Report z letného semestra

Úvod

Prvým cieľom v letnom semestri bolo zistiť, ktoré služby a knižnice najčastejšie spravujú audio na linuxových distribúciách. V súčasnej dobe sú to najmä **PipeWire,** prípadne trochu staršie **PulseAudio**. Kvôli maximálnej kompatibilite môjho programu s linuxovými distribúciami som sa rozhodol využiť PulseAudio. PipeWire je však spätne kompatibilný aj s PulseAudio.

PulseAudio

Zoznámil som sa teda s PulseAudio API, ktorá je zdokumentovaná tu.
PulseAudio využíva vlastný mainloop na spracovanie eventov, ktorý sa spúšťa volaniami *pa_threaded_mainloop_new()* a *pa_threaded_mainloop_start()*. Potom je potrebné vytvoriť a pripojiť tzv. **kontext** pomocou *pa_context_new()* a *pa_context_connect()*. Keďže môj program je multivláknový, je potrebné zavolať *pa_threaded_mainloop_lock()* pred spustením nejakej operácie a *pa_threaded_mainloop_unlock()* po ukončení, čím sa zamkne, resp. odomkne mutex. PulseAudio používa nasledovnú terminológiu: ***sink*** znamená výstupné zariadenie (napr. reproduktory), ***sink input*** je aplikácia, ktorá vydáva zvuk, ***index*** je číslo, ktorým je identifikovaný ***sink*** alebo ***sink input***.

Prehľad niektorých operácií, ktoré PulseAudio poskytuje:

- *pa_context_get_sink_info_list()* - získa kompletný zoznam zvukových výstupných zariadení
- *pa_context_get_sink_info_by_name()* - získa informácie ku konkrétnemu výstupnému zariadeniu
- *pa_context_get_sink_input_info_list()* - získa kompletný zoznam aplikácií, ktoré prehrávajú nejaký zvuk
- *pa_context_set_sink_volume_by_index()* - nastaví hlasitosť výstupného zariadenia
- *pa_context_set_sink_mute_by_index()* - stlmí výstupné zariadenie
- *pa_context_set_sink_input_volume()* - nastaví hlasitosť aplikácie
- *pa_context_set_sink_input_mute()* - stlmí aplikáciu

Program na začiatku zistí všetky výstupné zariadenia a aplikácie, ktoré prehrávajú zvuk. Uloží ich do std::map, kde kľúčom je ***index*** a hodnotou je štruktúra obsahujúca niekoľko informácií vrátane mena aplikácie, mena výstupného zariadenia, aktuálnej úrovne hlasitosti a pod. Mapa umožňuje vkladanie a vyhľadávanie v logaritmickom čase, čo je výhodné, keďže na manipuláciu so ***sink*** a ***sink input*** je potrebný ***index***. ***Index*** nezačína od nuly, je to prakticky náhodné číslo, ktoré priradí systém.

Program potom načíta konfiguráciu so súboru (ak nejaká existuje) a podľa nej priradí ***sinks*** a ***sink inputs*** do jedného zo šiestich dostupných kanálov na ovládači.
Program tiež sleduje udalosti (***events***), ktoré vytvára PulseAudio, ako napríklad pridanie nového ***sink input-u*** alebo zmena defaultného ***sink-u*** a korektne ich spracuje.

IO

Na vykonávanie IO operácií používam v programe samostatné vlákno, ktoré pomocou systémového volania *poll()* monitoruje niekoľko file deskriptorov. Sú to: sériový port, gui server socket, gui client socket a udev monitor. Na komunikáciu s gui slúži UNIX domain socket (typ AF_UNIX). Jeden socket čaká na pripojenie zo strany gui programu (volanie *listen()* a *accept()*). Po pripojení GUI vráti nový socket, ktorý je teda tiež monitorovaný. Posledný file deskriptor predstavuje monitorovanie udalostí o pripojených a odpojených zariadeniach (viac nižšie). Na komunikáciu medzi vláknami som vytvoril dátovú štruktúru - frontu s vlastným *mutexom* a mechanizmom *wait* a *signal*, ktorý poskytuje knižnica **pthread.h**. Pomocou tejto fronty môžu ostatné vlákna posielať hlavnému vláknu správy, napr. keď prídu dáta na sériový port, na gui socket, alebo keď je pripojené nové USB zariadenie.

Signály, udev

Na spracovanie signálov (Linux Signals) využívam samostatné vlákno, ktoré čaká na volaní *sigwait()*. Zachytáva SIGINT a SIGTERM, pri ich zachytení odovzdá hlavnému vláknu informáciu, že má skončiť.
Na sledovanie udalostí spojených s io-zariadeniami na Linuxe slúži knižnica **udev**. Umožňuje môjmu programu sledovať, či bolo pripojené, resp. odpojené nejaké zariadenie. Program konkrétne sleduje USB *tty* zariadenia a reaguje na tieto udalosti.

System-Tray ikony

Ku koncu semestra som zisťoval, aké knižnice umožňujú na Linuxe zobrazenie system-tray ikony. Existuje ich tiež niekoľko v závislosti od konkrétnej distribúcie. Napr. Ubuntu používa knižnicu **libayatana-appindicator**. Zistil som, ako sa s ňou pracuje a implementoval som aj jednoduché kontextové menu. Nechcel som však aby bol program viazaný na túto konkrétnu knižnicu, preto som sa rozhodol, že jej implementáciu zahrniem do dynamickej knižnice (prípona .so, Shared Object), ktorá bude samostatne skompilovateľná. Hlavný program sa potom pokúsi nájsť túto dynamickú knižnicu a načítať ju. Na toto sa využíva rozhranie **dlfcn.h**. Poskytuje funkcie *dlopen()*, *dlclose()*, *dlsym()*. Neskôr som sa ešte dozvedel o existencii **Qt** frameworku, ktorý využívajú hlavne systémy s KDE Plasma, avšak je podporovaný aj prostredím GNOME na Ubuntu. Implementoval som teda aj verziu dynamickej knižnice pomocou tohto frameworku. Používateľ sa vďaka tomu môže rozhodnúť,

ktorú verziu skompiluje, prípadne ani jednu, bez toho, aby to ovplyvnilo hlavnú funkcionalitu programu.

Záver

Podarilo sa mi naplniť všetky ciele na letný semester. Program je funkčný, komunikuje s hardvérom, s GUI, a používateľovi zobrazuje ikonu. Na jednoduchú kompiláciu a inštaláciu som vytvoril *makefile* a *shell skripty*. Celý projekt vrátane windows programu a GUI zo zimného semestra, linuxového programu, arduino kódu, ako aj schéma zapojenia a návod na výrobu ovládača, sa nachádza na Githube.