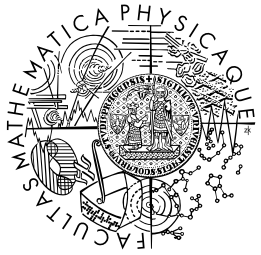


MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

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

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 nejefektivnější. 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 uspání 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 práci 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

ignorujme typos, ale ne deprecated warnings (UPDATE CBOX STATES)
Ne-topologie - ale proč by ne? - SRP vs využití
Nehezké 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 intrukcemi 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. nebýt 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 spojena 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 funkcionalita KernelSharku, tzv. Record okénko, dovoluje spustit program `trace-cmd record` pro nastartování trasování běžícího systému skrze GUI. Ačkoliv to není ihned zřejmé, tato funkcionalita 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. Také 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. Nakonec vybereme podporované události, tj. události, pro které plugin musí fungovat. Těmi 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.

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 zaškrtávací 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*.

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.

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 zjstit, jak dlouho byl proces nečinný, než byl probuzen, nezbývá 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 obdělníky 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.

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*.

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 zaškrtávacího 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 myší 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 idenitifikovat vlastnící proces události, jejíž záznam zásobníku si budeme chtít zobrazit. Navíc, pokud využijeme barvy, které používá KernelShark, pak se využité 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.

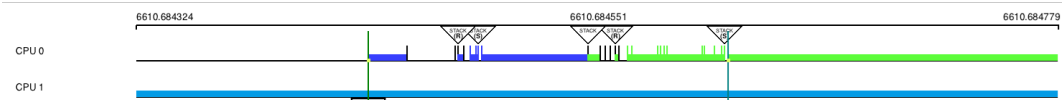
Možnost měnit text v informačním řádku

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 řáku 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. Příklad špatného zobrazování je na obrázku 4.1. 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 obdélníku spojených právě s nimi.



Obrázek 4.1 Ne všechny obdélníky mezi záznamy by se měly vykreslovat, některé tvoří iluzi opravdové práce.

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 konfigurující zachycování. Do jejích datových členů přidáme zašktávací 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 zašktnutého 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 zaškrtávací 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 zaškrtávací 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.

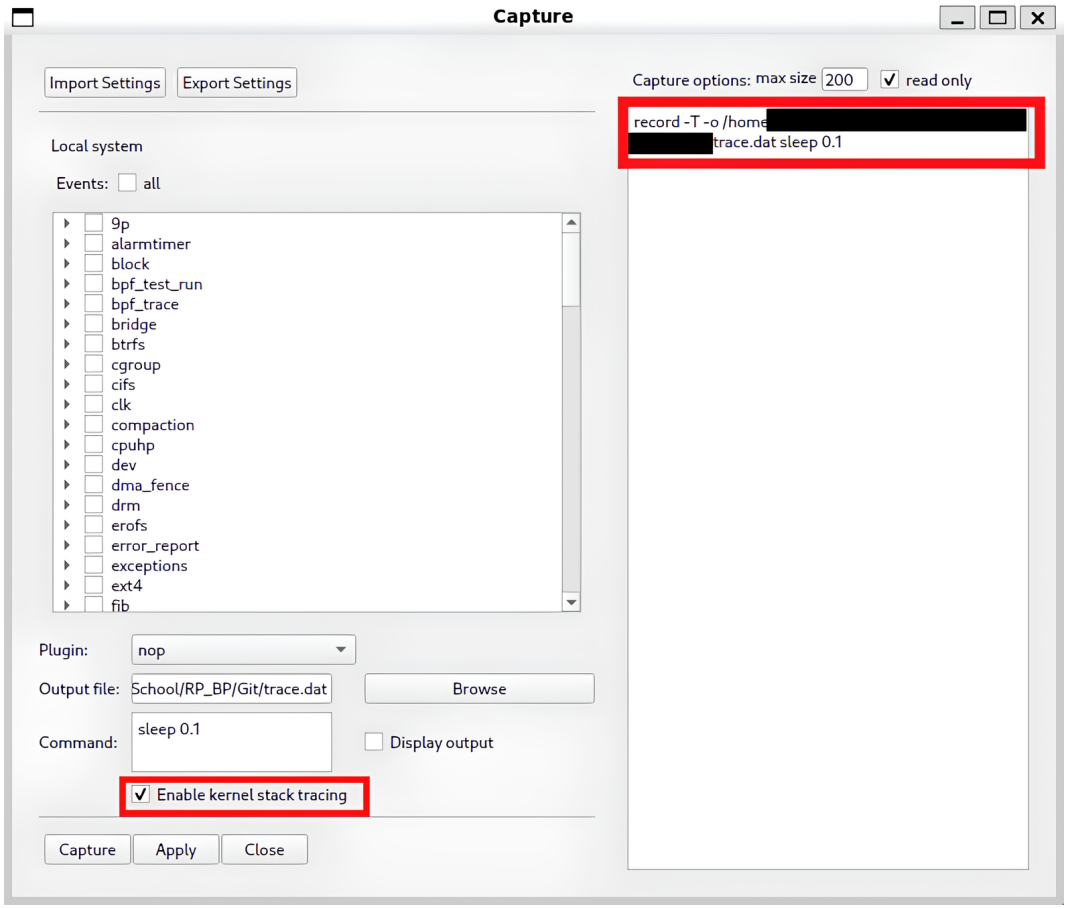
5.5 Rozšíření

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 zaškrtávacího 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

- 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 myší, 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.
- Plugin bude možné používat i pro KernelShark bez modifikací vytvořených jako součást této práce.

Poslední bod existuje pro případné uživatele, kteří nechtějí používat modifikovaný KernelShark, dokud nejsou modifikace součástí oficiálního vydání. Plugin se tak může využívat i v úplné izolaci od ostatních změn z této práce, což souhlasí s obecnými požadavky na pluginy. Ostatní cíle vychází z kapitoly *Obecná analýza a stanovení požadavků*. Jako samozřejmost se bere technická a uživatelská dokumentace a splnění obecných požadavků na pluginy.

6.2 Analýza

Tato sekce se pokusí zachytit postup, kterým se dostaneme k implementaci řešení. Jedná se o techničtější analýzu než analýza z kapitoly *Obecná analýza a stanovení požadavků*.

Zamysleme se nyní nad tím, jak bychom vytvořili plugin pro KernelShark s našimi cíli. Nejlepším začátkem by bylo nechat se inspirovat oficálními 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, které 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. Handlery jsou buď implementovány v C++, nebo, hlavně u jednoduchých handlerů událostí, ještě v 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 struktury 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.

Z našich cílů lze celkem přímo vymezit moduly, ze kterých se bude plugin skládat: *Propojující modul*, *Konfigurace*, *Tlačítka*, *Detailní pohledy*.

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 handlery. V „C části“ zároveň inicializujeme pluginový kontext. Tato část bude mít za úkol tento kontext i správně odstranit. V „C++ části“ definujeme handlery pro kreslení a přístup k hlavnímu oknu. Kreslící 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. Tento modul bude také obsahovat podporované události a bude je sbírat do dat v kontextu pluginu.

Konfigurace bude rozdělená mezi GUI okno a datovou strukturu s konfiguračními daty, tzv. konfigurační objekt. 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 můžeme jednoduše uložit jako čísla, pravdivostní hodnoty, textové řetězce a datovou strukturu pro barvy od KernelSharku. Aby bylo možné konfiguraci měnit pouze skrze konfigurační okno, tyto prvky schováme pomocí modifikátorů viditelnosti. Konfigurační objekt pak využije C++ mechanismus „zpřátelení“ přes

klíčové slovo `friend` a označí konfigurační okno za svého přítele. Konfigurace pluginů KernelSharku nejsou persistentní, tak dodáme i nějaké výchozí hodnoty. Kvůli zobrazování zásobníku v informačním řádku, přidáme ke každému nastavení specifickému pro událost i offset od vrchu zásobníku. Tak budeme moci stanovit oblast zásobníku, ve které se většinou nachází ty nejzajímavější části zásobníku pro danou událost.

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 zaškrtávacího tlačítka a dalšího spinboxu. Mimo požadavky navíc dodáme zaškrtávací tlačítko pro barvení tlačítek barvami jejich procesů.

Modul tlačítek se bude hlavně skládat z třídy pro tlačítka. Ta by měla být v grafu jasně viditelná a ihned rozpoznatelná uživatelem. Tvar tlačítek by měl nějak ukazovat, kterému záznamu patří. Zde je několik možností a tvarů na výběr, nicméně nejjednodušším tvarem bude trojúhelník. Jeden z vrcholů bude směřovat dolů, ostatní dva vrcholy budou nad tímto vrcholem. Rozpoznatelnost těchto tlačítek nakonec zajistíme přidáním nápisu `STACK` do každého z nich. Reakci na dvojité kliknutí a přejetí kurzorem myši implementujeme jako to dělají ostatní tvary definované KernelSharkem. U přejetí využijeme naše modifikaci na reakci na přejetí kurzorem myši a modifikaci na změnu informačního řádku, přesně jak říká jeden z cílů. Speciálně pro sched_switch budou tlačítka ještě obsahovat jedno písmeno symbolizující předchozí stav procesu před přepnutím. Takto nebude nutné se podívat do detailního pohledu, bude stačit se dívat do grafu. Nicméně uživatel bude muset znát zkratky pro možné předchozí stavy. Ty jsou součástí dokumentace Linuxu. [TODO: Sem se asi hodí zdroj.] Barva tlačítek bude určena hodnotami v konfiguračním objektu. Dodáme i možnost používat barvy procesů jako barvy tlačítek, to nám zajistí modifikace *Get Colors*. Aby nemohlo více tlačítek splývat díky stejné barvě, tlačítka budou mít obrys, jehož barva bude také dána konfigurací a bude oddělená od (vnitřní) barvy tlačítka.

Modul detailních pohledů bude hlavně obsahovat třídu pro okna těchto pohledů. V okně pak zobrazíme záznam zásobníku ve formě seznamu. Tento formát se hodí na jednoduché zvýraznění položky jedním kliknutím. Dodáme ale i možnost si data zobrazit jako prostý text - tento formát je naopak lepší když se kopírují data po jných částech než po položkách. Na vrchol zásobníku dodáme značku `top`, aby bylo jasné, kde se vrch zásobníku nachází. Jeden z cílů nám ještě udává nutnost zobrazit název procesu a nějaké bližší informace o události. Pro sched_switch zde můžeme udat přechozí stav procesu před přepnutím, například zdali byl proces již ve stavu zombie, nebo zdali začal čekat na nějaká data a nebo zdali prostě běžel a byl preemptivně přepnut. Předchozí stav zapíšeme jak zkratkou, tak celým názvem (například „S - sleep“, nebo „Z - zombie“). Pro sched_waking pouze napíšeme, že byl proces probuzen. Nakonec se zamyslíme nad tím, jak dlouho budou okna žít a kolik jich může být najednou pro jednu událost. Naše pohledová okna můžeme vytvořit jako podřízená oknu hlavnímu, tedy pokud je ukončeno hlavní okno,

budou ukončena i jeho podřízená okna. Toto zajistí Qt framework. Počet oken pro jednu událost nebudeme omezovat, nic se tím nerozbije a implementace bude jednodušší. Tato rozhodnutí by měla mít ten efekt, že uživatel bude schopen stream, nazvěme jej A, pak bude schopen začít zkoumat stream jiný, nazvěme tento B. Okna otevřená ve streamu A zůstanou otevřená i po přepnutí do streamu B, do té doby dokud je uživatel sám nezavře, nebo dokud uživatel nezavře hlavní okno. Lze tak porovnávat zásobníky z různých běhů trasování.

Na získání informací o předchozím stavu ještě dodáme mini-modul *Předchozí stavy*. Nutně bude muset být schopen vytáhnout data o předchozím stavu ze záznamu události přepnutí v KernelSharku. Tato data pak bud muset umět dodat pluginu buď jako zkratka pro typ stavu (tlačítka a detailní pohledy), nebo jako celý název (detailní pohledy).

Postupujme dále, podívejme se na informační řádek. Informační řádek nabízí pouze pět míst pro text. Pokud bychom museli zobrazovat konec zásobníku, nebo i prvky za koncem (například kvůli vysokému offsetu v konfiguraci), v informačním řádku pak vyznačíme, že jsme na konci zásobníku a místa, kde by byly prvky za koncem označíme nějakou čárou, třeba mínusem. Pokud ale zobrazujeme oblast zásobníku a za ní zásobník pokračuje, označíme toto třemi tečkami v posledním místě pro text v informačním řádku. Nakonec, pro jasnou identifikaci procesu z jehož záznamu zásobníku čerpáme bude první místo v informačním řádku obsazeno názvem tohoto procesu. Prvky v informačním řádku tedy budou vždy v následujícím pořadí:

- Název procesu
- Věc na zásobníku na offsetu X od vrcholu zásobníku, nebo `-`
- Věc na offsetu X+1, nebo `-`
- Věc na offsetu X+2, nebo `-`
- `...`, nebo `(End of stack)`

Informační řádek bude vždy schopen zobrazit nejvýše tři prvky ze zásobníku.

Nakonec se zamyslíme, jak získat data záznamů zásobníku kernelu. Tlačítka jsou nad záznamy, po kterých následuje záznam zásobníku. Tyto záznamy se vždy vytvoří na stejném CPU jako událost jim předcházející. Během načítání dat ale ještě nemáme události seřazené, ovšem při prvním kreslení už ano. Navíc neplatí, že všechny soubory trasovacích dat musí obsahovat záznamy zásobníku, lze systém trasovat i bez sbírání zásobníku. V tom případě nemá smysl, aby plugin cokoli kreslil. Do kontextu pluginu dodáme dvě proměnné, jednu jako indikátor „hledali jsme záznam trasování zásobníku kernelu“ a druhou jako „záznam trasování zásobníku existuje“, přičemž obě začnou na pravdivostní hodnotě „nepravda“. Při prvním kreslení prohledáme následníky všech záznamů námi podporovaných událostí. Pokud alespoň jeden záznam má za následníka záznam zásobníku, nastavíme druhou proměnnou v kontextu na „pravda“. Ukazatel na tento záznam si pak poznamenáme i u záznamu předcházející události. Po prohledání všech podporovaných záznamů nastavíme i první proměnnou na „pravdu“. Tak si zajistíme jediný chod při hledání těchto dat. Ačkoliv prohledávání může trvat dlouho, stane se tak pouze jednou a časová cena prohledávání tak není příliš výrazná.

Plugin pro nemodifikovaný KernelShark

Každá část implementace využívající nějakou z našich modifikací bude obalena podmínkovým `ifndef` makrem s názvem `_UNMODIFIED_KSHARK`. Toto makro bude aktivováno pokud uživatel při sestavování tuto proměnnou definuje argumentem pro CMake, více v uživatelské dokumentaci v sekci o instalaci. Pokud je makro definováno, pak zvolí alternativní implementaci z `else` větve, která nevyužívá žádnou z modifikací; jinak kód nebude součástí kompilace.

6.3 Vývojová dokumentace

Dokumentace pluginu

Dokumentace je napsána pro nástroj Doxygen. Dokumentovala se každá funkce i proměnná, nicméně vygenerovaná dokumentace obsahuje jenom prvky veřejné, pro skutečné implementační detaily je tedy doporučeno se podívat do zdrojového kódu. Dokumentace navíc obsahuje hlavní stránku a stránku s nástinem návrhu.

Struktura projektového adresáře

Adresář pluginu obsahuje další adresáře. Adresář „src“ pro zdrojový kód a adresář „doc“ pro dokumentaci uživatelskou a dokumentaci technickou. Očekává se, že na této úrovni jsou i adresáře pro sestavení. Na stejné úrovni bude žije 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 jako to dělá KernelShark.

Přehled modulů pluginu

- *Propojující modul* - Modul s kódem propojujícím KernelShark a plugin. Obsahem je hlavně kontext pluginu, funkce kontextu, handlery a implementačně pomocné funkce. Součástí tohoto modulu je část s C kódem a implementační část v C++ prvků z 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. Soubory modulu jsou *stacklook.h/c* a *Stacklook.cpp*.

- *Konfigurace* - Modul se skládá ze dvou tříd, konfigurační objekt a konfigurační okénko. Konfigurační objekt obsahuje konfigurační data, která plugin zrovna využívá a je navržen jako singleton. Konfigurační okénko představuje GUI element, kterým se data v konfiguračním objektu manipulují. Soubory modulu jsou *SlConfig.hpp/cpp*.

- *Tlačítka* - Tlačítka jsou vykreslována v grafu nad záznamy podporovaných událostí. Dokáží reagovat na dvojité kliknutí myši, čímž vyvolají detailní pohled na zaznamenaný zásobník, nebo přejetí nad nimi, kdy zobrazí část zásobníku v informačním řádku. Soubory modulu jsou *SlButton.hpp/cpp*.

- *Detailní pohledy* - Detailní pohledy jsou grafická okna zobrazující zaznamenaný kernel zásobník, informace o události, která záznamu zásobníku předcházela, a informace o procesu vlastnící tuto událost. Soubory modulu jsou *SlDetailedView.hpp/cpp*.
- *Předchozí stavy* - Mini-modul, dodává data tlačítkům a detailním pohledům o stavu procesu před přepnutím kontextu na CPU. Soubory modulu jsou *SlPrevState.hpp/cpp*.

6.4 Uživatelská dokumentace

Tato sekce popíše jak instalovat a používat plugin Stacklook v KernelSharku a co od něj během běhu očekávat, či na co si dát pozor. Obrázek 6.1 ukazuje fungující plugin v akci.

6.4.1 Jak sestavit a instalovat Stacklook

Kompatibilita

- CMake verze alespoň 3.1.2.
- Pokud chcete plugin využívající modifikovaný KernelShark, pak je nutná verze alespoň 2.4.0-couplebreak. Pokud chcete plugin pro nemodifikovaný KernelShark, pak je nutná verze alespoň 2.3.2.
- Závislosti KernelSharku (naleznete v README repozitáře KernelSharku), zejména Qt6.
- Doxygen na technickou dokumentaci.

Sestavení a instalace pouze pluginu

1. V terminálu nastavte pracovní adresář na složku `build` (pokud ještě neexistuje, pak ji nejlépe vytvořte v kořenovém adresáři projektu).
2. Spusťte příkaz `cmake ...` Pokud hlavní soubor `CMakeLists.txt` není v nadřazené složce, předejte programu CMake platnou cestu k němu.
    - Používáte-li verzi KernelSharku bez modifikací, přidejte do příkazu argument `-D_UNMODIFIED_KSHARK`. Sestavení pro nemodifikovaný KernelShark odstraňuje tyto funkce:
        - Tlačítka mohou být stejné barvy, jako procesy, jimž patří záznamy s tlačítky.
        - Přejetí myši zobrazí část zásobníku v informačním řádku KernelSharku.
    - Pokud chcete generovat dokumentaci pomocí Doxygenu, přidejte do příkazu argument `-D_DOXYGEN_DOC=1`.
    - Výchozí typ sestavení je `RelWithDebInfo`. Chcete-li ho změnit (např. na `Release`), použijte argument `-DCMAKE_BUILD_TYPE=Release`.

- Pokud se soubory Qt6 nenacházejí ve `/usr/include/qt6`, použijte argument `-D_QT6_INCLUDE_DIR=[PATH]`, kde `[PATH]` nahradíte cestou k souborům Qt6.
  - Pokyny pro sestavení předpokládají, že zadaný adresář má stejnou vnitřní strukturu jako výchozí možnost (tj. obsahuje složky QtCore, QtWidgets apod.).
- Pokud se zdrojové soubory KernelSharku nenachází v `../KS_fork/src`, použijte argument `-D_KS_INCLUDE_DIR=[PATH]`, kde `[PATH]` nahradíte cestou ke zdrojovým souborům KernelSharku.
- Pokud se sdílené knihovny KernelSharku (`.so` soubory) nenachází ve `/usr/local/lib64`, použijte argument `-D_KS_SHARED_LIBS_DIR=[PATH]`, kde `[PATH]` nahradíte cestou k sdíleným knihovnám KernelSharku.

3. Ve složce `build` spusťte příkaz `make`.
   - Pokud je třeba sestavit jen část pluginu, například pouze dokumentaci, můžete vybrat konkrétní cíl.
   - Pouhé spuštění `make` vytvoří: *plugin* (cíl `stacklook`), *symlink* na sdílený objekt pluginu (cíl `stacklook_symlink`) a případně *Doxygen dokumentaci* (cíl `docs`), pokud tak bylo specifikováno v předchozím kroku.
4. (*Instalace*): Nahrajte plugin do KernelSharku, buď přes GUI, nebo při spouštění přes CLI s argumentem `-p` a cestou k symlinku nebo přímo k sdílenému objektu.
   - *DŮLEŽITÉ*: Vždy nainstalujte/nahrajte plugin před načtením relace, ve které byl aktivní! Jinak může dojít k neúplnému načtení konfiguračního rozhraní nebo k pádu celého programu.

K odstranění vytvořených binárních souborů použijte `make clean`.

Sestavení a instalace pomocí KernelSharku

1. Ujistěte se, že všechny zdrojové soubory (`.c`, `.cpp`, `.h`) Stacklooku se nacházejí ve složce `src/plugins` v adresáři projektu KernelShark.
2. Zkontrolujte, že soubor `CMakeLists.txt` v této podsložce obsahuje instrukce pro sestavení pluginu (inspirovat se můžete podle jiných pluginů pro GUI). Pokud chcete sestavovat pro nemodifikovaný KernelShark, upravte tomu odpovídajícím způsobem build skript.
3. Sestavte KernelShark (pluginy se sestavují automaticky). Lze sestavit i pouze plugin, pokud jste již předtím vytvořili instrukce sestavení.
4. (*Instalace*): Spusťte KernelShark. Pluginy sestavené tímto způsobem se načítají automaticky. Pokud by se z nějakého důvodu nenačetly, najděte sdílený objekt stejně jako u ostatních oficiálních pluginů, opět buď přes GUI, nebo přes CLI.

VAROVÁNÍ - načítání více verzí pluginu

Máte-li dvě nebo více sestavených verzí pluginu, *NE*načítejte je současně do KernelSharku. Pokud to uděláte, *DOJDE K PÁDU PROGRAMU*. Používejte vždy jen jednu z verzí, *NIKDY OBOJE NAJEDNOU*.

6.4.2 Jak zapnout/vypnout Stacklook

Zapnutí pluginu je velmi jednoduché. Stačí spustit KernelShark a přejít na položku v panelu nástrojů `Tools > Manage Plotting plugins`. Pokud byl plugin načten přes příkazový řádek, zobrazí se v seznamu pluginů jako zaškrtnuté políčko se svým názvem. Pokud ne, lze plugin dohledat pomocí tlačítka `Tools > Add plugin` - stačí nalézt symlink, ale je možné vybrat i samotný soubor sdíleného objektu. Jak je vidět, plugin využívá standardní mechanismus načítání pluginů v KernelSharku. Obrázek 6.2 ukazuje GUI pro zapínání a vypínání pluginů.

6.4.3 Jak používat Stacklook

Konfigurace

Konfigurace pluginu může být provedena kdykoliv, i před načtením jakýchkoliv trasovacích dat. Pro otevření konfiguračního okna (viz obrázek 6.3) stačí v hlavním okně zvolit `Tools > Stacklook Configuration`. Vždy může být otevřeno jen jedno konfigurační okno.

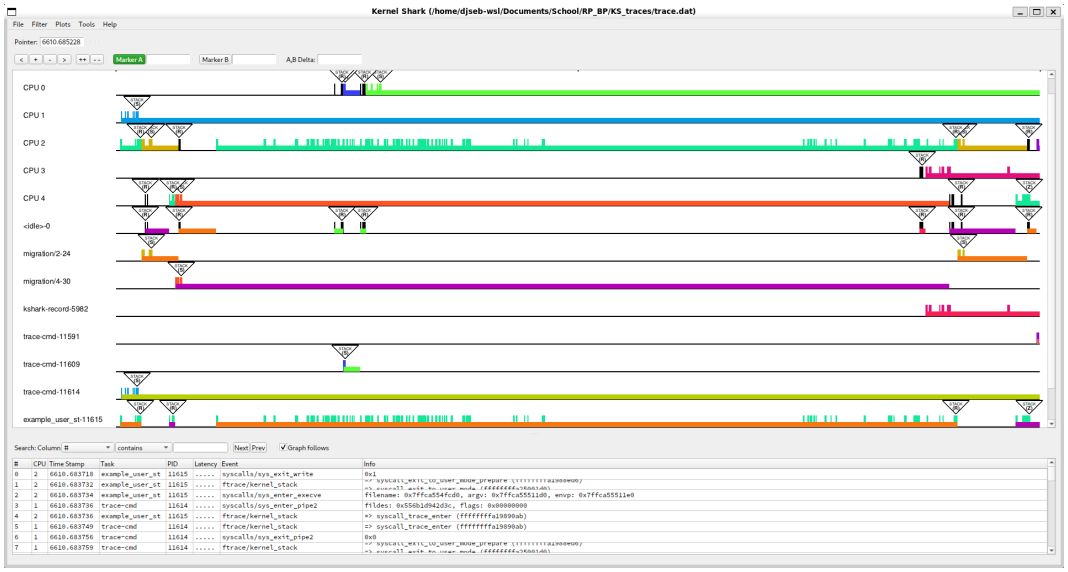
Používáte-li verzi pluginu pro nemodifikovaný KernelShark, bude v konfiguračním okně chybět zaškrtávací políčko pro barvení tlačítek barvami procesů a nastavení offsetu, použitého při zobrazování prvků zásobníku v informačním řádku, viz obrázek 6.4.

Nyní si popíšeme jednotlivé možnosti konfigurace a jak je ovládat. Seřazeno shora dolů:

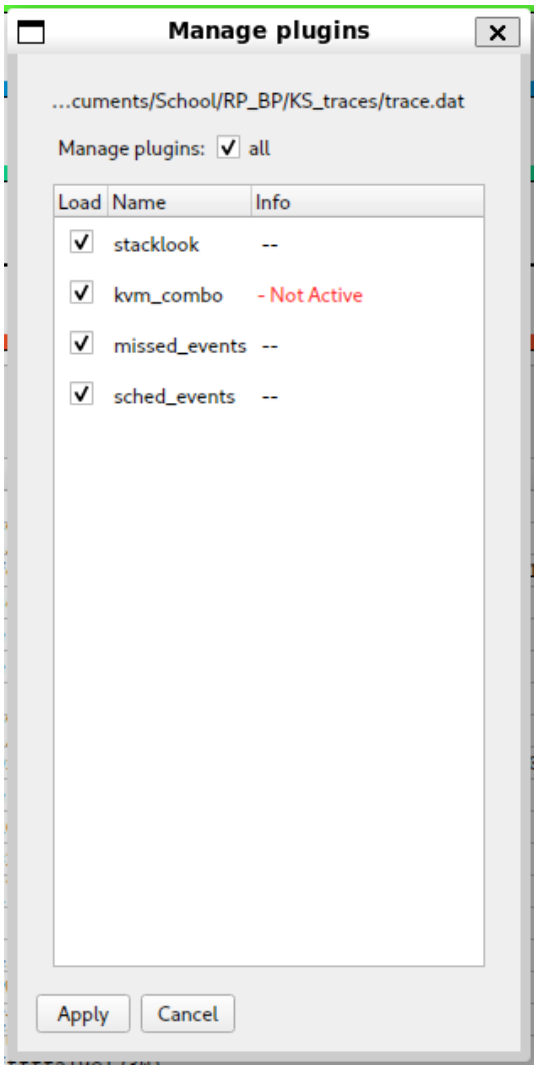
- *Limit záznamů v histogramu* - snížením této hodnoty omezíte, kdy se Stacklook aktivuje. Aktivace nastane pouze tehdy, pokud je viditelný počet záznamů menší nebo roven této hodnotě. Čím nižší číslo, tím větší přiblížení je třeba k aktivaci Stacklooku. Minimální hodnota je 0, maximální 1 000 000 000 (jedna miliarda), tato horní mez však bude pravděpodobně zřídka využita. Výchozí hodnota je 10 000 (deset tisíc).

- *Použít barvy procesů pro tlačítka Stacklooku* - zaškrtnutím tohoto políčka (pokud je k dispozici) bude výplň tlačítek Stacklooku zabarvena barvou procesu, ke kterému patřila událost, pro niž Stacklook našel záznam zásobníku. Pokud políčko necháte nezaškrtnuté, použijí se barvy v konfiguraci Stacklooku (výchozí barvy na obrázku 6.5). Tato volba je ve výchozím stavu vypnutá.

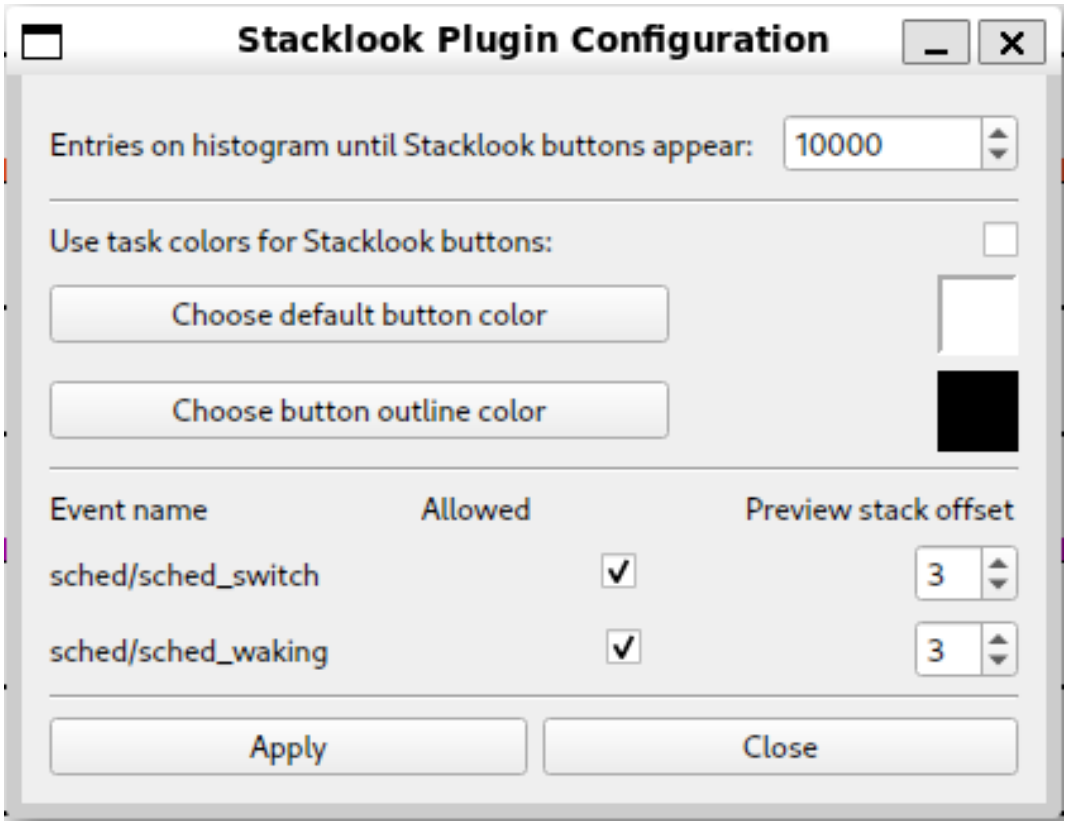
- *Barvy tlačítek Stacklooku a jejich obrysů* - tlačítka `Choose` otevřou dialog pro výběr barev, kde lze nastavit barvu výplně tlačítka nebo jeho obrys. Tato nastavení mají účinek pouze tehdy, pokud není dostupná nebo není aktivní volba barev podle procesů. Vedle tlačítek je zobrazen náhled aktuálně vybraných barev. Výchozí nastavení je bílá výplň a černý obrys. Příklad jiných barev si lze prohlédnout na obrázku 6.7.



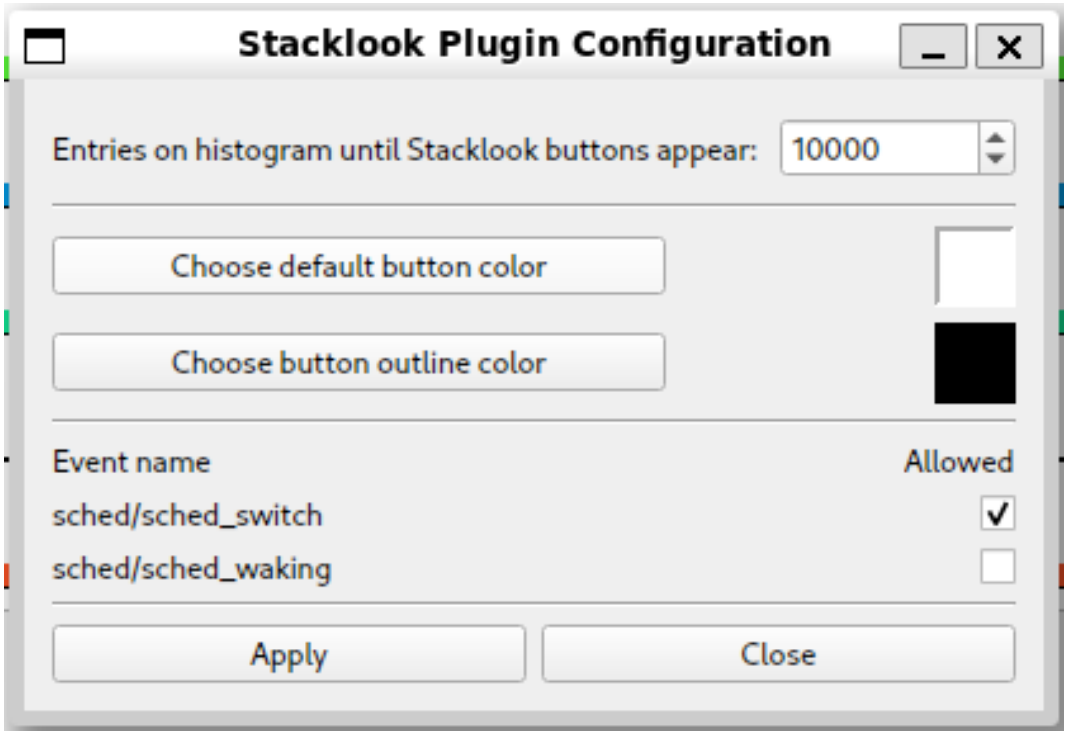
Obrázek 6.1 Fungující Stacklook



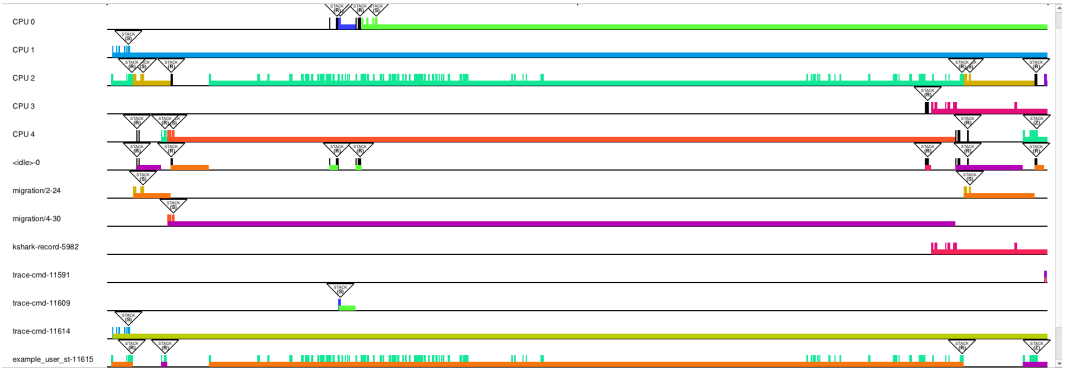
Obrázek 6.2 Okénko se správou pluginů



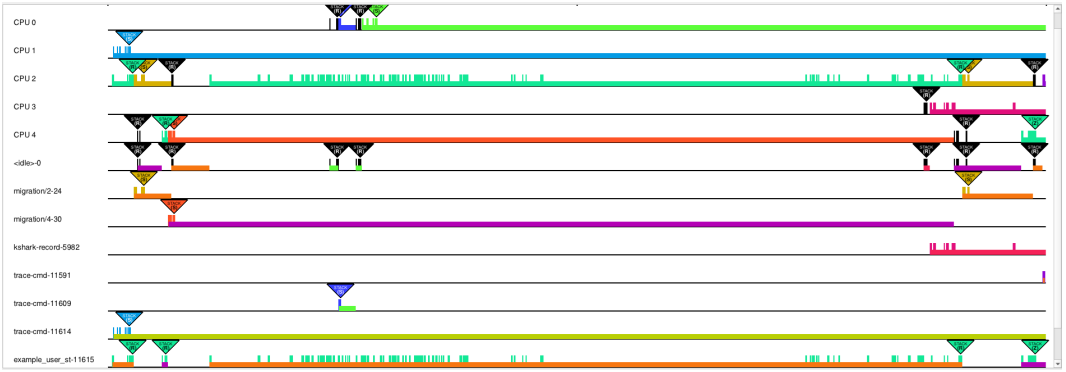
Obrázek 6.3 Konfigurační okno Stacklooku pro modifikovaný KernelShark



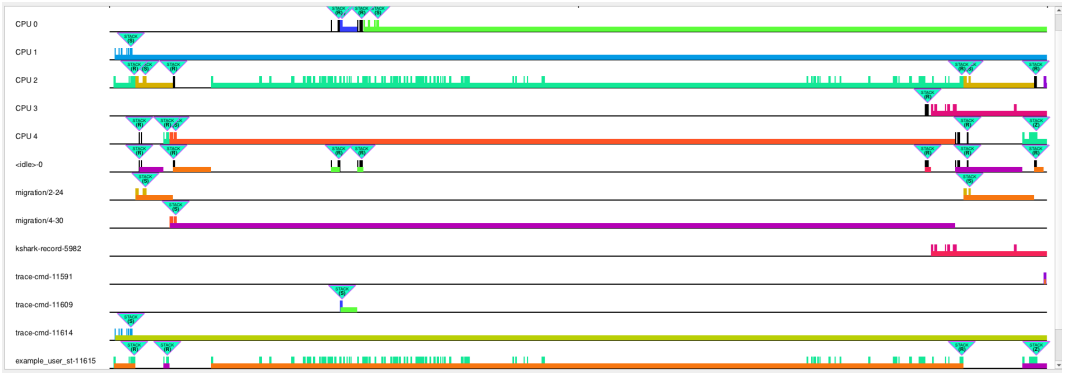
Obrázek 6.4 Konfigurační okno Stacklooku pro nemodifikovaný KernelShark



Obrázek 6.5 Tlačítka s výchozími barvami



Obrázek 6.6 Tlačítka využívající barvy procesů



Obrázek 6.7 Tlačítka s konfigurovanými barvami

- *Nastavení pro jednotlivé události* - Každá událost má své vlastní zaškrtávací políčko, kterým lze Stacklook zapnout nebo vypnout pro daný typ záznamu, a (pokud je k dispozici) číselník s offsetem do zásobníku kernelu, který slouží k určení „nejzajímavější“ oblasti v zásobníku pro danou událost. Maximální hodnota offsetu je 100 000 000 (sto milionů), minimální 0. I zde je nepravděpodobné, že by bylo třeba využít maximální hodnotu. Ve výchozím stavu jsou všechny události povoleny (zaškrtnuty) a offset je nastaven na 3.

  – Připomenutí - číselník offsetu se nezobrazí, pokud je použit nemodifikovaný KernelShark.

Tlačítko `Apply` uloží provedené změny a zavře konfigurační okno - pokud toto tlačítko nebude stisknuto, změny se neprojeví. V ovládacích prvcích konfiguračního okna se zobrazují pouze aktuálně platné hodnoty konfigurace, konfigurační okno si po zavření neaplikované změny nepamatuje. Tlačítko `Close` i tlačítko s křížkem v pravém horním rohu okna změny zahodí a okno zavřou.

U Stacklooku nelze konfigurovat:

- Podporované události - plugin momentálně podporuje pouze události sched/sched_switch a sched/sched_waking.
- Text v oknech detailních pohledů.
- Text tlačítek.
- Velikost tlačítek.
- Pozice tlačítek.

Konfigurace není persistentní. Její momentální stav se nikam neukládá, ani do relací.

V grafu

Po načtení (a případné konfiguraci) pluginu přibližte zobrazení tak, aby bylo v grafu viditelných méně záznamů, než je nastavený limit. Nad každým podporovaným záznamem se objeví tlačítko - buď ve výchozí barvě z konfigurace, nebo, pokud používáte upravený KernelShark a máte zapnutou příslušnou část nastavení, bude tlačítko obarveno podle procesu. Používání barev procesů je plně kompatibilní s barevným posuvníkem KernelSharku.

Plugin nezobrazí tlačítka nad nepodporovanými událostmi nebo pokud podporovaná událost nenajde záznam zásobníku, ze kterého by bylo možné čerpat data.

Pokud používáte modifikovaný KernelShark, přejeďte kurzorem nad libovolné tlačítko a informační řádek KernelSharku. Obsah řádku se změní a zobrazí:

- Název procesu jako první (nejlevější) položku.
- Položku v zásobníku kernelu na pozici danou konfigurovaným offsetem od vrcholu zásobníku.

- Položku v zásobníku kernelu následující po první.
- Položku v zásobníku kernelu následující po druhé.
- A nakonec buď tři tečky `...`, pokud stack obsahuje další položky, nebo zprávu `(End of stack)`, pokud už žádné další položky nezbývají.

Pohleďme na obrázek 6.8 s malou ukázkou této funkcionality. Součástí je také okno Stacklooku (o nich více níže) a záznam zásobníku kernelu v hlavním okně v seznamu všech událostí. Vše je takto uspořádáné, aby bylo zřejmé, že položky v informačním řádku skutečně pocházejí ze zásobníku (použitý offset byl výchozí, tedy 3). Červený kruh zvýrazňuje záznam, nad kterým právě kurzor myši přejíždí.

Offset lze nastavit i tak vysoko, že se v náhledu zobrazí pouze poslední jedna, dvě nebo tři položky, případně žádná. V takovém případě Stacklook zobrazí pouze mínus a zprávu `(End of stack)` (viz obrázek 6.9).

Po dvojkliku na tlačítko Stacklooku se otevře nové okno detailního pohledu, také nazývané okno Stacklooku. Poku čteme odshora, tak v okně nejprve vidíme, že si prohlížíme zásobník kernelu nějakého procesu. Níže je napsáno, zda byl proces probuzen (zobrazuje se pouze u událostí sched/sched_waking) nebo o jejím předchozím stavu (pouze u událostí sched/sched_switch). Následují dvě rádiová tlačítka a zobrazení zásobníku kernelu k dané události. Rádiová tlačítka přepínají mezi různými způsoby zobrazení zásobníku:

- Ve výchozím stavu je zobrazení ve formě surového textu, tedy zásobník je pouze řetězec s konci řádků. Toto je užitečné pro zkopírování zásobníku jako jednoho textu nebo pro zvýraznění konkrétní části položky v zásobníku.
- Alternativně lze zásobník zobrazit jako seznam, což umožňuje jednodušší a rychlejší (namísto dvojitého kliknutí stačí jedno) zvýraznění jednotlivých položek.

Pro jeden záznam může být otevřeno více oken, zároveň může být otevřeno více oken pro různé záznamy, v libovolné kombinaci.

Vše popsáno výše si můžete prohlédnout na obrázku 6.10.

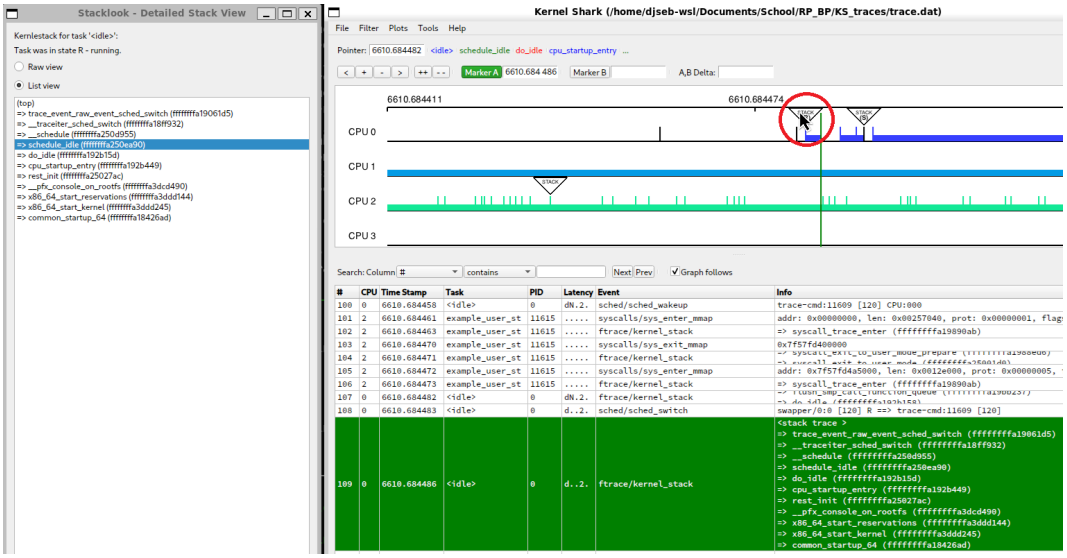
Okno lze zavřít přes tlačítko `Close` v dolní části okna, nebo pomocí tlačítka s křížkem v hlavičce okna. Všechna okna se zavřou, pokud bude zavřeno hlavní okno KernelSharku.

6.4.4 Bugy a chyby

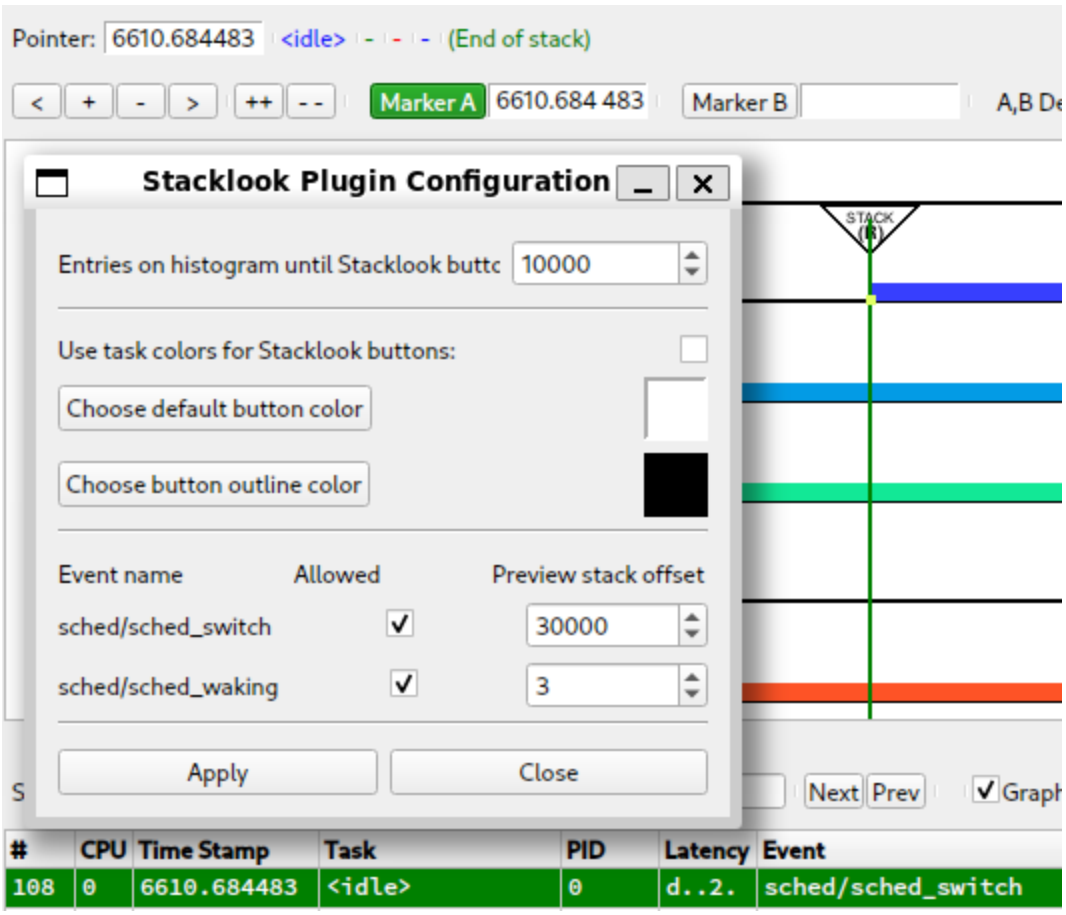
Pokud by se tlačítka Stacklook překrývala, bude tlačítko příslušící *starší* události vykresleno nad tlačítkem pro událost *pozdější*, avšak kliknutí nebo najetí myší na překryté tlačítko použije k interakci tlačítko pro *pozdější* událost. Podle autorových znalostí jde o interní chování KernelSharku, které nelze na straně pluginu opravit.

Načtení relace KernelSharku, kde byl Stacklook aktivní, bez předchozího načtení pluginu způsobí *segmentation fault* a *pád programu* při najetí myší na tlačítko Stacklook.

Pokud objevíte další problémy, kontaktujte autora přes e-mail *djsebofficial@gmail.com*



Obrázek 6.8 Reakce tlačítek na přejetí kurzorem myši



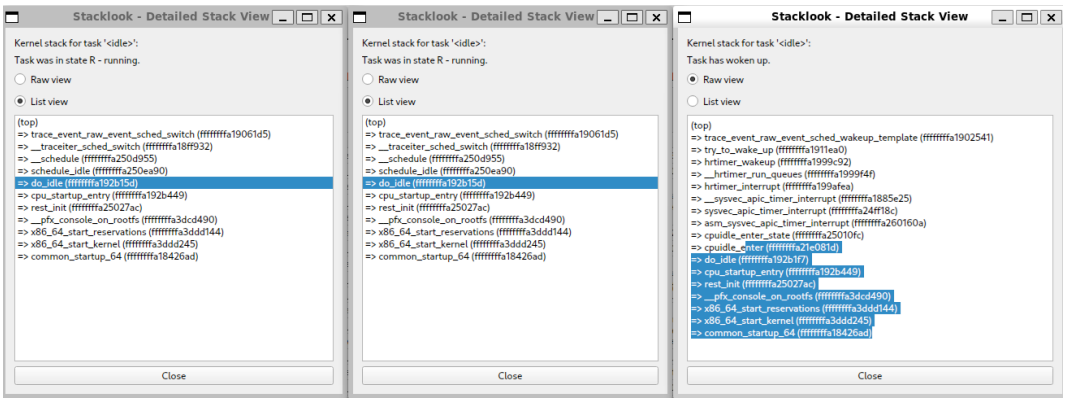
Obrázek 6.9 Chování informačního řádku při velkém offsetu do zásobníku

6.4.5 Doporučení

- Vždy načtěte Stacklook před načtením relace. Může to programu ušetřit nepříjemná překvapení.
- Neotvírejte stovky a stovky oken Stacklooku, pokud nechcete zbytečně zatížit paměť.
- Nedoporučuje se nastavovat příliš vysoký limit záznamů v histogramu v konfiguraci. Jinak by plugin mohl používat příliš mnoho paměti kvůli velkému množství naráz dostupných tlačítek Stacklooku.
- I když relace v KernelSharku fungují, jsou trochu nestabilní. Tento plugin se snaží jejich vnitřní logiku nenarušovat, ale varuje, že pokud plugin není načten předem, mohou nastat neočekávané problémy. Například načtení relace s aktivním pluginem nepřidá do menu `Tools` odpovídající položku pro vyvolání konfiguračního okna.

6.5 Rozšíření

Plugin lze rozšířit o další podporované události. Hlavními místy pro tato rozšíření je pak propojující modul, kde se podporované události sbírají a do kontextu se ukládají jejich identifikátory, a mapová konstanta s hodnotami informací specifických pro typ události, která je uvnitř funkci tlačítek, jež vytváří detailní pohledy.



Obrázek 6.10 Několik oken detailních pohledů představující svou strukturu

7 Couplebreak

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

7.1 Cíle

- 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.

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 popiskem 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 přidaných 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

- 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.

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

- 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.

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 zaškrtávacích políček

Nejpřímějším zlepšením byla aktualizace grafického kódu, aby využíval novější Qt API pro zaškrtávací 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 zaškrtávací políčka.

Změněné soubory: nejvyšší *CMakeLists.txt* KernelSharku (v KS_fork adresáři), *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 přejetí myší

Tato modifikace dodala všem potomkům třídy KsPlot::PlotObject, jednodušeji plot-objekty, veřejnou metodu pro reakci na přejetí myší a redefinovatelnou privátní virtuální metodu, kterou veřejná metoda volá pro viditelný objekt. Privátní metoda má výchozí definici prázdnou. Krom toho je dodána detekce přejetí přes plot-objekt v grafu a reakce na přejetí. 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 přejetí myší o zobrazení několika prvků zásobníku kernelu, viz obrázek 10.2.

10.5 NoBoxes

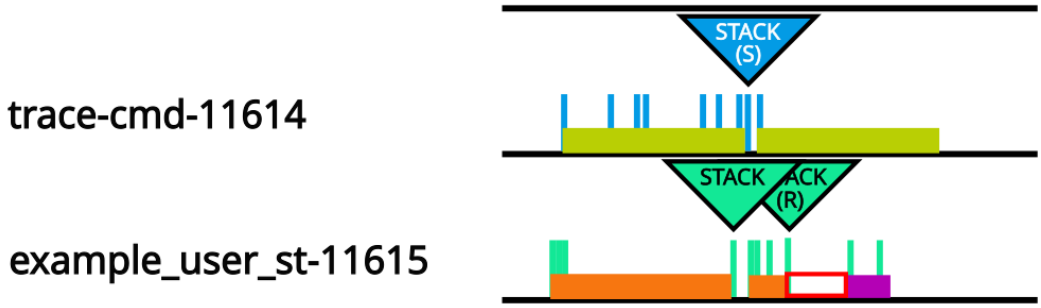
Vykreslování obdélníků se děje při každém vykreslení grafu. Ovlivnit obdélníky dokáže v této chvíli máloco. Pracuje se zde s biny, tedy sdruženími jednoho či více záznamů. V binech se pak lze spolehnout na málo, nejvíce na data viditelnosti, ve kterých mohou být nastaveny některé bity jako přepínače 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 obdélníčků. Pokud bin využije viditelnost záznamu, který má daný bit nastaven na 0, bude moci toto nastavení detekovat. Při vykreslování obdélníčků pak žádné obdélníčky nezačínají a nekončí u binů s tímto bitem vynulovaným.

Maska byla vložena 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.

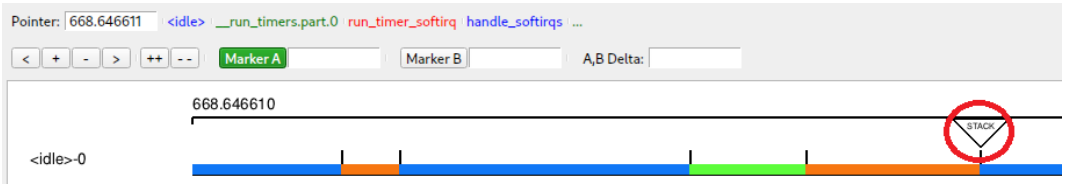
Na výběr záznamů, kterým se nastaví bit na 0, byl vytvořen plugin pro tuto modifikaci, nese název NoBoxes. V jeho kódu jsou zapsány události, na jejichž záznamy má být maska použita, plugin pak lze zapínat a vypínat jako každý jiný. Pokud plugin není načten, modifikace nemá na chod KernelSharku vliv.

Příklad fungujícího vylepšení je na obrázku 10.3. Oproti minulému obrázku (obrázek 4.1) ubylo několik obdélníků a graf nyní odpovídá skutečné práci.

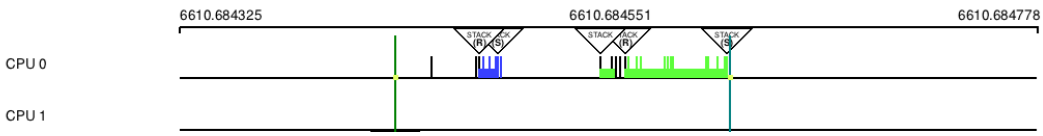
Problém tohoto vylepšení je ale právě vykreslování obdélníků. Vykreslování se děje často a použitá implementace tohoto vylepšení při grafických změnách v hlavním okně KernelSharku často i přes vytvořené filtrování obdélník vykreslí. Při častých změnách pak nastává problikávání obdélníků, ačkoliv by vůbec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



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



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



Obrázek 10.3 Vylepšení donutí některé obdélníky k tomu, aby nebyli nakresleny.

Závěr

Cíle/úvod této kapitoly...

10.6 Shrnutí práce

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek

A Přílohy

A.1 První příloha