pak vzájemně narušují očekávaná umístění těchto dvou typů událostí a ani jeden z pluginů nefunguje správně.

Zřejmým řešením problému je pak odstranění nutnosti měnit vlastníky událostí. Například umět události rozdělit a každému z procesů dát jednu polovinu. Ale rozdělování implikuje, že by pak bylo nutné rozdělit i data z jedné události do vzniklých polovin. To ovšem není vhodné, jelikož oba procesy nejspíše budou potřebovat data celé události k efektivní analýze. Lepší bude trochu odlišný přístup. Namísto dělení události dovolíme druhému procesu z páru číst stejná data, ať už kopií události nebo odkazem na ní. Takto nerozdělíme událost samotnou, ale její vlastnictví.

## Extrakce požadavků

Podívejme se, co to znamená pro KernelShark. Záznamy událostí v KernelSharku obalují události z `trace-cmd`, nebo se na ně dokážou odkázat. Jelikož KernelShark používá hlavně své záznamy, bude lepší pracovat s nimi. Z architektury KernelSharku vychází, že můžeme buď přidat umělé záznamy, které se automaticky zobrazí v grafu, nebo v grafu jenom vykreslovat objekty simulující záznam. Druhý přístup není vhodný. Simulace chování by byla zbytečně složitá na implementaci a záznamy v grafu by neodpovídaly záznamům v seznamu. Naopak první přístup jenom vytvoří umělý záznam a KernelShark se postará o zbytek. O co víc, umělé záznamy už KernelShark sám vytváří během vytváření záznamů z trasovacích dat. Přístup je tedy nejen jednodušší na implementaci, ale už i používán. Vytvořené umělé záznamy by měly umět odkázat na záznam původní. Ten totiž obsahuje i původní událost a její data. Zároveň by umělé záznamy měly obsahovat data, díky kterým budou přiřazeny do správných grafů a bude možné je rozlišit od ostatních záznamů, například jménem.

S vylepšením budou muset oficiální pluginy nebo ostatní části KernelSharku umět spolupracovat. Nové záznamy by jako ostatní záznamy měly podporovat rozhraní dotazů na záznamy. KernelShark by s nimi měl také umět alespoň základně pracovat, alespoň je umět kromě zobrazení i jednoduše filtrovat. Jelikož budeme měnit chování KernelSharku, bylo by dobré umět tuto změnu chování zapínat a vypínat. Toto nastavení by mělo být uložitelné do relací. Jediný oficiální plugin, který je nutno poupravit je `sched_events`.

Výše vypsané změny ukazují, že vylepšení nelze napsat jako plugin. Pouze modifikace bude schopná přidávat další záznamy do grafu a seznamu. Modifikaci nazveme *Couplebreak*, podle jejího účelu rozdělit párovou událost mezi dva procesy.

Pro přehlednost sepíšeme vlastní požadavky do seznamu:

- Podporované události dvou procesů dají vzniknout dvěma záznamům - původci a cíli.
- Modifikace bude navržena rozšiřitelně o další události.
- Nové záznamy budou patřit tomu procesu z páru, který předtím událost nevlastnil.
- Nové záznamy budou obsahovat odkaz na záznam s původní událostí.
- Nové záznamy budou splňovat rozhraní dotazů na záznamy KernelSharku.

- Nové záznamy bude možné filtrovat jednoduchým filtrem.
- Vylepšení bude možno zapnout a vypnout. Toto nastavení bude možné uložit do a načíst z relací KernelSharku.
- Součástí vylepšení bude i zajištění kompatibility s pluginem `sched_events`.

Podporovanými událostmi budou `sched_switch` a `sched_waking`. Vybrány jsou proto, že budou podstatné i u dalších vylepšení. Tím se dá testovat izolace různých vylepšení, případně i jejich kompatibilita. Zároveň jsou to události, které pro Naps a `sched_events` představují bez Couplebreaku problém a dá se na nich ukázat využitelnost vylepšení.

### 4.2.3 Vizualizace nečinnosti procesů

#### Identifikace problémů k vyřešení

Procesy se během běhu na CPU často střídají. Důvodů může být několik - proces třeba čeká na otevření souboru, nebo mu prostě vypršel čas na CPU a byl preemptivně přepnut. Vizualizace nečinnosti v KernelSharku chybí, ačkoliv data jsou dostupná v trasovacích událostech. Pokud uživatel potřebuje zjistit předchozí stav procesu (stav před přepnutím), musí si jej najít a vyčíst v seznamu procesů. Pokud uživatel chce zjistit, jak dlouho byl proces nečinný, než byl probuzen, nezbyvá mu než manuálně najít událost přepnutí a následnou událost probuzení - tato událost ovšem patří nějakému jinému procesu.

#### Extrakce požadavků

Cílem vylepšení je vizualizovat pauzu mezi přepnutím procesu (jeho `sched_switch`) a probouzením jiným procesem (`sched_waking` procesu, který první proces probouzí). Lze se inspirovat pluginem `sched_events`, který kreslí obdélníky mezi událostmi pro něj zajímavými. Nicméně samotné obdélníky by nedokázaly nést informaci o předchozím stavu procesu, tedy nějaké barevné či textové označení bude také požadováno. Bez vylepšení Couplebreak bude nutné přesunout záznamy pro `sched_waking` do grafu probouzeného procesu. S vylepšením bude stačit hledat záznamy cílových událostí probouzení. Neměli bychom se snažit kreslit obdélníky neustále. Velké množství by mohlo program zpomalovat. Mnoho záznamů na obrazovce také znamená, že budou v grafu blízko u sebe a obdélníky by pak pro sebe neměly prostor. Jelikož všechny systémy nejsou stejné, toto by mělo být konfigurovatelné v nějakém grafickém okénku. Všechny tyto změny se dají provést v pluginu.

Jelikož se bude zajímat o dobu, kdy jsou procesy nečinné, nazveme plugin *Naps*, kdy naps (česky zdřímnutí) označují právě dobu nečinnosti procesu.

Pro přehled si požadavky sepišeme do seznamu:

- Mezi událostmi `sched_switch` procesu P1 a `sched_waking` procesu P2, který probouzí P1, se bude vykreslovat obdélník a text. Vykreslený text bude název předchozího stavu P1 před přepnutím. Vykreslený obdélník bude měnit svou barvu dle předchozího stavu.

- Plugin musí dostat `sched_waking` události do grafů procesů, které jsou těmito událostmi probouzeny, aby mohl své grafické objekty vykreslovat.
- Plugin bude spolupracovat s vylepšením `Couplebreak`. Namísto využívání `sched_waking` událostí se využijí cílové události probouzení. Spolupráce musí být automatická.
- Plugin bude možné konfigurovat skrze grafické okénko. Minimální konfigurovatelné nastavení bude maximální počet záznamů viditelných v grafu, než se plugin aktivuje.

#### 4.2.4 Vizualizace NUMA topologie CPU vedle grafu

##### Identifikace problémů k vyřešení

KernelShark se primárně zabývá vizualizací trasovacích dat. Ovšem jenom z nich nelze vyčíst vše. Máme-li mnoho procesorů, je možné optimalizovat přístupy do paměti přes NUMA model. NUMA ve zkratce znamená, že některé procesory jsou blíže nějaké paměti či pamětem, což jejich přístupy zrychluje. To není ale tak podstatné jako to, že NUMA vytváří nějakou topologii procesorů - CPU se seskupují do NUMA uzlů. NUMA uzly určují skupinu procesorů a pamětí, ke kterým mají tyto procesory blízko. Právě tu KernelShark nedokáže nijak zohlednit - procesory v grafu označuje pouze z trasovacích dat a to pomocí indexů dodaných operačním systémem. Topologie ovšem ovlivňuje výkon aplikací s více komunikujícími procesy, nebo s procesy s pamětí na vzdáleném NUMA uzlu. Jak kernelový plánovač úloh, tak alokátor paměti se snaží NUMA lokalitu respektovat - zviditelněním v KernelSharku bude možné analyzovat, jak dobrá je tato snaha. Program *hwloc* byl navržen právě pro zkoumání a vizualizaci topologií systémů využívajících NUMA technologii. Tento nástroj umí topologii systému i exportovat do souboru formátu XML a načíst data z takovýchto exportů. Co víc, *hwloc* dokáže zachytit i zdali jsou některá CPU součástí jednoho jádra a jeví se jako samostatná CPU díky hyperthreadingu. Spojením schopností obou programů bychom mohli dodat KernelSharku možnost zobrazovat NUMA topologii procesorů systému a zároveň z grafu KernelSharku vyčíst, která část topologie byla více namáhána, než část jiná.

##### Extrakce požadavků

KernelShark se zobrazováním topologií nijak nepočítá, není pro ně tedy žádná podpora. Topologie bude vytvořena pomocí programu *hwloc*. Nutné tedy bude vytvořit/zabrat místo v hlavním okně, kde se bude vizualizace topologie zobrazovat. Toto místo by bylo dobré umět schovat, pokud bychom topologická data v daný moment nepotřebovali a vybrané místo by byl nevyužitý prostor. Topologii by bylo nejlepší zobrazit takovým způsobem, že CPU grafy by na zobrazení přirozeně navazovaly. *hwloc* nečísluje CPU stejně jako operační systém, povolíme si tedy reorganizaci CPU grafů tak, aby byla respektována topologie. Části topologie by měly být rozlišitelné, alespoň nějakým popiskem, nebo barvou. Struktura topologie by měla být jednoduše pochopitelná, zobrazení ve stromovém stylu by se dalo použít. Dále budeme muset dát uživateli nějaké okénko, ve kterém si vybere soubor s topologií, kterou chce zobrazit, a zdali topologii zobrazit chce. Soubor by

vybíral pro každý otevřený stream zvlášť. KernelShark nijak nevynucuje stejný trasovaný systém v otevřených streamech, dává smysl toto respektovat a vybírat topologie pro každý stream odděleně. Toto vylepšení by také mělo být součástí ukládaných relací, už jen kvůli odstranění času stráveného hledáním a načtením souboru topologie. Nakonec, pokud systém nevyužívá NUMA technologii, pak nemusíme zobrazovat NUMA uzly a zobrazíme pouze jádra. Ta se zobrazí i kdyby byl hyperthreading vypnutý a jedno jádro by bylo ekvivalentní jednomu CPU.

Vylepšení bude jistě modifikací, pluginy zde nemají využití. Modifikaci nazveme *NUMA Topology Views*, zkratkovitě *NUMA TV*.

NUMA TV má tedy tyto požadavky:

- Modifikace bude umět zpracovat topologická data z XML souboru vytvořeného programem hwloc. Z tohoto souboru bude hlavně chtít vyčíst NUMA topologii procesorů.
- Zpracovaná topologická data budou zobrazena někde v hlavním okně. Místo zobrazení by mělo dovolovat přirozenou návaznost na CPU grafy. Ty mohou být přeuspořádány tak, aby respektovaly řazení v topologii.
- Pokud nemáme topologická data k dispozici pro streamy, nebudeme topologii pro dané streamy zobrazovat.
- Topologie budou zobrazovány jako stromy.
- Každý prvek stromu bude viditelně pojmenován. Pokud by jméno bylo příliš dlouhé, lze použít popisky při najetí myši.
- Topologie nebudou zobrazovat NUMA uzly, pokud existuje pouze jeden (a NUMA technologie je tedy nevyužitá).
- Topologie budou vždy vykreslovat alespoň jádra v topologii. Ta budou vždy obsahovat alespoň jeden procesor.
- Jádra budou zabarvena průměrnou barvou ze svých procesů. NUMA uzly budou zabarveny průměrnou barvou jader, která jsou součástí uzlu.
- Místo s topologickými stromy bude možné schovat přes GUI prvek.
- Modifikace bude mít konfigurační okénko, ve kterém si bude uživatel schopen vybrat soubor s topologickými daty a typ pohledu na stream - buď klasický, nebo se zobrazením topologie. Pokud nebude vybrána topologie, ale bude vybrán topologický pohled, bude namísto toho použit klasický pohled. Vybrání souboru topologie s odlišným počtem CPU, než jsou v daném streamu tuto topologii nezobrazí, použije klasický pohled a uživatele o nesrovnalosti informuje.
- Modifikace bude uložitelná do relací.

#### 4.2.5 Dodatečná vylepšení

Následující odstavce v rychlosti představí vylepšení KernelSharku, na která se práce nesoustředí, ovšem jejich existence KernelShark udělá trochu příjemnějším k použití a nebo jsou užitečná ve vícero jiných vylepšeních.

## Aktualizace grafického kódu

KernelShark používá Qt6 v minimální verzi 6.3.0. Nicméně Qt6 verze 6.7.0 přináší nové rozhraní pro kontrolu statusu zaškrtačacího políčka a staré rozhraní označuje při kompilaci jako zastaralé. Tato verze bude již brzy součástí hlavních Linuxových distribucí a aktualizace těchto částí kódu v KernelSharku je na místě. Dodatečným vylepšením bude modifikace kódu KernelSharku, která bude nové rozhraní využívat. Název vylepšení bude samovysvětlující *Update Cbox States*.

## Objekty v grafu reagují na najetí kurzoru myši

Grafová oblast KernelSharku dovoluje uživateli dodat další tvary ke kreslení a interakci. Takové tvary jsou velmi časté u pluginů, příkladem může být `sched_events` nebo `Stacklook`. Tyto tvary mají předdefinovaná rozhraní KernelSharkem, nicméně součástí těchto rozhraní není reakce na najetí myši přes grafický objekt. KernelShark již dokáže událost najetí myši zpracovat, využívá ji při najetí na záznamy. Jednou uživatelskou interakcí s tvary je dvojité kliknutí myši na objekt. Cílem tohoto vylepšení je dodat grafickým objektům pro grafovou oblast možnost reagovat na najetí myši a to podobným způsobem, jako je definováno dvojité kliknutí. Název tohoto vylepšení bude *Mouse Hover Plot Objects*.

## Využití barevných tabulek KernelSharku

KernelShark nedovoluje využívat barvy, které používá pro procesy, CPU a streamy. Jejich využití by ale mohlo v některých případech vylepšit organizaci informací, například `Stacklook` může zabarvovat svá tlačítka barvou procesů, kterým události patří. Lze tak i v CPU grafu identifikovat vlastní proces události, jejíž záznam zásobníku si budeme chtít zobrazit. Navíc, pokud využijeme barvy, které používá KernelShark, pak se využití barvy budou měnit společně s jinou hodnotou barveného slideru, který KernelShark uživateli zpřístupňuje. Vylepšení *Get Colors* dodá KernelSharku, ve formě modifikace, dodá funkce, kterými se získají barvy momentálně využívané KernelSharkem.

## Možnost měnit text v informačním řádku

KernelShark nedovoluje měnit obsah informačního řádku. Možnost jej měnit se může hodit, pokud nějaký plugin, jako například `Stacklook`, by chtěl zobrazit nějaké rychlé informace uživateli. Dodatečným vylepšením bude modifikace kódu KernelSharku, která možnost měnit text v informačním řádku dodá. Vylepšení dostane název *Preview Labels Changable*.

## Obdélníky mezi záznamy

KernelShark při vykreslování grafu dokresluje obdélníky mezi jednotlivé grafické reprezentace záznamů. Tyto obdélníky jsou definovány a nakresleny během každého vykreslení na obrazovku. Ovšem ne všechny záznamy by se měly podílet na kreslení obdélníků, například záznamy zásobníku kernelu se dějí po přepnutí kontextu, kde by měl obdélník většinou skončit, a záznamy vytvořené `Couplebreakem` mohou chybně být začátky a konce jiných obdélníků. Vylepšení *NoBoxes* jako modifikace dodá masku viditelnosti záznamů, kterou se zakáže účast na kreslení

zmíněných obdélníků, a plugin, který touto maskou označí vybrané záznamy událostí. Toto opravné vylepšení bude nicméně fungovat jako best-effort při každém vykreslování a může tak produkovat vizuální anomálie v grafu. Řešení, které by anomáliím zamezilo, bude ponecháno jako rozšíření. [TODO: Nevím, jestli je tohle nejlepší formát pro vyřknutí: "Bylo by to hrozně moc práce, na kterou není čas, a je popravdě nepříjemně zvláštní, že to nefunguje už teď."]

## 5 Record Kstack

Tato kapitola se bude zabývat modifikací, která dodává uživateli přímější způsob, jak zapínat trasování zásobníku kernelu přes GUI KernelSharku. Modifikace je přímá a velmi jednoduchá, kapitola je tak krátká.

### 5.1 Cíl

Dát KernelSharku GUI prvek, kterým se zapne/vypne trasování zásobníku kernelu při trasování přes Record okénko.

### 5.2 Analýza

Jelikož chceme rozšířit GUI, dává smysl se porozhlédnout v kódu KernelSharku po GUI kódu pro Record okno. Jeho kód se nachází v souborech `KsCaptureDialog.hpp/cpp` - zde tedy budeme modifikovat.

#### 5.2.1 Návrh

Modifikace bude nejspíše malá, proto nemá smysl vymýšlet složitý návrh. Nicméně si jako návrhové cíle vytyčíme jednoduchost použití, to nám zajistí Qt knihovna, a podobnost kódu modifikace s kódem KernelSharku. Tento cíl udělá kód příjemnějším ke čtení v budoucnosti.

#### 5.2.2 Implementace

V hlavičkovém souboru najdeme třídu `KsControlCapture`, která sdružuje prvky konfiguruující zachycování. Do jejích datových členů přidáme zaškrtačací políčko. V konstruktoru pak toto políčko nezapomeneme iniciovat a nastavit jako nezaškrtnuté jako výchozí stav. Poté najdeme místo, kde se stavy GUI prvků interpretují na konfiguraci zachycení. Zde přibude překlad zaškrtnutého políčka na přidání argumentu `-T` k ostatním argumentům pro zachytávací program `trace-cmd`. Tím bude modifikace dokončena.

### 5.3 Vývojová dokumentace

Vývojová dokumentace této modifikace slouží k orientaci ve vylepšení.

Modifikace používá značku `RECORD KSTACK` v ohraničeních změn. Jedinými změněnými soubory jsou `KsCaptureDialog.hpp/cpp` - v těchto souborech je přidáno zaškrtačací políčko do Record okénka KernelSharku, přidána je i inicializace tohoto políčka a význam zaškrtnutí.

Modifikace je pouze rozšíření GUI o políčko a přidání jeho sémantiky zaškrtnutí. Políčko musí být nutně v Record okénku, resp. ve třídě `KsCaptureControl`, jelikož nikde jinde nemá nastavení vliv.

## 5.4 Uživatelská dokumentace

Uživatelská dokumentace pojednává hlavně o tom, jak modifikaci instalovat a používat.

### 5.4.1 Uživatel GUI

Stejně jako u jiných modifikací, pokud již máme instalovaný KernelShark s touto modifikací, není potřeba dělat nic. Jinak se musí KernelShark sestavit pomocí oficiálních instrukcí z kódu s touto modifikací. Tato modifikace nevyžaduje upravený soubor CMakeLists.txt.

Po otevření KernelSharku otevřeme okno Record. Zde najdeme zaškrťovací políčko s popiskem „Enable kernel stack tracing“. Zaškrtnutím tohoto políčka povolíme trasování zásobníku kernelu. Poté spustíme trasování a otevřeme výsledná data v KernelSharku. Po každé události, vyjímaje události vytvořené KernelSharkem během načítání trasovacích dat, například Couplebreak události, se budou zobrazovat události ftrace/kernel\_stack.

Změny v okénku lze pozorovat na následujícím obrázku 5.1 (obrázek byl zvětšen pomocí AI). Červené obdélníky zvýrazňují místa, kde se Record okno díky modifikaci změnilo. Černé obdélníky schovávají cesty na autorově stroji.

## 5.5 Rozšíření

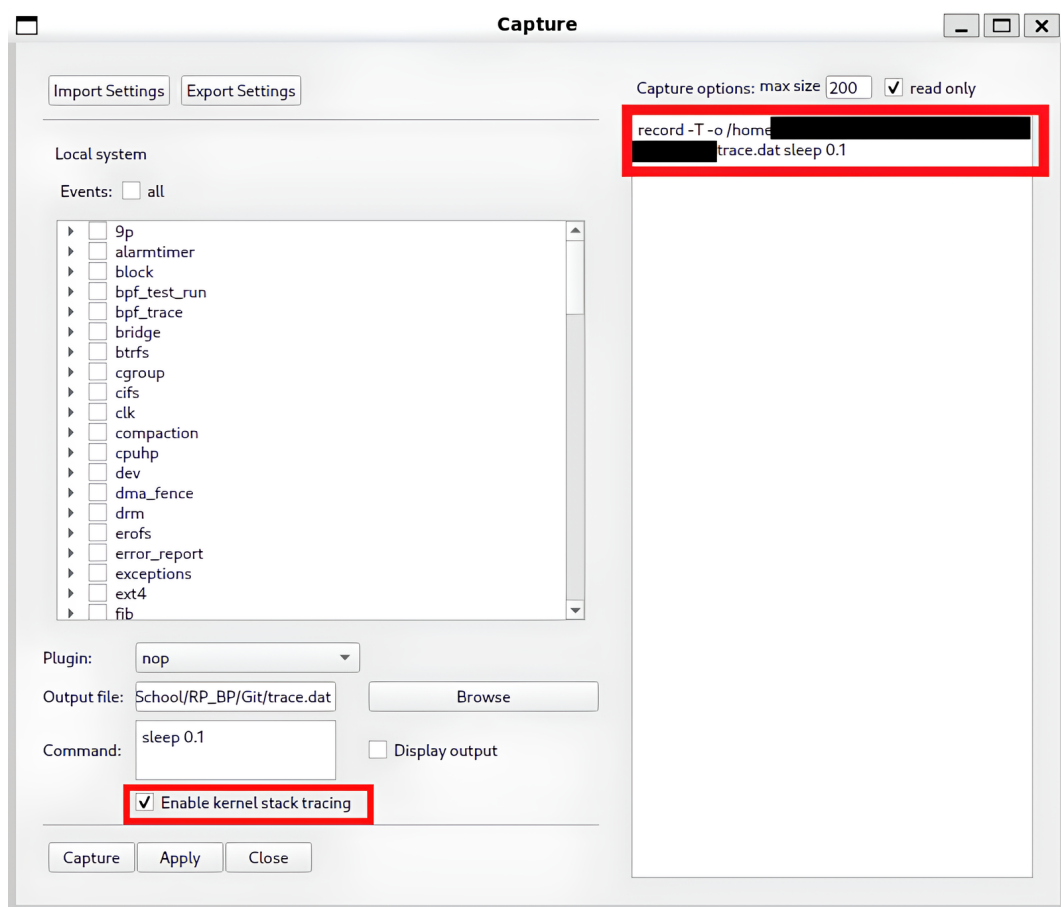
Ne každá událost může mít zajímavý zásobník. Možnost zachytávat pouze po některých událostech by zde pomohla. Nicméně to je spíše rozšíření pro trace-cmd. KernelShark by mohl umět ignorovat záznamy zásobníku jádra, pokud navazují na nějakou událost, po které nás zásobník nezajímá. Data budou stále v souboru, ale KernelShark by je nenačetl.

## 5.6 Zhodnocení splněných požadavků

Jediný vlastní požadavek této modifikace bylo přidání zaškrťovacího tlačítka pro jasnější zapínání sběru zásobníků kernelu. Toto bylo splněno přidáním takového tlačítka do Record okénka a připsání interpretace zaškrtnutí jako dodání argumentu trace-cmd.

Obecné požadavky byly splněny také, GUI prvek byl přidán, kód byl rozšířen. Žádná modifikace existujícího chování se neudála. Tlačítko je ve výchozím stavu nezaškrtnuté a lze jej po zaškrtnutí odškrtnout. Tím jsou obecné požadavky splněny.





**Obrázek 5.1** Zvýraznění změn této modifikace viditelných v GUI

## 6 Stacklook

Tato kapitola se zabývá pluginem pro KernelShark, díky kterému bude jednodušší analyzovat trasovací data, kde se zachytával zásobník kernelu. V kapitole se seznámíme s cíli, analýzou řešení, návrhem a použitím tohoto pluginu.

### 6.1 Cíle

Cíle pluginu jsou obecné požadavky na pluginy, funkcionální požadavky a technické cíle.

Pro zopakování a přehlednost jsou funkcionální požadavky zkopírovány z kapitoly *Obecná analýza a stanovení požadavků*.

- Nad podporovanými záznamy bude klikatelný objekt - po dvojitém kliknutí se zobrazí vyskakovací okénko.
- Tlačítka se budou zobrazovat nad záznamy jak v CPU grafech, tak v grafech procesů.
- Ve vyskakovacím okénku se zobrazí záznam zásobníku, název procesu, kterému událost záznamu patří a nějaká bližší informace specifická pro událost, nad kterou bylo tlačítko zobrazeno.
- Plugin bude mít konfigurační okénko, kde se bude moci některé chování upravit. Minimální součástí musí být zapínání a vypínání kreslení tlačítek nad podporovanými záznamy, úprava barev tlačítek a nastavování maximálního počtu viditelných záznamů v grafu, při jehož překročení se tlačítka nebudou zobrazovat.
- Informační řádek bude schopen zobrazit část zásobníku při přejetí kurzoru myši přes tlačítko. Která část zásobníku se zobrazí bude nastavitelné v konfiguraci pluginu. Jelikož informační řádek nemůže být změněn přes rozhraní pluginů a grafové objekty pluginů nereagují na přejetí myši, toto budou dodatečné modifikace, resp. modifikace *Preview Labels Changeable* a *Mouse Hover Plot Objects*.
- Tlačítka mohou být barevná stejně jako je zabarven proces, kterému záznam pod tlačítkem náleží. Získání barev procesů, CPU a streamů není v KernelSharku možné, KernelShark dovoluje barevné tabulky jenom vytvořit nanovo, které ovšem nebudou synchronizovány s tabulkami využívanými KernelSharkem. Zpřístupnění těchto tabulek bude dodatečná modifikace s názvem *Get Colors*.
- Je nutné podporovat alespoň události typu sched/sched\_switch a sched/sched\_waking.

Techničtější cíle níže nejsou jádrem pluginu a jeho funkcionality, nicméně upřesňují, co může uživatel čekat od této implementace.

- Plugin bude možné sestavit i pro KernelShark bez modifikací vytvořených jako součást této práce. Možnost bude zachytitelná při sestavení, výsledný programový kód se bude lišit.
- Součástí pluginu bude uživatelská a technická dokumentace. Uživatelská dokumentace musí obsahovat instalaci a používání instalovaného pluginu. Technická dokumentace stručně popíše návrh a veřejné prvky pluginu (funkce, datové struktury, třídy a jejich metody). Komentáře v kódu by měly popsat každý prvek pluginu (tj. i privátní či pouze implementační). Technickou dokumentaci lze vytvořit pomocí komentářů v kódu.

První bod bude nejen zajímavým procvičením psaní CMake instrukcí, ale i psaním kódu, který může být kompilovatelný různými způsoby. Toto není jádrový cíl, protože pro desktopovou aplikaci nejspíš nebude tolik podstatný rozdíl výkonu mezi změnou chování za běhu programu či změnou chování při kompilaci.

Druhý bod lze brát spíše jako samozřejmost, jelikož vydat nedokumentovaný kód postrádá na smyslu, pokud se nejedná o jednoduchou kalkulačku. Nicméně existence dokumentace není něco, co určuje fungování pluginu nebo bez čeho plugin nedokáže fungovat.

## 6.2 Analýza

### Struktura adresáře

Adresář pluginu bude obsahovat další adresáře. Jeden z nich pro samotný kód, další z nich pro dokumentaci uživatelskou a dokumentaci technickou, a adresář či adresáře pro sestavení. Na stejné úrovni bude žít i README, soubor s licencí a nejvyšší CMakeLists.txt. V těchto CMake instrukcích se nastaví proměnné sestavení, například typ sestavení, a případně se zavolá generace dokumentace. CMake instrukce zodpovědné za vytvoření binárního souboru dáme do adresáře se zdrojovým kódem, stejně tak to dělá KernelShark.

### Dokumentace

Jedním z technických cílů je dokumentace každého prvku pluginu pomocí zdrojových komentářů a existující technická a uživatelská dokumentace. Technická dokumentace bude vytvořena pomocí nástroje Doxygen, který nám i udává pěkný a známý formát pro komentáře. Technická dokumentace dostane navíc hlavní stránku a stránku s abstraktním návrhem pluginu.

### Cesta k řešení

Zamysleme se nyní nad tím, jak bychom vytvořili plugin pro KernelShark s našimi cíli. Nejlepším začátkem by bylo nechat se inspirovat oficiálními pluginy, od nich se lze naučit, jak plugin propojit s KernelSharkem. Oficiální pluginy definují hlavičkový soubor v jazyce C, kde se definuje kontext pluginu. Tento kontext je pak struktura s globálně přístupnými daty pro plugin. S kontextem se i deklarují kontextové funkce ovládající jeho inicializaci a destrukci přes KernelSharkem definované makro. Krom toho se zde deklarují i další globálně přístupné prvky,

kteřé nevyžadují funkcionality C++. Tento hlavičkový soubor může své části implementovat jak C kódu, tak v C++ kódu. Oficiální pluginy mají v C napsanou hlavně definici kontextových funkcí, registraci handlerů pro různé události během načítání trasovacích dat, registrace handlerů na kreslení do grafu, inicializaci dat pro textový font a inicializaci ukazatelů na hlavní okno. Handlers jsou buď implementovány v C++, nebo, hlavně u jednoduchých handlerů událostí, ještě v tomto implementačním C souboru. Kód napsaný v C je tedy většinou použit na inicializaci pluginu a další části, většinou s komplikovanější business logikou, jsou implementovány v C++.

Postup oficiálních pluginů je dobrý, využijeme jej tedy též. Stacklook ovšem bude vyžadovat více C++ kódu. Jednotlivé soubory pak budou sloužit jednotlivým modulům Stacklooku, například modul konfigurace bude obsahovat datovou strukturu či datové struktury, které konfiguraci tvoří. Pokud možno, hlavičkové soubory by pak měly reprezentovat každý jeden z modulů. Podobné postupy jsou hojně využívány, hlavně při dekompozicích souboru s kódem, který obsahuje vícero modulů najednou.

Dle našich cílů lze vymezit moduly celkem přímo. Moduly budou následující:

- *Propojující modul* - Modul s kódem propojujícím KernelShark a plugin. Obsahem budou hlavně kontext pluginu a implementačně pomocné funkce, které využívají ostatní moduly. Součástí tohoto modulu bude právě část s C kódem a část v C++ pro zbytek hlavičkového C souboru. Právě v „C++ části“ (napsaná v C++) se ostatní moduly budou propojovat; zároveň tato část bude ukládat některá globální data s C++ typy.
- *Konfigurace* - Pro konfiguraci bude sloužit konfigurační modul, který bude obsahovat alespoň třídu pro konfigurační okénko. Pro oddělení této interaktivní složky vytvoříme i třídu pro data konfigurace samotné. Plugin pak bude využívat data hlavně z tohoto objektu a konfigurační okénko bude sloužit pouze jako GUI pro náhle aktuální konfigurace a její změnu.
- *Tlačítka* - Tlačítka budou vykreslována v grafu nad záznamy, budou tedy nějaký grafický objekt, na který lze klikat a který dokáže reagovat na přesun myš nad ním.
- *Detailní pohledy* - Detailní pohledy budou okénka zobrazující hlavně zaznamenaný kernel zásobník a nějaké informace o procesu a události, která záznamu zásobníku předcházela.
- *Předchozí stavy* - Mini-modul, bude dodávat data tlačítkům a detailním pohledům o stavu procesu před přepnutím kontextu na CPU.

## Moduly podrobněji

Nyní detailněji zanalyzujeme jak budou moduly vypadat. Propojující modul máme zčásti navržen díky oficiálním pluginům. V „C části“ (napsaná v jazyce C) nastavíme textový font používaný v pluginu, vybereme podporované události při načítání dat a registrujeme handlers. V „C části“ zároveň inicializujeme pluginový kontext. Tato část bude mít za úkol tento kontext i správně odstranit. V „C++ části“ definujeme handlers pro kreslení a přístup k hlavnímu oknu. Kreslicí

handler bude vyžadovat funkci na vytvoření tlačítek, tedy zde se propojí tlačítkový modul. Funkce vytvoření tlačítek bude potřebovat data pro barvu tlačítek, ta je v konfiguraci, takže se zde objeví i konfigurační modul.

Konfigurace bude rozdělená mezi GUI okno a datová struktura s konfiguračními daty. Konfigurační objekt samotný implementujeme jako singleton. Důvodem je nutnost znát konfiguraci v různých částech programu a že konfigurace musí být vždy pouze jedna, což právě singleton splňuje. Prvky z cíle o konfiguraci [TODO]

Konfigurační okno využije framework Qt (specificky Qt6), stejně jako ostatní grafické prvky KernelSharku. Na barevná nastavení vytvoříme tlačítko, které na kliknutí vyvolá nějaké okno na výběr barvy, například výchozí barevný dialog od Qt. Pro pohodlí uživatele i někde blízko tlačítka výběru barvy zobrazíme i aktuálně použitou barvu. Nastavení maximálního počtu viditelných záznamů lze reprezentovat jako spinbox. Konfigurace pro specifické události, tj. zdali nad danými událostmi zobrazovat tlačítka nebo offset použitý při zobrazování části zásobníku v informačním řádku, lze reprezentovat pomocí zaškrtačacího tlačítka a dalšího spinboxu. Mimo požadavky navíc dodáme zaškrtačací tlačítko pro barvení tlačítek barvami jejich procesů.

## 6.3 Vývojová dokumentace

Cíle/úvod této sekce...

### 6.3.1 Modifikované soubory

### 6.3.2 Struktura pluginu

## 6.4 Uživatelská dokumentace

[TODO: Přeložit]

This document serves as a simple to grasp manual for the "Stacklook" KernelShark p

![Fig. 1](../images/SlWorking.png)

Figure 1.

# "How do I build and install Stacklook?"

## Prerequisites

- CMake of version at least 3.1.2
- KernelShark and its dependencies
  - version \*2.4.0-couplebreak\* and higher for custom KernelShark
  - version \*2.3.2\* for unmodified KernelShark
- Doxygen for documentation

## Compatibility

Plugin is compatible with KernelShark's **custom** version \*2.4.0-couplebreak\* and

Unmodified KernelShark usage is achievable through a build argument. Unmodified KernelShark features from the plugin:

- Buttons optionally being the same color as the task is in the graph
- Mouse hover showing a preview of the kernel stack with an adjustable stack offset

No other dependencies are necessary, except maybe the standard libraries of C and C++.

## Build and install only this plugin

1. Set your working directory in terminal as the build directory (best created in the directory of [README](../README.md)), if not already present).
2. Run ``cmake ..`` command (if the main ``CMakeLists.txt`` file isn't in the parent directory, specify a valid location).
  - If using an unmodified KernelShark copy, add ``-D_UNMODIFIED_KSHARK`` to the command.
  - If **Doxygen documentation** is desired, include ``-D_DOXYGEN_DOC=1`` in the command.
  - By default, the **build type** will be ``RelWithDebInfo`` - to change this, e.g. ``-DCMAKE_BUILD_TYPE=Release``.
  - If **Qt6 files** aren't in ``/usr/include/qt6``, use the option ``-D_QT6_INCLUDE_DIR=[PATH]``, where ``[PATH]`` is replaced by the path to the Qt6 files.
    - Build instructions still expect that the specified directory has same inner structure (i.e. it contains ``QtCore``, ``QtWidgets``, etc.).
  - If **KernelShark source files** aren't in the relative path ``../KS_fork/src``, use the option ``-D_KS_INCLUDE_DIR=[PATH]``, where ``[PATH]`` is replaced by the path to the source files.
  - If **KernelShark's shared libraries** (``*.so`` files) aren't in ``/usr/local/lib``, use the option ``-D_KS_SHARED_LIBS_DIR=[PATH]``, where ``[PATH]`` is replaced by the path to the libraries.
3. Run ``make`` while still in the ``build`` directory.
  - If only a part of building is necessary, select a target of your choice.
  - Just running ``make`` builds: **the plugin** (target ``stacklook``), **symlink** (target ``stacklook_symlink``) and, if specified, the **Doxygen documentation** (target ``docs``).
4. (**Installation**) Plug in the plugin into KernelShark - either via KernelShark's CLI with the ``-p`` option and location of the symlink or the SO itself.
  - **IMPORTANT**: Always install/load the plugin before loading a session where the plugin is used, so will result in KernelShark not completely loading configuration menus or modules.

Use ``make clean`` to remove built binaries.

## Building KernelShark from source and this plugin with it

1. Ensure all source files (``*.c``, ``*.cpp``, ``*.h``) of Naps are in the ``src/plugins`` subdirectory of the project directory.
2. Ensure the ``CMakeLists.txt`` file in said subdirectory contains instructions for building the plugin (and other Qt-using GUI plugins). Adjust the build instructions if ``_UNMODIFIED_KSHARK`` is defined.
3. Build KernelShark (plugins are built automatically).
4. (**Installation**) Start KernelShark. Plugins built this way will be loaded automatically. If loading failed, look for the SO as for any other default-built KernelShark plugin, again.

## WARNING - loading plugin twice

If you have two or more versions of the plugin, do **\*\*NOT\*\*** load them at the same time in the same program. Use either one or the other, but **\*\*NEVER BOTH\*\***.

# "How do I enable/disable Stacklook?"

Enabling the plugin is very simple. All one has to do is open KernelShark and navigate to the `Tools > Manage Plotting plugins` toolbar menu button. If the plugin was loaded via the GUI line interface,`

it will be shown in the list of plotting plugins as a checkbox plus the name, checked. If not, it is possible to search for the plugin via provided `Tools > Add plugin` button. You can find the symlink, but searching for the actual shared object file is possible too. This follows standard KernelShark plugin loading behaviour.`

![Fig. 2](../images/SlManagePlottingPlugins.png)

Figure 2.

Ticked checkbox means the plugin is enabled, empty checkbox means the plugin is disabled.

# "How do I use Stacklook?"

## Configuration

Plugin configuration can be done at any time, even before any trace file is loaded. Open the configuration dialog window through `Tools > Stacklook Configuration` in the main KernelShark configuration window open (figure 3).`

![Fig. 3](../images/SlConfigWindow.png)

Figure 3.

If using the unmodified build, the configuration window will be missing the checkbox for the offset for the preview labels (figure 4).

![Fig. 4](../images/SlUnmodifiedKsharkConfigWindow.png)

Figure 4.

Now to explain each of the options and how to control them. In top to bottom order:

- **\*Histogram entries limit\*** - Decrease this number to force Stacklook to activate histogramming. Equal to this many entries visible. Lesser the number, greater the zoom necessary. If set to 0, maximum is 1 000 000 000 (one billion) - though this high of a value is not recommended. By default, the value is 10 000 (ten thousand).
- **\*Use task colors for Stacklook buttons\*** - Check this box (if present) to color Stacklook buttons according to the task which owned the event Stacklook found kernel stack trace for. See figure 6 for Stacklook button colors (figure 6). By default, this option is off.█

![Fig. 5](../images/SlDefaultColors.png)

Figure 5.

![Fig. 6](../images/SlTaskColors.png)

Figure 6.

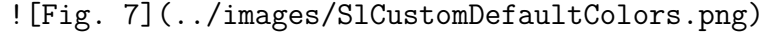
- *\*Stacklook button & outline colors\** - The "Choose" buttons will bring up a color picker to choose a color to be used for Stacklook button's filling or a color its outline. These buttons have a checkbox either is not present or is not checked. Next to this button is a space for a color to be picked for the buttons. By default it is white for filling, black for outline. See Figure 7. [Fig. 7](../images/SlCustomDefaultColors.png)

Figure 7.

- *\*Per-event configurations\** - Each event is assigned a checkbox toggling Stacklook (if present) an spinbox with an offset into the kernel stack, which is used to read from the kernel stack trace. The maximum value of the offset is the 100 000 000. Again, the maximum value will probably never be used. By default, the values for the offset of 3.
  - To reiterate, the spinbox will not appear if the unmodified KernelShark version is used.

The `Apply` button will save the changes made and close the dialog - if not pressed. Only active configuration values show up in the control elements - the configuration is applied to it after closing, unless they were applied. The `Close` button and top-right corner X button will discard changes made in the window and close the dialog.

There is no configuration of:

- Supported events - plugin currently only support `sched/sched\_switch` and `sched/sched\_wakeup`
- The text in stacklook windows
- The text in stacklook buttons
- Stacklook button sizes
- Stacklook button positions

## In the graph

After loading (and maybe configuring) the plugin, zoom in until less than the configuration is visible in the graph. A button will show up above each supported entry, colored by the configuration, or if using custom KernelShark and having the corresponding option in the graph. The task color option is fully compatible with KernelShark's color slider.

The plugin won't show the buttons above event entries that aren't `sched/sched\_switch` or `sched/sched\_wakeup` such entries are missing their kernel stack trace event.

### Hovering over buttons

*\*If using custom KernelShark\**, hover over any button and look at the top-left quarter of the screen. You will see the preview label's contents change to have:

- Task name as the leftmost (first) entry.
- First item in the kernel stack, offset by configured number for this event type (newer entries go)
- Item in the kernel stack after the first item
- Item in the kernel stack after the second item
- Either three dots "...", signifying that the kernel stack has more entries than shown.



signifying there are no more items to display after the ones shown.■

See figure 8 for a small showcase. Included is also the Stacklook window (more on stack trace in the main window's list view of all events - both to really show that in the kernel stack (chosen offset was the default of 3). There's also a red circle cursor hovered over.

![Fig. 8](../images/SI\_MouseHover.png)

Figure 8.

It is possible to set the offset so high that only last one, two, three entries are preview. In that case, Stacklook shows just a dash and a the "(End of stack)" message.

![Fig. 9](../images/SI\_TooBigOffset.png)

Figure 9.

### Double-clicking on buttons

Upon double-clicking a Stacklook button, a new window will open. This is a \*Stacklook\* window that we are viewing a kernel stack of a task (whose name is shown at the sentence 'Task: ...'). Whether task has woken up (which shows only for `sched/sched\_waking` events) or about to wake up (for `sched/sched\_switch` events). Then two radio buttons and the view with the kernel stack entry is shown. The radio buttons toggle what kind of view is used for the kernel stack - \*By default\*, the view is set as raw text, which means the kernel stack is just a single line of text useful for copying the stack as a single string or for highlighting only a specific line. - \*Alternatively\*, the stack can be viewed in list view, which allows for simpler navigation.

There can be multiple windows present for a single entry, there can be multiple windows for different entries, any mix of the two previous situations.

All of the above for this section can be seen below in figure 10.

![Fig. 10](../images/SI\_MultipleWindowsMultipleViewsTwoSameEntries.png)■

Figure 10.

Toggle between the raw text view and the list view using two radio buttons above the stack.

The window can be closed with the `Close` button at the very bottom of the window or the window's header. Last option is to close the main KernelShark window, which will close all windows.

## Using Stacklook as a library

See technical documentation, as this is not intended usage of the plugin and such.

# Bugs & glitches

If multiple stacklook buttons would lay over one another, the one belonging to an event drawn above a button for an event happening *\*later\**, **\*\*BUT\*\*** clicking on or hovering over the button for the *\*later\** event. This is, to the author's knowledge, something not yet fixed on the plugin side.

Opening a KernelShark session where Stacklook was active, but without preloading a session, results in a **\*\*crash\*\*** upon hovering over a Stacklook button.

If more are discovered, contact the author via e-mail ``djsebofficial@gmail.com``.**]**

#### # Recommendations

A few recommendations of usage by the author for the smoothest user experience.**]**

Always preload Stacklook before loading a session. It can save the program from crashing.

Do not open hundreds upon hundreds of stacklook windows, lest you wish your memory to be freed.

It is recommended to not set the histogram limit in the configuration too high as it uses too much memory with many of stacklook's buttons being present.

While KernelShark's sessions work, they are a little buggy. This plugin attempts to fix some of their inner logic, but a warning should be issued that if the plugin isn't loaded correctly, unexpected behaviours, e.g. loading a session when the plugin was active won't add items to the ``Tools`` menu.

### 6.4.1 Instalace

### 6.4.2 Uživatel GUI

### 6.4.3 Vývojář pluginů

## 6.5 Rozšíření

Cíle/úvod této sekce...

## 6.6 Zhodnocení splněných požadavků

# 7 Couplebreak

Couplebreak je modifikace zdrojového kódu KernelSharku a dává mu nové funkcionality...

## 7.1 Cíle

Hlavním cílem modifikace je dodat programu schopnost rozdělovat trasovací události, které náleží dvěma procesům. Z toho plynoucím cílem je i dodatečná kompatibilita této funkcionality s ostatními částmi programu a oficiálními pluginy.

## 7.2 Analýza

Cíle/úvod této sekce...

### Finální podoba

Cíle/úvod této sekce...

Terminologie Couplebreaku je následující:

- *Couple/pár* je označení pro dva procesy, které sdílí nějakou událost. Například trasovací událost `sched_waking`, kdy nějaký proces rozhodne o probuzení jiného procesu, přirozeně obsahuje dva procesy - probouzejícího a probouzeného. Páry se dají často rozdělit na procesy cílové a počáteční.
- *Couplebreak událost* je fiktivní událost vytvořená Couplebreakem. Takto vytvořené události mohou být počáteční i cílové, Couplebreak toto vyznačuje v sufixu jména události jako „[origin]“ nebo „[target]“. Každá taková událost v KernelSharku obsahuje ve svém jméně prefix „couplebreak/“, podobně jako události scheduleru obsahují prefix „sched/“. Tyto události mají pevně stanovené negativní hodnoty identifikátorů.
- *Couplebreak záznam* je záznam vytvářený Couplebreakem pro Couplebreak událost. Tyto záznamy obsahují odkaz na
- *Origin/počáteční proces* je proces z páru, pro který existuje nějaká událost, se kterou tento proces ovlivní druhý proces z páru.
- *Origin event/počáteční událost* je označení pro událost, která náleží počátečnímu procesu. Tento typ událostí momentálně není vytvářen Couplebreakem.
- *Target/cílový proces* je proces z páru, pro který existuje nějaká událost, která tento proces nějak ovlivní.
- *Target event/cílová událost* je označení pro událost, která náleží cílovému procesu. Pouze tento typ událostí je momentálně vytvářen Couplebreakem.
- Termíny (*datový stream*, *záznam*, *událost*) jsou převzaty z terminologie KernelSharku.

## 7.3 Vývojová dokumentace

Cíle/úvod této sekce...

### 7.3.1 Modifikované soubory

Modifikace používá značku COUPLEBREAK v ohraničeních změn. Níže je abecedně seřazený seznam spolu s krátkým popisem změn uvnitř souboru.

- *KsMainWindow.hpp/cpp* - v těchto souborech byly o hlavního okna přidány grafické elementy pro ovládání Couplebreaku přes GUI.
- *KsWidgetsLib.hpp/cpp* - v těchto souborech byl definován nový widget, přes který se Couplebreak zapíná a vypíná pro jednotlivé streamy; zároveň se zde upravil widget filtrující události (třída *KsEventCheckboxWidget*).
- *KsUtils.hpp/cpp* - v těchto souborech byla definována pomocná C++ funkce k získání identifikátorů všech Couplebreak událostí aktivních ve streamu.
- *libkshark.h/c* - v těchto souborech byla upravena datová struktura pro datové streamy, datová struktura definující rozhraní streamů, inicializace hodnot při alokaci a konstrukci nového streamu a nakonec byla přidána definice nové funkce v rozhraní streamů pro získání identifikátorů všech Couplebreak událostí aktivních ve streamu.
- *libkshark-configio.c* - do tohoto souboru se přidalo ukládání stavu Couplebreaku do relací.
- *libkshark-couplebreak.h/c* - v těchto souborech jsou definovány identifikátory pro jednotlivé Couplebreak události jako makra, pozice indikátorů v bitové masce aktivních Couplebreak událostí ve streamech a pomocné funkce s Couplebreakem spojené: získání původního záznamu, získání pozice indikátoru z ID původní události, získání pozice indikátoru z ID Couplebreak události, získání ID Couplebreak události z pozice indikátoru, zda je daná událost Couplebreak událostí a získání jména Couplebreak události z ID události.
- *libkshark-tepdata.c* - zde se Couplebreak záznamy vytvářejí a upravují; zároveň jsou zde upravené implementace rozhraní streamů a nastavování přidáných datových polí streamů.
- *sched\_events.c* - zde byl plugin upraven tak, aby respektoval aktivní Couplebreak a aktivně ho využíval ve svůj prospěch.

### 7.3.2 Struktura modifikace

Couplebreak se dá rozdělit na pět částí: nové API, integrace s datovými streamy, spolupráce s relacemi, spolupráce *sched\_events* s Couplebreakem a konfigurace Couplebreaku.

Nové API...

Integrace s datovými streamy...

Spolupráce s relacemi...  
Spolupráce sched\_events s Couplebreakem...  
Konfigurace Couplebreaku...

## **7.4 Uživatelská dokumentace**

Cíle/úvod této sekce...

### **7.4.1 Uživatel GUI**

### **7.4.2 Vývojář pluginů**

### **7.4.3 Vývojář KernelSharku**

## **7.5 Rozšíření**

Cíle/úvod této sekce...

## **7.6 Zhodnocení splněných požadavků**

# 8 Naps

Cíle/úvod této kapitoly...

## 8.1 Cíle

## 8.2 Analýza

Cíle/úvod této sekce...

## 8.3 Vývojová dokumentace

Cíle/úvod této sekce...

### 8.3.1 Modifikované soubory

### 8.3.2 Struktura pluginu

## 8.4 Uživatelská dokumentace

Cíle/úvod této sekce...

### 8.4.1 Instalace

### 8.4.2 Uživatel GUI

### 8.4.3 Vývojář pluginů

## 8.5 Rozšíření

Cíle/úvod této sekce...

## 8.6 Zhodnocení splněných požadavků

# 9 NUMA Topology Views

Cíle/úvod této kapitoly...

## 9.1 Cíle

## 9.2 Analýza

Cíle/úvod této sekce...

### 9.2.1 Terminologie

## 9.3 Vývojová dokumentace

Cíle/úvod této sekce...

### 9.3.1 Modifikované soubory

### 9.3.2 Struktura modifikace

## 9.4 Uživatelská dokumentace

Cíle/úvod této sekce...

### 9.4.1 Nové závislosti KernelSharku

### 9.4.2 Uživatel GUI

### 9.4.3 Vývojář pluginů

### 9.4.4 Vývojář KernelSharku

## 9.5 Rozšíření

Cíle/úvod této sekce...

## 9.6 Zhodnocení splněných požadavků

# 10 Dodatečná vylepšení

Jedná se o vylepšení, která vznikla kvůli ostatním vylepšením, nebo jako rychlá zlepšení chování KernelSharku. Pro každé z nich bude stručně popsán návrh, řešení, použití a případná varování.

## 10.1 Aktualizace kódu zaškrťávacích políček

Nejpřímějším zlepšením byla aktualizace grafického kódu, aby využíval novější Qt API pro zaškrťávací políčka. V nové API se nahrazuje funkce `QCheckBox::stateChanged` za `QCheckBox::checkStateChanged`. Hlavní změna se týká v argumentu značící zaškrtnutí - předtím stačilo dodat argument typu `int`, od Qt 6.7 ale nová funkce vyžaduje jednu z enumerovaných možností specifických pro `QCheckBox`.

Aktualizace nakonec spočívala pouze v nahrazení signálu `stateChanged` na `checkStateChanged` a použití výčtových hodnot (dle původního čísla) namísto pouze celých čísel.

KernelShark původně vynucoval v sestavení Qt verze 6.3, nyní vyžaduje alespoň verzi 6.7, aby mohl využít novější signál pro zaškrťávací políčka.

Změněné soubory: nejvyšší *CMakeLists.txt* KernelSharku (v `KS_fork` adrese), *KsTraceViewer.hpp/cpp*, *KsCaptureDialog.hpp/cpp*, *KsMainWindow.hpp/cpp*. Značkou modifikace v C++ kódu a CMake instrukcích je `UPDATE CBOX STATES`.

Pokud mají v plánu vývojáři KernelSharku či jeho pluginů dále pracovat s KernelSharkem, musejí tedy použít alespoň Qt6 verze 6.7.0.

## 10.2 Zpřístupnění barev užívaných v grafu

Modifikace je pouze zpřístupnění barevných tabulek, které KernelShark používá pro streamy, CPU a procesy. Díky zpřístupnění pak mohou tyto tabulky využívat i jiné části programu, nebo i pluginy. Jedná se o pouhé získání `const` reference na objekty využívané uvnitř `KsGLWidget` objektu. Přirozeně se jedná o malé úpravy, lokalizované v souboru *KsGLWidget.hpp*, se kódovou značkou `GET COLORS`. K využití této modifikace stačí mít přístup k GL objektu a zavolat nové metody.

K využití přes pluginy se váže varování - pokud si KernelShark uloží do relace plugin, který využívá tuto modifikaci, ale načtení pluginu není dokonalé, program při první snaze o získání hodnoty z jakékoliv z tabulek spadne. Plugin je nedokonale načten vždy, pokud KernelShark načte relaci s daným pluginem, ale ten není předem explicitně načten uživatelem, tedy skrze argumenty při spouštění, nebo přes GUI. Další možnou podmínkou je, že plugin není postaven jako oficiální pluginy KernelSharku, tedy během sestavování KernelSharku samotného. Takto lze postavit například plugin `Stacklook`. Tato podmínka ovšem nebyla rigorózně otestována a je to spíše pouhá domněnka. Chybě se lze vyhnout třemi způsoby: buď uživatel vždy explicitně načte plugin, nebo plugin nebude tyto tabulky využívat, nebo bude obsahovat výchozí hodnotu, kterou použije namísto tabulek při načtení z relace - barevné tabulky se využijí až později, například až pokud si je uživatel zapne v konfiguraci pluginu.



Tuto modifikaci využívají pluginy Stacklook a Naps pro barvy procesů, a modifikace NUMA Topology Views pro barvy procesorů.

Příklad: Stacklook nabízí možnost barvit tlačítka dle barev procesů, kterým patří, viz obrázek 10.1.

## 10.3 Reakce objektů v grafu na přjetí myši

Tato modifikace dodala všem potomkům třídy `KsPlot::PlotObject`, jednodušší plot-objekty, veřejnou metodu pro reakci na přjetí myši a redefinovatelnou privátní virtuální metodu, kterou veřejná metoda volá pro viditelný objekt. Privátní metoda má výchozí definici prázdnou. Krom toho je dodána detekce přjetí přes plot-objekt v grafu a reakce na přjetí. Toto chování bylo vsunuto na konec zpracování události pohybu myši a funguje podobně jako detekce dvojitého kliknutí. Soubory s modifikací: *KsPlotTools.hpp* s novými metodami a *KsGLWidget.cpp* pro vsunutou detekci a reakci. Kódová značka modifikace je `MOUSE HOVER PLOT OBJECTS`.

Nabízí se zde otázka, zdali je toto dostatečně výkonná implementace - myš se pohybuje často, objektů může být mnoho. Řešení musí vždy projít všechny plot-objekty a u každého se rozhodnout, zdali reagovat na myš či nikoliv. Při praktickém použití s rozumnými limity pro zobrazované plot-objekty (nedává smysl neustále zobrazovat tlačítka Stacklooku nad každým prvkem grafu, vedlo by to k přemíře informací) nenastaly problémy a nejvíc program zpomaloval objem dat a náhlé vykreslování objektů, nikoliv práce této modifikace. Pokud bude ale objektů příliš mnoho, výkon může být ovlivněn. Proto se optimalizace výkonu nechává jako *rozšíření* této práce. Inspirací může být nahrazení lineárního prohledávání for-cyklem vyhledáváním přes souřadnice, tedy přes nějaké mapování souřadnic na objekty na těchto souřadnicích.

Tuto modifikaci používá hlavně plugin Stacklook.

## 10.4 Měnitelné nápisy v hlavičce grafu

Tato modifikace přidává do souborů *KsTraceGraph.hpp/cpp* veřejnou funkci, s níž lze přepsat obsah informačního řádku KernelSharku. Jedná se o funkci typu setter, pouze nastaví hodnoty nápisů v informačním řádku.

K použití stačí mít k dispozici `KsTraceGraph` objekt. Pak se dá s modifikací pracovat i v pluginech a jiných částech KernelSharku.

V kódu lze tuto modifikaci nalézt pod značkou `PREVIEW LABELS CHANGEABLE`.

I zde se objevuje bug ze sekce o zpřístupnění barev využívaných v grafu, jenom se tentokrát netýká tabulek, nýbrž informačního řádku. Zde ale nelze spoléhat na nějaké výchozí hodnoty, tedy uživatel musí buď plugin vždy explicitně načíst, nebo nepoužívat plugin využívající tuto modifikaci.

Příklad použití: tlačítka Stacklooku (v červeném kroužku) žádají na přjetí myši o zobrazení několika prvků zásobníku kernelu, viz obrázek 10.2.

## 10.5 NoBoxes

Během vývoje se ukázalo, že některé záznamy by se neměly účastnit kreslení obdélníků mezi záznamy (přesněji, mezi biny). Těmito záznamy jsou ty pro události od Couplebreaku a pro ftrace/kernel\_stack událost. Záznamy od Couplebreaku nezaznamenávají opravdovou práci. Události se zásobníkem dělají nepořádek hlavně po událostech typu sched/sched\_switch, po které by žádný obdélníček neměl být nakreslen, jelikož se přepnul CPU kontext. Jenže událost se zásobníkem je dle KernelSharku validní práce a tak se kreslí další obdélník.

Příklad špatného zobrazování je na obrázku 10.3. Na tomto obrázku je ftrace/kernel\_stack událost vyznačená velkou svislou čarou vpravo, couplebreak/sched\_waking[target] je událost s velkou svislou čarou vlevo. Událost ftrace/kernel\_stack vytváří velký obdélník až do konce grafu, ačkoliv se událo po přepnutí kontextu a procesor ve skutečnosti po zachycení dále nepracuje. Událost couplebreak/sched\_waking[target], ačkoli neprovádí žádnou skutečnou práci na procesoru, vytváří dojem, že ano právě díky nakreslenému obdélníčku. V grafu CPU 1 je také velký obdélník, který začíná u události zachycení zásobníku kernelu a neměl by tedy být kreslen. Události zachycení zásobníku jsou časté a v grafu je více obdélníků spojených právě s nimi.

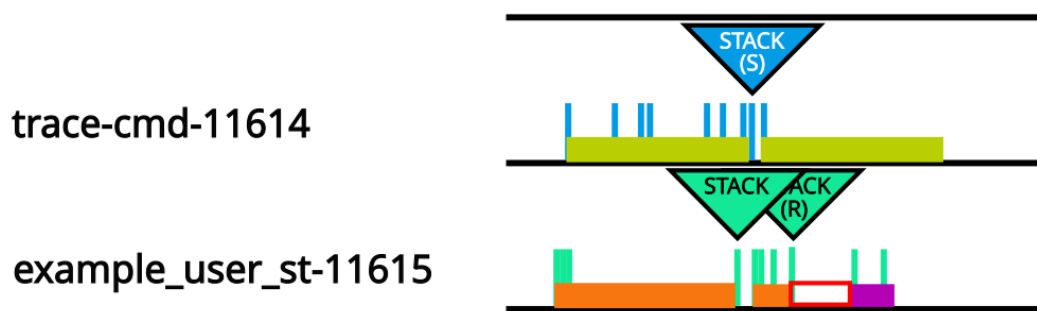
Toto chování je nepříjemné a tak byla snaha jej opravit. Nicméně KernelShark vykreslování obdélníků řeší pro každé vykreslení zvlášť a pouze podle vlastností binů, tedy sdružení jednoho či více záznamů. V binech se pak lze spolehnout na málo, ale nejlépe na data viditelnosti, ve kterých mohou být nastaveny některé bity jako přepínače chování. Viditelnost binu lze ovlivnit viditelností záznamů, které sdružuje. Tak byla představena nová maska pro nastavování sedmého bitu pole viditelnosti pro záznamy, `KS_PLUGIN_UNTOUCHED_MASK`. Tato maska značí, zdali se záznam účastní kreslení mezi-záznamových obdélníků. Pokud bin využije viditelnost záznamu, který má daný bit nastaven na 0, bude moci toto nastavení detekovat. Při vykreslování obdélníků pak žádné obdélníčky nezačínají a nekončí u binů s tímto bitem vynulovaným.

Maska byla přidána do enumerace pro masky viditelnosti. Tento výčet již předtím počítal s nějakým rozšířením, tak bylo jedno z volných bitových míst zabráno naší novou maskou. Rozšíření počítalo s nějakou `KS_X_VIEW_FILTER_MASK` pro volná místa - nicméně naše maska je spíše podobná masce `KS_PLUGIN_UNTOUCHED_MASK`.

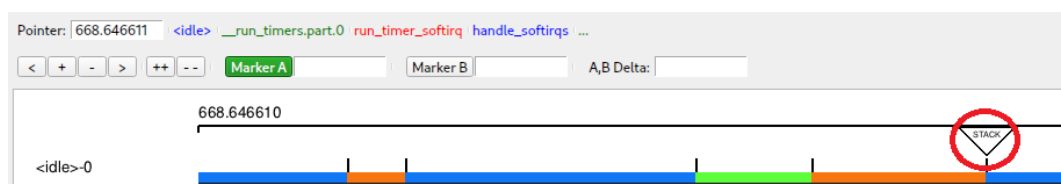
Na výběr záznamů, kterým se nastaví bit na 0, byl vytvořen plugin pro tuto modifikaci, nese název NoBoxes. V jeho kódu jsou zapsány události, na jejichž záznamy má být maska použita, plugin pak lze zapínat a vypínat jako každý jiný. Pokud plugin není načten, modifikace nemá na chod KernelSharku vliv.

Příklad fungujícího vylepšení je na obrázku 10.4. Oproti minulému obrázku (obrázek 10.3) ubylo několik obdélníků a graf nyní odpovídá skutečné práci.

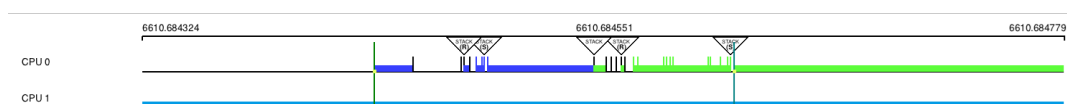
Problém tohoto vylepšení je ale právě vykreslování obdélníků. Vykreslování se děje často a použitá implementace tohoto vylepšení při grafických změnách v hlavním okně KernelSharku často i přes vytvořené filtrování obdélník vykreslí. Při častých změnách pak nastává problikávání obdélníků, ačkoliv by vůbec kresleny být neměly. Oprava byla označena za rozšíření, jelikož objem práce k tomuto dodatku byl odhadnut na příliš velký.



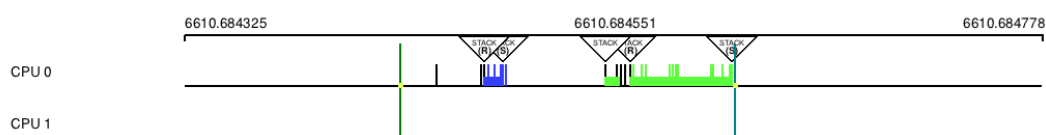
**Obrázek 10.1** Stacklook tlačítko využívá barvu, kterou KernelShark udělil procesu.



**Obrázek 10.2** Při přjetí kurzorem myši se vlevo nahoře informační řádek změní.



**Obrázek 10.3** Ne všechny obdélníky mezi záznamy by se měly vykreslovat, některé tvoří iluzi opravdové práce.



**Obrázek 10.4** Vylepšení donutí některé obdélníky k tomu, aby nebyli nakresleny.

# Závěr

Cíle/úvod této kapitoly...

## 10.6 Shrnutí práce

# Seznam obrázků

5.1	Zvýraznění změn této modifikace viditelných v GUI . . . . .	25
10.1	Stacklook tlačítko využívá barvu, kterou KernelShark udělil procesu.	43
10.2	Při přejetí kurzorem myši se vlevo nahoře informační řádek změní.	43
10.3	Ne všechny obdélníky mezi záznamy by se měly vykreslovat, některé tvoří iluzi opravdové práce. . . . .	43
10.4	Vylepšení donutí některé obdélníky k tomu, aby nebyli nakresleny.	43

# Seznam tabulek

# Seznam použitých zkratek

# A Přílohy

## A.1 První příloha