VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

PŘESTAVBA PRŮMYSLOVÉHO POČÍTAČE PRO APLIKACI KIOSKU

RETROFITTING INDUSTRIAL COMPUTER FOR KIOSK APPLICATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — Petr Bužga
AUTHOR

VEDOUCÍ PRÁCE — Ing. Ladislav Dobrovský
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání bakalářské práce

Ústav:	Ústav automatizace a informatiky
Student:	**Petr Bužga**
Studijní program:	Strojírenství
Studijní obor:	Aplikovaná informatika a řízení
Vedoucí práce:	**Ing. Ladislav Dobrovský**
Akademický rok:	2022/23

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma bakalářské práce:

Přestavba průmyslového počítače pro aplikaci kiosku

Stručná charakteristika problematiky úkolu:

Vyřazená výpočetní technika představuje značný podíl elektronického odpadu (e–waste). Průmyslový počítač Lauer VPC se vyznačuje pevnou trvanlivou konstrukcí, velmi odolnou HW klávesnicí přímo v panelu a prostorem pro instalaci LCD displaye. Je tedy velmi žádoucí zvážit přestavbu s použitím co nejmenšího množství nových součástek s instalací do racku či na zeď. Případný provoz z baterie si vyžaduje volbu vhodných komponent a vývoj nativní aplikace.

Cíle bakalářské práce:

Analýza možnosti využití stávajících součástek.
Domluva na volbě platformy v týmu projektu.
Proveření funkčnosti periferí panelu (klávesnice, atd.).
Propojení HW s klávesnicí a displayem (přepokládá se nutnost doplnění HW pro obsluhu klávesnice vhodným převodníkem či RPi/Arduino).
Jednoduchá demonstrační aplikace kioskového typu napojená na webové rozhraní.
Analýza možností provozu z baterie.

Seznam doporučené literatury:

PECINOVSKÝ, Rudolf. Python: kompletní příručka jazyka pro verzi 3.10. Praha: Grada Publishing, 2021. Knihovna programátora (Grada). ISBN 978-80-271-3442-7.

PyQt: Introduction [online]. Riverbank Computing [cit. 2022-10-10]. Dostupné z: https://riverbankcomputing.com/software/pyqt/

ABSTRAKT

Průmyslové počítače jsou konstruovány s ohledem na funkčnost po mnoho let. Tato práce se zabývá případem, kdy byl i přesto takovýto počítač vyřazen z laboratoře jako nefunkční a je žádoucí se pokusit oživit alespoň některé části. Práce se nejdříve zabývá analýzou možností přestavby původního počítače Lauer VS386. V práci je navrženo využití RaspberryPi. Dále pak práce navrhuje ovladač pro původní maticovou klávesnici zabudovanou v předním panelu počítače a také kioskovou aplikaci pro správu docházky zaměstnanců pro ověření funkčnosti přestavby.

ABSTRACT

Industrial computers are designed to last for many years. This paper deals with the case where a computer was discarded from the laboratory as non-functional and it is desirable to try to revive at least some parts. The thesis first analyzes the possibilities of rebuilding the original Lauer VS386 computer. The use of RaspberryPi is proposed in the thesis. Then thesis proposes a driver for the original matrix keyboard embedded in the front panel of the computer as well as a kiosk application for managing employee attendance to verify the functionality of the rebuild computer.

KLÍČOVÁ SLOVA

Průmyslový počítač, RaspberryPi, PyQt, Elektronický odpad, Linux input subsystém, Klávesnice

KEYWORDS

Industrial computer, RaspberryPi, PyQt, E-waste, Linux input subsystem, Keyboard

ÚSTAV AUTOMATIZACE A INFORMATIKY

2023

BIBLIOGRAFICKÁ CITACE

BUŽGA, Petr. *Přestavba průmyslového počítače pro aplikaci kiosku.* Brno, 2023. Dostupné také z: https://www.vut.cz/studenti/zav-prace/detail/149472. Bakalářská práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky. Vedoucí práce Ladislav Dobrovský.

PODĚKOVÁNÍ

Chtěl bych poděkovat všem, kteří přispěli svými podněty a nápady k vytvoření této práce. Zejména se jedná o spolužáka Filipa, který byl součástí týmu, pracujícím na přestavbě. Dále bych chtěl poděkovat vedoucímu práce panu Ing. Dobrovskému za cenné nasměrování a připomínky.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že tato práce je mým původním dílem, vypracoval jsem ji samostatně pod vedením vedoucího práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury.

Jako autor uvedené práce dále prohlašuji, že v souvislosti s vytvořením této práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následku porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestně právních důsledků.

V Brně dne 20. 5. 2023 .....

Petr Bužga

OBSAH

1 ÚVOD

Průmyslové počítače jsou důležitou součástí moderního průmyslu. Představme si ale situaci, kdy takovýto počítač zůstane bez využití v jedné z laboratoří VUT. Právě to je případ počítače Lauer VS386, kterým se zabývá tato práce. Jedná se o průmyslový počítač firmy Elektronik-Systeme Lauer a na různých bazarech je stále možné jej koupit.

Průmyslové počítače jsou navrženy tak, aby vydržely náročné podmínky průmyslového prostředí a zároveň byly schopny spolehlivě pracovat po mnoho let. Nicméně, v průběhu času se mohou tyto počítače stát zastaralými a jejich výkon může být nedostatečný pro nové aplikace. Navíc během let může dojít k poškození různých součástí počítače a je nutné vhodně nahradit, což je případ právě námi přestavovaného počítače. Cílem práce je také návrh kioskové aplikace pro správu přístupů zaměstnanců, na kterém ověříme funkčnost přestavby.

Předkládaná bakalářská práce se tedy zabývá přestavbou průmyslového počítače a skládá se ze dvou hlavních částí. První část se zabývá propojením klávesnice předního panelu počítače s mikropočítačem. Je zde vysvětlen princip, jakým operační systém linux zpracovává vstupní zařízení. Dále je navržen samotný ovladač klávesnice. Druhá část se zabývá kioskovou aplikací pro jejíž vývoj byl zvolen programovací jazyk python. V této části se budeme věnovat i více programům s více vlákny, která slouží pro paralelní zpracování kódu programu.

2 NÁVRH PROVEDENÍ PŘESTAVBY

Námi přestavovaný počítač sloužil v jedné z laboratoří VUT. Po jeho vyřazení byl ovšem odložen a jeho součástky částečně rozebrány jako náhradní díly. Jedná se o průmyslový počítač firmy Lauer. Německý výrobce Elektronik-Systeme Lauer byl jeden z velmi významných výrobců průmyslových počítačů již od roku 1979. V roce 2007 byl ovšem v akvizici převzata společností Beijer Electronics. [1]

2.1 Průmyslové počítače

V dobách průmyslové revoluce v 19. století byly vynalezeny stroje k zvýšení efektivity či úplnému nahrazení tvrdé lidské práce. Průmysl se velmi záhy stal velmi důležitou součástí naší společnosti. Postupně se výroba přesunula spíše do rukou strojů než lidí a čím dál více byl kladen důraz co nejvíce osvobodit člověka od ovládání a sledování práce strojů. Díky tomu se vyvinulo průmyslové řízení a automatizace. Již v 80. letech minulého století se objevily první programovatelné řídící automaty (PLC). S rozvojem mikropočítačů a integrovaných obvodů se začalo přecházet k řízení celých výrobních provozů pomocí počítačů. [2]

Počítače obecně jsou jedním z nejvýznamnějších vynálezů moderní doby a mají obrovský dopad na náš každodenní život. Díky nim můžeme rychle a efektivně zpracovávat informace, komunikovat s ostatními lidmi na celém světě a řešit složité matematické a vědecké problémy. V posledních letech se s příchodem IoT (Internet of Things) stávají součástí vybavení domácností, jako jsou chytré reproduktory, termostaty a podobně.

Průmyslové počítače, jak již název napovídá, jsou takové počítače jejichž hlavním uplatněním je využití pro průmyslové aplikace. Využíváme je například v automatizaci, řízení či logistice. Průmyslové počítače jsou od kancelářských či domácích počítačů rozdílné. Jsou totiž ve většině případů konstruovány pro zpracování velkých objemů dat v reálném čase, přičemž je kladen velký důraz na spolehlivost. Právě spolehlivost je velmi důležitá, protože zatímco uživatel v kanceláři si může dovolit počkat, zastavení výrobní linky je velmi nákladné. Klíčovou vlastností takovýchto počítačů je tedy právě schopnost zpracovat informace v reálné čase. Dále také možnost napojení na průmyslové sběrnice a systémy. Tyto výhody jsou ovšem vykoupeny vysokou pořizovací cenou.

Jelikož průmyslové počítače využíváme v nepříznivém prostředí jsou často koncipovány tak, aby vydrželi náročné podmínky jako jsou nárazy, vibrace, vlhkost, elektromagnetická pole, prach či teplota. Tento typ počítačů označujeme pojmem „rugged computer“ – odolné počítače. Nicméně existují případy, kdy průmyslový počítač umístíme například do racku a není nutné zajistit odolnost vůči vnějším vlivům. Při výběru průmyslového počítače je tedy vždy nutné zohlednit prostředí ve kterém bude provozován. Důležité může být například využití pasivního chlazení místo chlazení vzduchem z důvodu nasávání prachu. [2]

Neopomenutelnou vlastností je dlouhověkost počítačů a s nimi spjatých výrobních provozů. Z důvodů vysoké pořizovací ceny jsou výrobní linky a stroje konstruovány tak, aby sloužili řádově nižší desítky let. Podíváme-li se ovšem na osobní počítače s životností řádově v letech zjistíme, že jejich využitelnost je v tomto ohledu naprosto nedostatečná. Proto se pro tyto aplikace využívají právě průmyslové počítače jejichž výrobci musí zajistit hardwarovou i softwarovou podporu po celou dobu nasazení počítače. [2]

2.2 Problematika elektroodpadu

Výpočetní technika se v posledních letech rozrůstá neuvěřitelným tempem. Během roku 2019 bylo na celém světě vytvořeno 53,6 milionů tun elektronického odpadu. Což je o 8,85 milionů tun více než v roce 2016 [3]. Očekává se že do roku 2030 naroste elektronický odpad na 74.7 milionů tun, pokud nezměníme náš přístup [4]. Jedním z následků tohoto prudkého vývoje je krátká doba, za kterou i výkonný hardware začíná zaostávat při využívání nejnovějšího softwaru. Elektronický odpad se stal nejrychleji rostoucím odpadem na světě. Rostoucí množství tohoto odpadu představuje hrozbu pro životní prostředí, nicméně jeho velkou výhodou je poskytování běžných, drahocenných a kritických surovin, které lze z elektronického odpadu získat zpět. Pouze 17 procent vygenerovaného e-odpadu v roce 2019 bylo hlášeno jako shromážděné a recyklované. [4]

Elektronický odpad (v zahraniční literatuře jako e-waste) je problematický právě díky obsahu velkého množství látek nebezpečných pro životní prostředí jako jsou olovo, kadmium, rtuť, nikl, šestimocný chrom či bromované zpomalovače hoření [5]. Na druhou stranu je výhodou elektronického odpadu poměrně velké množství materiálů, které je možné recyklovat a znovu využít ve výrobě. Typickým zástupcem je měď, která se používá nejčastěji pro vodivá spojení na deskách plošných spojů (PCB).

2.3 Stav součástek počítače

Právě z důvodů popsaných v kapitole o elektroodpadu výše využijeme co nejvíce původních součástí průmyslového počítače. Počítač se skládal z počítačové skříně, základní desky a čelního panelu s klávesnicí a displayem. Jedná se o počítač Lauer VS386 VCP GRF10. Na obrázku 1 je zobrazena fotodokumentace podobné verze původního počítače prezentovaná na internetu. Náš model měl pouze méně funkčních tlačítek.

Obr. 1: Fotografie původního počítače [30]

Základní deska postrádala paměti RAM. Centrální procesorová jednotka (CPU) byla značně poškozená, pravděpodobně díky vysunutí z patice (socket) a pohození v krabici. Jednalo se o procesor i486. Na obrázku 2 je vyfocen vnitřek počítače se základní deskou.

Obr. 2: Základní deska počítače

Naproti tomu počítačová skříň a přední panel byl ve velmi zachovalém stavu. Přední panel obsahoval plošný spoj s klávesnicí a display. Display se nám bohužel nepodařilo zprovoznit. Na plošném spoji klávesnice byly tlačítka plně funkční. Původně byla klávesnice a počítač připojeny pomocí DIN konektoru. V předním panelu bylo možné připojit pomocí tohoto konektoru i externí klávesnici či myš.

2.4 Volba postupu práce

Jelikož by bylo velmi problematické nahradit chybějící či poškozené díly novými, jako například paměti RAM či procesor, bylo rozhodnuto nahradit celou původní základní desku mikropočítačem. Dále bylo rozhodnuto zachovat přední panel s klávesnicí. Display pro jeho nefunkčnost nahradíme novým. I přes to, že velkou část původního počítače nebylo možno využít, má přestavba smysl alespoň díky zachování kvalitního předního panelu a klávesnice.

Na přestavbě pracoval dvoučlenný tým. Součástí této práce je právě oživení klávesnice (kapitola 3) a vývoj kioskové aplikace (kapitola 4). Kolega pak zpracovával 3D tisk dílů k sestavení počítače jako celku a vývoj serverové části aplikace, která bude komunikovat s kioskem pomocí HTTP.

2.5 Volba mikropočítače

Pro náš projekt byl zvolen mikropočítač RaspberryPi 3 díky dobré využitelnosti GPIO pinů, dostatečnému výpočetnímu výkonu a operačním systémem s grafickým uživatelským rozhraním (GUI), které je vhodné právě pro naši aplikaci.

RaspberryPi vzniklo jako projekt několika vysokoškolských profesorů, kteří si na počátku tisíciletí všimli výraznému snižování povědomí o programovaní mezi studenty nastupujícími na vysokou školu. Zjistili, že studenti často nemají přístup k vhodnému hardwaru, na kterém by si mohli zkoušet programovat. Založili tedy neziskovou organizaci a přišli s návrhem mikropočítače s cenou 20$ a takovými vlastnostmi, které budou vhodné právě pro výuku programování. Brzy narazili na obrovskou poptávku a jejich mikropočítač RaspberryPi se stal známým po celém světě. Desítky vývojářů pracovalo na vylepšeních celé platformy, a to v rámci neziskové organizace bez nároku na honorář. Jejich platforma se tak záhy stala jednou z nejpoužívanějších pro výukové účely či domácí projekty. [6]

RaspberryPi se řadí mezi jednodeskové počítače neboli SBC (Single Board Computer). Jedná se o takový počítač, který je postavený celý na jediné desce plošného spoje s pamětí, mikroprocesorem, vstupně-výstupními rozhraními a dalšími funkcemi. [7]

Vzhledem k využití RJ-45 konektoru pro připojení do sítě byl vybrán model B+. Druhou variantou je model A+, který ovšem tento konektor postrádá. Plus přidané na konci označení modelu značí druhou vylepšenou verzi RaspberryPi 3. Zvolený model je zobrazen na obrázku 3. [8]

Obr. 3: RaspberryPi model 3 B+[8]

2.6 Analýza možností provozu z baterie

Díky robustnosti předního panelu počítače se nabízí možnost využití kiosku v mobilních aplikacích, jakou jsou stavby či odlehlá pracoviště. Zvolený mikropočítač RaspberryPi je pro takovouto aplikaci vhodný, jelikož se napájí pomocí USB. Pokud bychom se ovšem rozhodli prohlásit kiosek jako mobilní, bylo by nutné lépe vyřešit připojení k internetu na odlehlém pracovišti. Ať už pomocí modulu se SIM kartou, či využitím off-line přístupu.

Požadavky na napájení jsou rozdílné dle modelu RaspberryPi. Faktem ovšem je, že všechny modely vyžadují napájecí napětí 5,1 V. Rozdílný je nicméně proud, který se obvykle zvyšuje v závislosti na modelu. Všechny modely až po RaspberryPi 3 vyžadují napájecí konektor micro USB, zatímco RaspberryPi 4 a RaspberryPi 400 využívají konektor USB-C. Zatímco původní model RaspberryPi 1 Model A měl doporučenou kapacitu zdroje 700 mA, námi použitý model RaspberryPi 3 již vyžaduje 2,5 A. [9]

Pro napájení lze využít již existujících řešení dedikovaných přímo pro použití s RaspberryPi či vytvoření vlastního řešení baterie.

PiJuice HAT

Jedná se o zdroj nepřerušovaného napájení, často označovaný zkratkou UPS (z anglického Uninterruptible Power Supply) pro RaspberryPi. Umožnuje celkovu správu napájení RaspberryPi, jako je plánované vypnutí či zapnutí, či funkce záložního zdroje při výpadku napájení. Dále je možné připojení alternativních zdrojů napájení jako jsou solární panely, větrné turbíny, ale také připojení externích baterií. PiJuice Hat má vestavěné hodiny pro sledování času a plánování úloh, co lze využít zejména, když je RaspberryPi vypnuto. Má také integrovaný čip mikrokontroléru, který řídí funkci bezpečného softwarového vypnutí, ale také vzdálené zapnutí a vypnutí. [10]

Obr. 4: PiJuice HAT [31]

3 PROPOJENÍ KLÁVESNICE S RASPBERRYPI

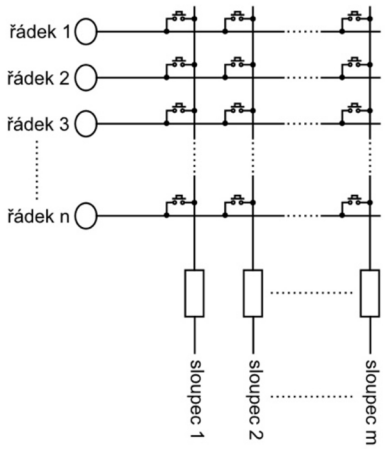
Hlavním z cílů naší přestavby bylo využití předního panelu. Právě klávesnice je jednou z nejvíce zachovalých součástí. Samotná klávesnice se původně skládala ze samotného plošné spoje s tlačítky a také z druhého plošného spoje kontroléru, který zajišťoval komunikaci s původním procesorem počítače. Vzhledem k neexistující dokumentaci a stavu kontroléru bylo rozhodnuto zachovat pouze klávesnici samotnou a kontrolér nahradit pomocí softwarového ovladače v RaspberryPi.

3.1 Princip maticových klávesnic

Většina aplikací mikroprocesorové techniky vyžaduje zajištění komunikace uživatele se strojem. I přes vývoj počítačů zůstávají klávesnice již od počátku základním způsobem využívaným pro tuto komunikaci.

Jednou z možností konstrukce jsou tzv. maticové klávesnice. Jedná se o pole tlačítek, jež jsou uspořádány do řádků a sloupců tak, aby každé klávese byl přiřazen právě jeden sloupec a jeden řádek (viz. obrázek 5). Detekce zmáčknutých kláves poté probíhá díky skenování. Nejdříve na právě jeden sloupec přivedeme napětí. Pokud uživatel zmáčkne tlačítko, propojí se řádek se sloupcem a na řádku můžeme naměřit nástupnou hranu napětí. Máme tedy přesně určeno, ve kterém sloupci a na kterém řádku byla zmáčknuta klávesa, což odpovídá konkrétnímu znaku na klávesnici. Tento postup opakujeme pro každý sloupec klávesnice v dostatečně malých intervalech. [11]

V typických klávesnicích se využívá mikrokontroleru, který čte zmáčknuté klávesy a zajišťuje komunikaci s počítačem. V závislosti na stisknuté klávese dojde k odeslání tzv. scancode do počítače. Driver klávesnice mu poté přiřadí symbol. V našem řešení jsme skenování pomocí separátního mikrokontroleru nahradili softwarovým řešením.



Obr. 5: Schéma maticové klávesnice

3.2 Měření plošného spoje

Pro funkci klávesnice je potřeba rozklíčovat fyzické umístění jednotlivých tlačítek v matici klávesnice. Vzhledem ke tvaru klávesnice, jak lze vidět na obrázku 6, nejsou tlačítka rovnoměrně rozmístěna a bylo nutné fyzicky proměřit jejich polohu. Bylo zjištěno, že klávesnice se skládá z 8 sloupců a 8 řádků. Dále jsme přiřadili každému tlačítku konkrétní číslo.

Obr. 6: Rozmístění tlačítek na klávesnici

Díky tomu bylo možné vytvořit tabulku přiřazující každému tlačítku konkrétní sloupec a řádek. Sloupcům a řásků se pak přiřadí ještě číslo GPIO vstupu a také pozice daného spoje v konektoru.

Tabulka 1: Přiřazení tlačítek do sloupců a řádků

sloupec číslo

	s0	**s1**	**s2**	**s3**	**s4**	**s5**	**s6**	**s7**
	1	2	3	4	5	6	58	50
	7	8	9	10	11	12	49	51
	13	14	15	16	17	18		52
	19	20	21	22	23	24		53
	25	46	26	27	28	29		54
	30	47	31	32	33	34		55
	38	48	44	35	36	37		56
	42	43	45	39	40	41		57
konektor	11	12	13	14	15	16	17	18
GPIO	4	17	27	22	5	6	13	19

řádek číslo

	r0	**r1**	**r2**	**r3**	**r4**	**r5**	**r6**	**r7**
	1	7	13	19	25	30	42	48
	2	8	14	20	26	31	44	45
	3	9	15	21	27	32	43	38
	4	10	16	22	28	33	35	39
	5	11	17	23	29	34	36	40
	6	12	18	24	47	46	37	41
	57	56	55	54	53	52	51	50
	58							49
	R8	R7	R6	R5	R4	R3	R2	R1
konektor	10	9	8	7	6	5	4	3
GPIO	18	23	24	25	12	16	20	21

Tabulka 2: Přiřazení tlačítek k jednotlivým symbolům

symbol	číslo tlačítka	sloupec	řádky
A	6	5	0
B	5	4	
C	4	3	
D	3	2	
E	2	1	
ESC	1	0	
F	12	5	1
G	11	4	
H	10	3	
I	9	2	
J	8	1	
DEL	7	0	
K	18	5	2
L	17	4	
M	16	3	
N	15	2	
O	14	1	
INS	13	0	
P	24	5	3
Q	23	4	
R	22	3	
S	21	2	
T	20	1	
BS	19	0	
UP	25	0	4
TAB	30		5
DWN	38		7
-	26	2	4
9	27	3	
8	28	4	
7	29	5	

symbol	číslo tlačítka	sloupec	řádky
+	31	2	5
6	32	3	
5	33	4	
4	34	5	
3	35	3	6
2	36	4	
1	37	5	
return	39	3	7
,	40	4	
0	41	5	
ctrl	47	1	4
alt	46		5
shift	48	1	7
up	43	1	6
down	44	2	6
left	45	2	7
right	42	0	6
F01	58	6	0
F02	57	7	0
F03	56		1
F04	55		2
F05	54		3
F06	53		4
F07	52		5
F08	51		6
F09	50		7
F10	49	6	7

3.3 Propojení klávesnice pomocí GPIO

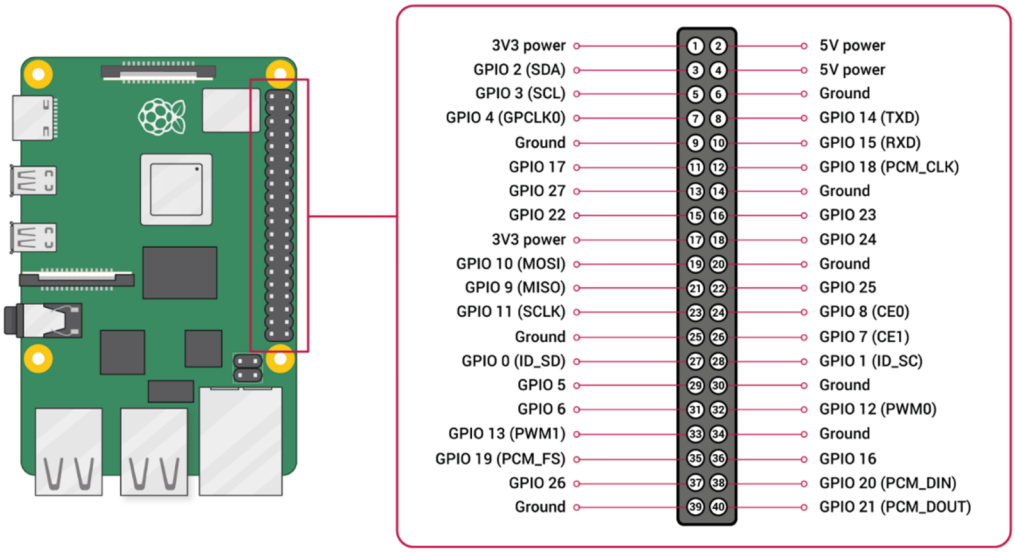
S růstem mikropočítačů vzniká potřeba k nim připojovat rozličná vstupní či výstupní zařízení dle požadavků uživatelů. Právě použití „General Purpose Input Output“ (GPIO) je jednou z metod propojení procesorů s okolními zařízeními. Jedná se o takový přístupný softwarově ovládaný elektrický kontakt, který můžeme využít pro zachycení vstupních signálů nebo také jejich pomocí zajistit výstup informací. Jejich využití ovšem není navrhnuto pro konkrétní účel [12].

Vstupy lze často použít jako signály IRQ. To jsou takové signály, kterými požádá zařízení procesor o pozornost, tedy požádá o přerušení probíhajícího procesu za účelem provedení důležitější akce. [13]

Dnešní SoC procesory (System on Chip) si velmi často zakládají na využití GPIO. V některých případech lze každý nepřidělený pin nakonfigurovat jako GPIO. Většina

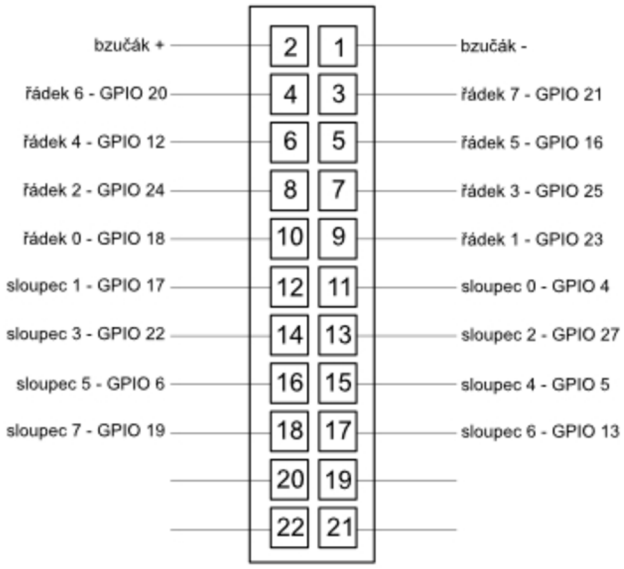
jednodeskových počítačů (SBC) je GPIO vybavena. Populárním zařízením, které pracuje s GPIO, je mikrokontroler Arduino. Tento počítač nicméně nedisponuje operačním systémem, a je spíše zařízením, které je navrženo tak, aby bylo programovatelné speciálně pro ovládání jiných zařízení. [12, 14]

Zatímco původní model RaspbberyPi 1 z roku 2014 mohl operovat s kratším 26 pinovým konektorem, na nynějších deskách nalezneme 40 pinů, ze kterých můžeme využít jako GPIO 27. Jak lze vidět na obrázku 7, GPIO piny nejsou řazeny v číselném pořadí. GPIO 0 a 1 jsou pak rezervovány ke speciálnímu užití. [9]

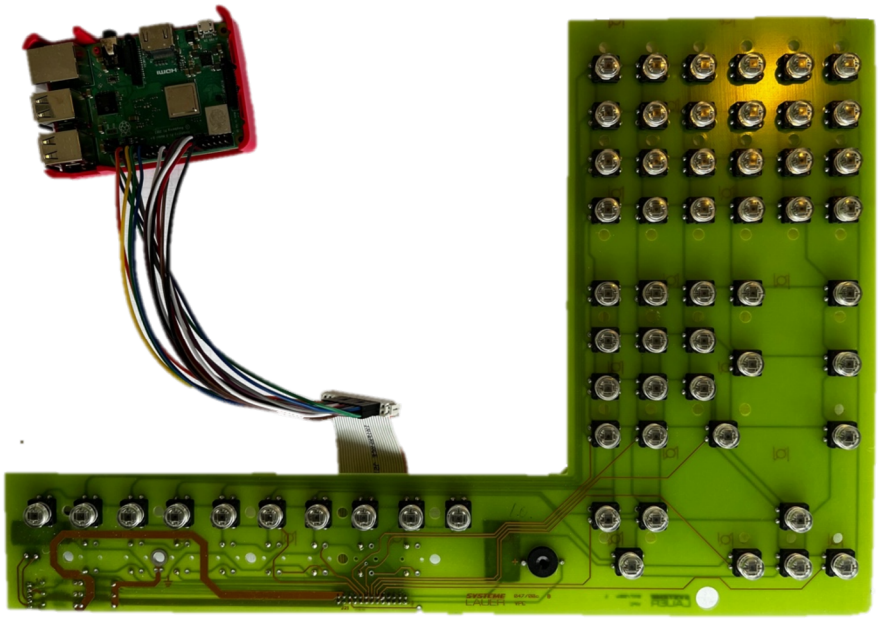


Obr. 7: Schéma konektoru RaspberryPi [9]

Pro propojení klávesnice a RaspberryPi těchto GPIO využijeme. Plošný spoj klávesnice má výstupní konektor, který propojíme s piny na RaspberryPi dle přiloženého diagramu na obrázku 8. Na obrázku 9 vidíme samotný plošný spoj propojený s RaspberryPi.



Obr. 8: Schéma konektoru klávesnice



Obr. 9: Plošný spoj klávesnice propojený s GPIO RaspberryPi

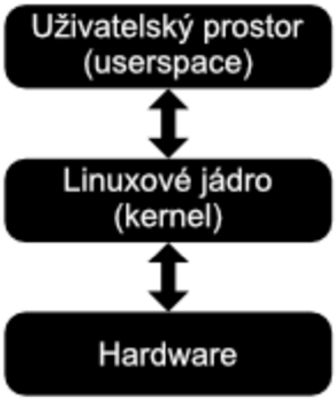
3.4 Ovladač klávesnice v RaspberryPi

Typicky probíhá vývoj ovladačů a systémových souborů pro operační systém linux v jazyce C [15]. Pro náš celý projekt byl ovšem zvolen programovací jazyk Python. I když by bylo vhodnější použít jazyk C pro ovladač klávesnice, jelikož je to jazyk používaný v linuxovém jádře, rozhodl jsem se pro vývoj prototypu ovladače v jazyce Python. V případě komerčního využití by bylo vhodné kód přepsat do jazyka C, nicméně lze prohlásit, že implementace v jazyce Python je plně funkční.

K využití Pythonu také přispěl fakt, že lze bez problémů v operačním systému spouštět procesy v Pythonu. Nevýhodou ovšem je, že jako podmínka funkčnosti ovladače je nutná instalace Pythonu na daný operační systém. Další nevýhodou oproti jazyku C je také pravděpodobně větší zatížení výpočetní výkonu a paměti procesoru.

3.4.1 Linux Input Subsystem

Linux Input Subsystém je klíčovým prvkem jádra Linuxu, který slouží k zpracování vstupních zařízení, jako jsou klávesnice, myši a dotykové obrazovky. Tento subsystém umožňuje aplikacím získávat a zpracovávat vstupní data od uživatelů. Skládá se z řady ovladačů (driverů), které zprostředkovávají komunikaci mezi hardwarovými zařízeními, jádrem operačního systému (kernel) a uživatelským prostorem (userspace). Díky tomu mohou aplikace pracovat se vstupními zařízeními bez potřeby znalosti hardwarových detailů a komunikovat s nimi pomocí standardních rozhraní. Input subsystém je tedy klíčovým prvkem pro vývoj aplikací, které pracují se vstupními zařízeními.



Obr. 10: Architektura linux systému

Pro správné porozumění fungování tohoto systému je nezbytné mít povědomí o architektuře linuxových systémů. Jak ukazuje diagram na obrázku 10, systém lze rozdělit na několik částí. Základem je samotný hardware, který zahrnuje výpočetní jednotky, paměti a také vstupní a výstupní zařízení, jako jsou klávesnice, myši, monitory, tiskárny a další. Pro uživatele je vyhrazen uživatelský prostor, ve kterém mohou spouštět aplikace a pracovat se soubory. Mezi uživatelským prostorem a hardwarem se nachází další vrstva – jádro operačního systému (kernel). Jedná se o počítačový program, který

má kompletní kontrolu nad celým systémem. Kernel zpracovává procesy, systémové prostředky a eviduje připojená zařízení. Pokud tedy uživatel potřebuje interagovat s hardwarem, musí o to požádat právě kernel. I přes značný počet linuxových distribucí mají všechny společné právě jádro.

K linux input subsystému lze přistupovat jak z jádra, tak uživatelského prostoru. V této práci se bude zabývat přístupem z uživatelského prostoru. Architektura tohoto systému se skládá ze dvou skupin modulů. Jsou to ovladače zařízení (device drivers) a obsluhy událostí (event handlers).

Díky této architektuře mohou aplikace pracovat se vstupními zařízeními bez potřeby znalosti hardwarových detailů a komunikovat s nimi pomocí standardních rozhraní. Mezi výhody také patří snadná konfigurace a správa vstupních zařízení. Moduly ovladačů jsou navrženy tak, aby mohly být snadno přidávány nebo odebrány podle potřeby. To znamená, že nová vstupní zařízení mohou být snadno integrována do systému a starší zařízení mohou být nahrazena novými bez nutnosti zásahu do jádra operačního systému. [16]

Ovladače zařízení (Device Drivers)

Ovladače zařízení komunikují s hardwarem (například přes USB) a generují události (stisknutí kláves, pohyby myší). Jsou tedy navrženy tak, aby dokázaly zprostředkovat komunikaci mezi hardwarem a jádrem. Když uživatel používá vstupní zařízení, jako je klávesnice nebo myš, moduly ovladačů přijímají signály od hardware a převádějí je na události. Mezi univerzální komplexní ovladače v linuxovém systému se řadí například hid-generic. Pro zařízení usb je možné také využít jednodušší ovladač usbmouse. [16]

Zpracování událostí (Event Handlers)

Moduly pro zpracování událostí (Event handlers) získávají události z jádra. Jsou navrženy tak, aby mohly zpracovávat různé typy událostí, jako jsou stisknutí kláves, pohyby myší, joysticky, dotykové obrazovky a další. Když modul ovladače zařízení přijme událost z hardwarového vstupního zařízení, předá ji modulu pro zpracování událostí, který ji zpracuje a následně předá aplikaci nebo systémové službě, která ji potřebuje.

Mezi základní patří modul evdev, které poskytuje univerzální a nezávislé rozhraní pro různá vstupní zařízení. Evdev předává události generované jádrem přímo programu a poskytuje jim vždy časovou značku, typ, kód a hodnotu. Typ události určuje, jaký druh události nastal. Například typ EV_REL označuje událost relativního pohybu, zatímco typ EV_KEY označuje stisknutí nebo uvolnění klávesy. Kód události (event code) označuje konkrétní událost, jako například REL_X pro relativní pohyb osy X nebo KEY_BACKSPACE pro klávesu backspace. Hodnota (value) události říká jakou číselnou hodnotu událost přenáší. Pro typ události EV_REL se jedná o relativní změnu polohy myši, pro typ EV_ABS o absolutní novou polohu (např. u joysticku) či pro typ EV_KEY je hodnota 0 pro uvolnění klávesy a 1 pro stisknutí klávesy. Všechny typy událostí a kódů jsou definovány v souboru „*include/uapi/linux/input-event-codes.h*“,

který obsahuje seznam kódů událostí a jejich příslušných typů. Kódy událostí jsou nezávislé na hardwaru, což umožňuje snadnou konfiguraci a správu vstupních událostí.

Rozhraní evdev je preferovanou metodou zpracování událostí v uživatelském prostoru a všichni uživatelé a aplikace jsou vyzýváni k používání evdev rozhraní jako preferovaného způsobu získávání uživatelského vstupu. Dalšími již méně využívanými moduly pro zpracování událostí jsou například keyboard, mousedev či joydev. Jejich využití je ovšem specifické. [16]

3.4.2 Linux systemd

Systemd je jedním z moderních init démonů linuxového jádra, které zajišťují inicializaci operačního systému a následné řízení služeb. Byl vytvořen programátorem Lennartem Poetteringem ze společnosti Red Hat jako řešení problémů s tehdejšími init démony, které byly navrženy pro práci ve statickém prostředí. Tyto starší init démony měly problémy s novým hardwarem, jelikož se počítače změnili ze statického na událostmi řízené prostředí. S nástupem nového hardware, který se stal událostmi řízeným prostředím, nastala potřeba vytvořit nové init démony, které by dokázaly efektivně pracovat, například díky využití paralelního zpracování. Navíc, s nástupem nových služeb, se klasické init démony musely vypořádat s tím, že musí spouštět stále více a více služeb. Celý proces inicializace systému byl tedy méně efektivní a znatelně pomalý.

Moderní init démony, ke kterým systemd patří, se snaží řešit problémy neefektivního startu systému a nestatického prostředí. Systemd se záhy stal nejpopulárnějším novým init démonem. Velká část linuxových distribucí, jako je Ubuntu, RHEL či Fedora přešly na systémový démon systemd, přičemž zachovávají zpětnou kompatibilitu s klasickými SysVinit, Upstart nebo BSD init démony.

Ačkoli hlavním úkolem systemd je spouštět a zastavovat služby, může spravovat i další typy věcí označované jako takzvané „Target units“. Jsou to skupiny, která se skládají z názvu, typu a konfiguračního souboru a jsou zaměřeny na konkrétní službu nebo akci. Je jich definováno 12: automount, device, mount, path, service, snapshot, socket, target, timer, swap, slice a scope.

Mezi dva hlavní patří „Service unit“, kterou se zabýváme v této práci. Tento typ slouží právě ke spouštění a zastavování démonů. Řeší nejen samotné spuštění, ale i spoustu další požadavků jako je uživatelské oprávnění, přidělení systémových prostředků, které další služby musí být spuštěny, kdy může být tato služba spuštěna, který konfigurační soubor s prostředím použít atd. [17]

Každá služba se skládá v konfiguračního souboru a samotné scriptu, který chceme spustit. Samotné scripty můžeme zavést do adresáře „/usr/bin“. Konfigurační soubor pro systemd lze zavádět do dvou umístění [17]:

1. */etc/systemd/system*
    a. Toto umístění je využito pro ukládání vlastních přizpůsobených konfiguračních souborů.
    b. Soubory v tomto umístění nejsou přepsány instalacemi nebo aktualizacemi softwaru.
2. */lib/systemd/system*
    a. Toto umístění je využito pro ukládání systémových konfiguračních souborů.
    b. Soubory v tomto umístění se přepisují při aktualizaci softwaru.

3.4.3 Algoritmus skenování

Jak již bylo zmíněno v kapitole 3.1 o maticových klávesnicích funguje ovladač na principu skenování. Pokud program při skenování zjistí, že je na nějakém řádku napětí, zapíše pomocí metody UInput zmáčknutí dané klávesy. Jaká konkrétní klávesa byla zmáčknuta se přiřadí pomocí čísel GPIO pinů daného sloupce a řádku ze slovníku. Poté se čeká na sestupnou hranu (puštění klávesy), se kterou se zapíše pomocí metody UInput puštění klávesy a program se vrací ke skenování. Pokud bychom klávesu z jakéhokoliv důvodu nepustili, tak čekání má časové omezení 500ms a poté dojde k zápisu puštění klávesy.

Jedním z problémů je fakt, že kontakty klávesnice mají ve většině případů sklony k zakmitávání při sepnutí i při rozpojení. Tyto zákmity (trvají obvykle řádově 10 ms) je nutné při skenování odstranit, jelikož způsobují při jednom stisku tlačítka zapsaní dané klávesy několikrát. Toto je řešeno pomocí časové prodlevy mezi jednotlivými cykly skenování. Tento časový interval, kdy program čeká vždy pro naskenování všech sloupců a řádků byl zvolen 1/60 sekundy.

3.4.4 Vlastní řešení ovladače klávesnice

Ovladač se spouští při startu systému jako služba v systemd. Tato služba pomocí práv uživatele root spouští python program, který v pravidelném intervalu skenuje jednotlivé sloupce matice klávesnice. Ve výpisu kódu 1 je zobrazena část kódu ovladače zajišťující skenování. Hlavní cyklus využívá nekonečné smyčky while a v případě detekce zmáčknutí klávesy volá příslušnou funkci pro zpracování události.

Výpis 1: Algoritmus skenování klávesnice

```
#nastavení nové klávesnice
ui = UInput(name = "GPIO Keyboard")

#přiřazení GPIO pinů řádkům a sloupcům
column_pin_numbers = [4, 17, 27, 22, 5, 6, 13, 19]
row_pin_numbers = [18, 23, 24, 25, 12, 16, 20, 21]

#mapování kláves
keymap = {
   (column_pin_numbers[5], row_pin_numbers[0]): (e.KEY_A),
        ...
}

shift_keymap = {
   (column_pin_numbers[5], row_pin_numbers[0]): (e.KEY_U),
   ...
 }

#nekonečný cyklus, který zajišťuje skenování klávesnice
while 1:
   time.sleep(1/60)
   for column_pin in column_pin_numbers:
      GPIO.output(column_pin, 1)
      for row_pin in row_pin_numbers:
         if GPIO.input(row_pin) == 1:
            if (row_pin == row_pin_numbers[7] and
               column_pin == column_pin_numbers[1]):
               Write_Shift_Presed_Key()
            else:
               Write_Input(column_pin, row_pin, keymap)
      GPIO.output(column_pin, 0)
```

Funkce „Write_Input“ (viz. výpis 2) pak zajišťuje zapsání zmáčknuté klávesy. Využívá k tomu výše již zmíněný modul pro zpracování událostí (Event Handler) evdev. Nejdříve definujeme nové zařízení klávesnice, které se tak zapíše mezi ostatní vstupní zařízení do „*/dev/input*“. Pokud chceme zapsat nový stisk klávesy využijeme metody write. Funkce umožňuje mapování kláves z více slovníků z důvodu možnosti zmáčknutí klávesy shift.

Výpis 2: Funkce zapisující stisknutí klávesy do evdev

```
#funkce, která zapíše zmáčknutí klávesy do systemd
from evdev import UInput, ecodes as e
def Write_Input(column_pin, row_pin, keys_mapping):
   ui.write(e.EV_KEY, keys_mapping[column_pin,row_pin], 1)
   GPIO.wait_for_edge(row_pin, GPIO.FALLING, timeout = 500)
   ui.write(e.EV_KEY, keys_mapping[column_pin,row_pin], 0)
   ui.syn()
   return
```

Funkce „Write_Shift_Pressed_Key“ je volána v případě zmáčknutí klávesy shift, která slouží k přístupu k sekundárním znakům klávesy. Pokud je zmáčknuta klávesa shift, tak program vstoupí do této funkce a čeká na stisk dalšího tlačítka. Po jeho stisku zapíše sekundární znak klávesnice. Mapování sekundárních znaků je tedy aktivní, dokud není máčknuta další klávesa. Není tedy důležité stále držet klávesu shift, při psaní sekundárních znaků. Funkční jsou nicméně obě varianty.

Některé speciální symboly není možné zapsat standartním zavoláním jejich kódu. Jsou to takové klávesy, které standartně píšeme s pomocí klávesy shift (například dvojtečka). Pro tyto případy se nevolá funkce Write_Input, ale zapíšeme je stiskem jiné klávesy společně s klávesou shift. Pro zapsání dvojtečky je tedy potřeba zadat systému stisk klávesy shift a středníku. Bylo tedy nutné rozšířit slovníkv v těchto případech pomocí tuple.

Výpis 3: Zápis znaků pomocí klávesy shift

```
#mapování kláves
shift_keymap = {
   ...
   (column_pin_numbers[3], row_pin_numbers[1]):
   (e.KEY_LEFTSHIFT, e.KEY_SEMICOLON),
   ...
}
```

```
#funkce, která řeší zápis klávesy po zmáčknutí tlačítka shift
def Write_Shift_Presed_Key():
   GPIO.output(column_pin_numbers[1], 0)
   while 1:
      time.sleep(1/60)
      for column_pin in column_pin_numbers:
         GPIO.output(column_pin, 1)
         for row_pin in row_pin_numbers:
            if GPIO.input(row_pin) == 1 and row_pin != row_pin_numbers[7]:
                if isinstance(shift_keymap[column_pin,row_pin], tuple):
                  ui.write(e.EV_KEY, shift_keymap[column_pin,row_pin][0], 1)
                  ui.write(e.EV_KEY, shift_keymap[column_pin,row_pin][1], 1)
                  ui.write(e.EV_KEY, shift_keymap[column_pin,row_pin][0], 0)
                  GPIO.wait_for_edge(row_pin, GPIO.FALLING, timeout = 500)
                  ui.write(e.EV_KEY, shift_keymap[column_pin,row_pin][1], 0)
                  ui.syn()
                  return
            else:
                Write_Input(column_pin,row_pin, shift_keymap)
                return
         GPIO.output(column_pin, 0)
```

Dále bylo nutné vytvořit nový konfigurační soubor služby a zavést ho do systemd. Naše nová služba tedy byla zavedena do „/etc/systemd/system/GPIO-keyboard.service“. Jak lze vidět ve výpisu kódu 4, služba je nastavená tím způsobem, aby se při pádu, či neočekávaném ukončení opět znovu načetla. Služba spouští script „*/usr/bin/GPIO-keyboard.py*“.

Výpis 4: Konfigurační soubor služby

```
[Unit]
Description=Ovladac klavesnice pomoci gpio
StartLimitInternalSec=0

[Install]
WantedBy=multi-user.target

[Service]
Type=simple
ExecStart=/usr/bin/python3 /usr/bin/GPIO-keyboard.py

User=pi
Restart=always
```

4 KIOSKOVÁ APLIKACE

Dalším cílem této bakalářské práce bylo navrhnout kioskovou aplikaci, která bude využívat programovací jazyk Python a knihovnu pro tvorbu uživatelského rozhraní PyQt. Aplikace byla navržena pro evidenci docházky zaměstnanců s využitím identifikace pomocí RFID čtečky karet.

Jak již bylo zmíněno v kapitole 2.4, aplikace je rozdělena na dvě části – přístupový terminál a serverovou část. Přístupový terminál slouží zaměstnancům k fyzickému přihlášení nebo odhlášení a serverová část ukládá data o jednotlivých průchodech. V této práci se zaměříme pouze na přístupový terminál.

Při návrhu aplikace bylo zohledněno využití již existujících zaměstnaneckých karet nebo karet ISIC pro identifikaci zaměstnanců. Cílem aplikace je zjednodušit proces evidování docházky zaměstnanců a umožnit snadnou správu dat o průchodech.

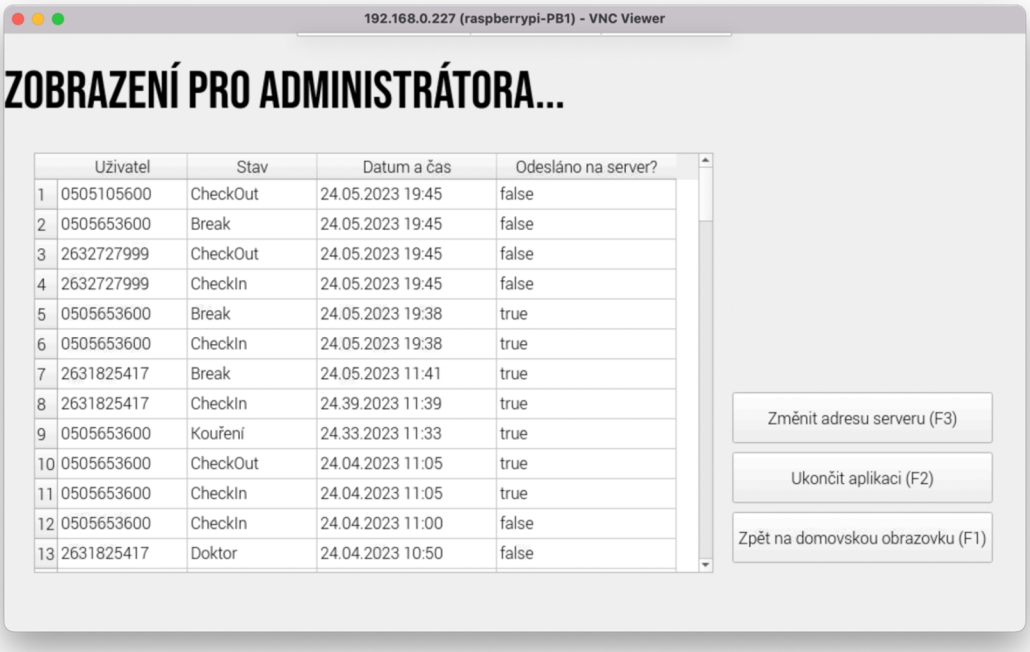
4.1 Návrh aplikace

Systém musí být dostatečně uživatelsky jednoduchý, tak aby nikdo ze zaměstnanců nemusel procházet obsáhlým školením o používání. Navrhli jsme proto rozhraní, kdy stačí přiložit čipovou kartu a tím se odešle informace o průchodu. Uživatel si pomocí jednoduchých kláves může dopředu zvolit, zda chce zaznačit příchod či odchod. Na obrázku 11 lze vidět rozhraní aplikace. Pro ovládání využijeme funkčních kláves na předním panelu.

Obr. 11: Návrh rozhraní aplikace

4.1.1 Přístup pro administrátora

Návrh aplikace počítá i s jednoduchým přístupem pro administrátora systému. Tento přístup slouží jak pro nastavení, tak také pro zobrazení off-line historie průchodů. Je možné zde nastavit například IP adresu serveru či aplikaci ukončit. Historie se zobrazuje jako lokální log průchodů. Jak lze vidět na obrázku 12 u každého průchodu vidíme, zda se podařil nahrát na server.



	Uživatel	Stav	Datum a čas	Odesláno na server?
1	0505105600	CheckOut	24.05.2023 19:45	false
2	0505653600	Break	24.05.2023 19:45	false
3	2632727999	CheckOut	24.05.2023 19:45	false
4	2632727999	CheckIn	24.05.2023 19:45	false
5	0505653600	Break	24.05.2023 19:38	true
6	0505653600	CheckIn	24.05.2023 19:38	true
7	2631825417	Break	24.05.2023 11:41	true
8	2631825417	CheckIn	24.39.2023 11:39	true
9	0505653600	Kouření	24.33.2023 11:33	true
10	0505653600	CheckOut	24.04.2023 11:05	true
11	0505653600	CheckIn	24.04.2023 11:05	true
12	0505653600	CheckIn	24.04.2023 11:00	false
13	2631825417	Doktor	24.04.2023 10:50	false

Obr. 12: Návrh zobrazení pro administrátora

4.2 Uživatelské rozhraní pomocí PyQt5 a Qt Designer

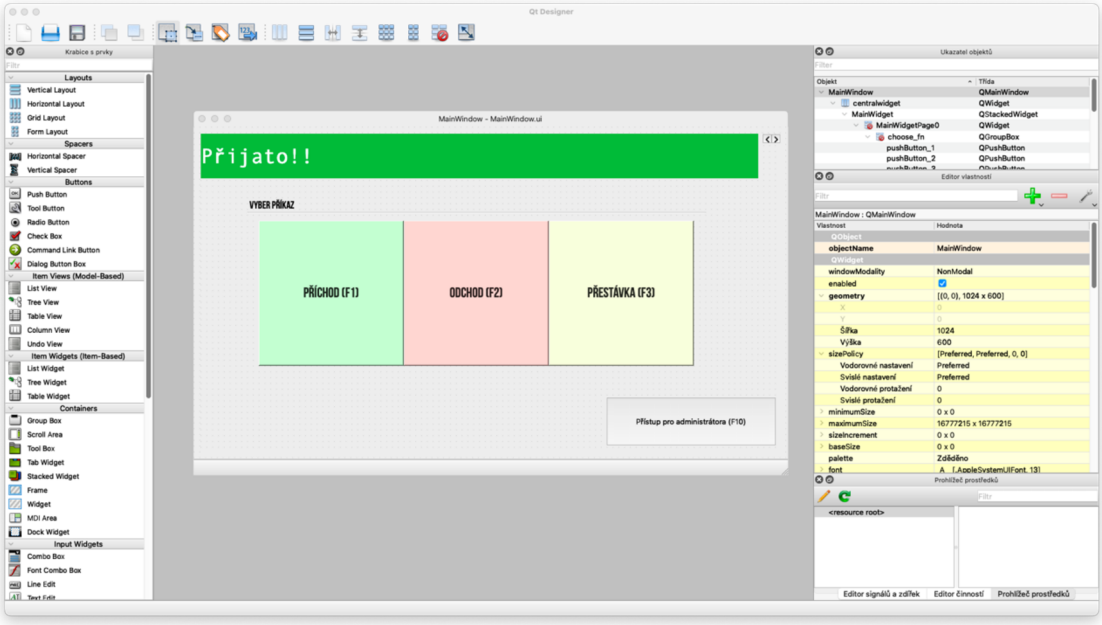
Pro tvorbu aplikace byla zvolena vývojová platforma (framework) PyQt5. PyQt vychází z C++ knihovny Qt přizpůsobené pro využití s jazykem Python. Existuje i druhá varianta Qt pro Python, a to PySide. Rozdíl mezi nimi ovšem není příliš velký. PyQt vzniklo mnohem dříve než PySide. PySide byla společností Qt přijata jako oficiální verze pro Python, nicméně PyQt je stále také plně podporováno.

Qt je nástrojem pro tvorbu uživatelských rozhraní (widget toolkit), který umožňuje vytvářet GUI aplikace pro více platforem, včetně Windows, macOS, Linux a Android, s použitím stejného zdrojového kódu. Qt byl založen Eirikem Chambe-Engem a Haavardem Nordem v roce 1991 a v roce 1994 byla založena první společnost Qt Company Trolltech. V současné době je Qt vyvíjen společností The Qt Company a pravidelně se aktualizuje, přidává nové funkce a rozšiřuje podporu pro mobilní zařízení a více platforem. [18]

Pro vytvoření grafického rozhraní byl zvolen Qt Designer. Qt Designer je nástroj pro rychlou tvorbu uživatelských rozhraní s využitím widgetů z Qt. Poskytuje jednoduché rozhraní pro přetažení a umístění komponent, jako jsou tlačítka, textová pole, seznamy a další. Umožňuje také umožnuje velice snadno upravovat aplikaci v různých stylech a rozlišeních, či tvorbu nabídek a nástrojových lišt.

Velkou výhodou Qt Designer je jeho nezávislost na platformě a programovacím jazyce. Vytváří totiž soubory .ui ve formátu XML. Z nich se potom generuje uživatelské rozhraní. Obsah souborů .ui lze přeložit do kódu Python s pomocí nástroje pyuic5, což je nástroj příkazové řádky, který je součástí PyQt. Poté lze tento kód Pythonu použít aplikacích s uživatelským rozhraním. Druhou možností je možnost přímo číst soubory .ui v rámci kódu aplikace a načíst jejich obsah pro generování příslušného uživatelského rozhraní. Velkou výhodou tohoto přístupu je jednodušší možnost úpravy uživatelského rozhraní během celého vývoje aplikace. Z tohoto důvodu jsem se rozhodl využít právě tento způsob. [19, 20]

Qt Designer je součástí integrovaného vývojového prostředí (IDE) od Qt Creator, který je oficiálním vývojovým prostředím pro platformu Qt, a který umožňuje vývojářům vytvářet, testovat a ladit aplikace na jednom místě. Pro náš účel tvorby uživatelského rozhraní můžeme využít pouze modul Qt Designer, který je možné stáhnout samostatně. Na obrázku 13 lze vidět celkové rozhraní aplikace Qt Designer.

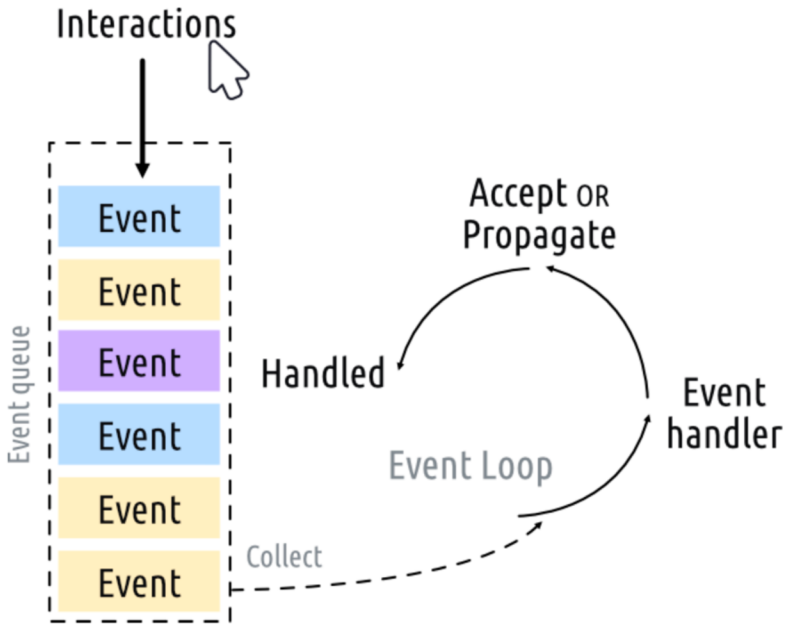


Obr. 13: Rozhraní aplikace Qt Designer

4.3 Návrh struktury programu

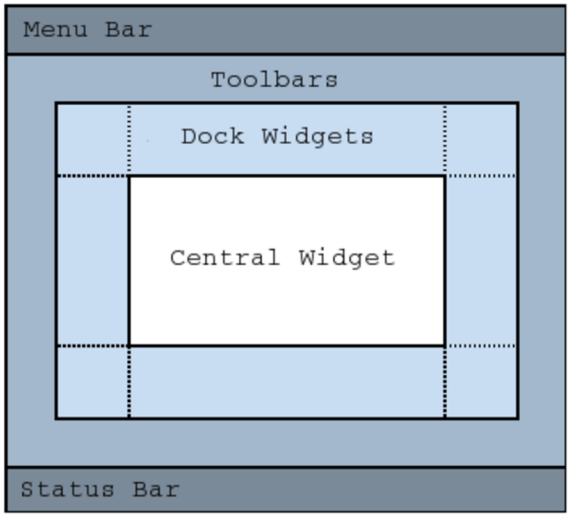
Jádrem každé Qt aplikace je třída QApplication. Každá aplikace potřebuje právě jednu instanci QApplication pro svůj běh. Tento objekt drží smyčku událostí (event loop), která řídí veškerou interakci uživatele s grafickým rozhraním.

Každá interakce s aplikací – ať už stisk klávesy, kliknutí myší nebo pohyb myši – vyvolá událost, která je umístěna do fronty. Tato fronta je společně se smyčkou událostí zobrazena na obrázku 14. Fronta událostí se kontroluje při každé iteraci. Pokud je nalezena čekající událost, tak je předána společně s kontrolou nad aplikací konkrétnímu obslužnému programu (metodě) přiřazené pro danou událost. Obslužným programem je událost vyřešena a poté je předána kontrola zpět. V jedné aplikaci je možné obsluhovat pouze jednu smyčku událostí, což s sebou nese problém při skenování čtečky karet viz. kapitola 4.6.1. [18]



Obr. 14: Schéma smyčky událostí [18]

Dalším důležitým stavebním prvkem pro aplikaci je poté výchozí widget, který bude QApplication spouštět. I když je možné využít v podstatě jakýkoliv widget včetně tlačítka, pro praktickou tvorbu aplikace je vhodné využít QMainWindow. Poskytuje totiž možnosti pro tvorbu standartních oken rozhraní aplikace, která obsahují více dalších widgetů. Lze přidávat nástrojové lišty QToolBar, dokovací widgety QDockWidget, nabídku QMenuBar či stavový řádek QStatusBar. QMainWindow má definované rozložení. Rozložení obsahuje takzvaný „Central widget“, který lze obsadit jakýmkoliv typem widgetu a který je povinnou součástí QMainWindow. Obrázku 15 lze vidět příklad tohoto rozložení. [21]



Obr. 15: Příklad rozložení QMainWindow[21]

V programu Qt Designer bylo tedy vytvořeno okno QMainWindow. Pro možnost zobrazení více stránek, jako je hlavní stránka či nastavení administrátora byl zvolen widget QStackedWidget, který je právě nastaven jako „Central widget“. Pro naši aplikaci je zapotřebí vytvořit 3 samostatné stránky. Hlavní stránku, kde může uživatel zapisovat příchody či odchody. Poté výše zmíněnou stránku pro administrátora a také stránku s výzvou pro přiložení čipu administrátora pro přístup do nastavení. Qt Designer pak uloží XML soubor MainWindow.ui, který importujeme v kódu aplikace.

Dále byl vytvořen python program. Jak lze vidět v ukázce kódu výpis 5 obsahuje novou třídu MainWindow, která je podtřídou QtWidgets.QMainWindow. V ní pomocí metody uic.loadUi připojíme ke třídě uživatelské rozhraní. Tato třída je pak spouštěna v okně aplikace.

Výpis 5: Struktura kódu aplikace

```
from PyQt5 import QtWidgets, uic
import sys


class MainWindow(QtWidgets.QMainWindow):
    def __init__(self, *args, obj=None, **kwargs):
        super(MainWindow, self).__init__(*args, **kwargs)
        uic.loadUi('MainWindow.ui', self)


if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec())
```

4.4 Signály a sloty

Abychom pochopili, jakým způsobem mezi sebou jednotlivé části aplikace komunikují je nejdříve nutné vysvětlit dva základní pojmy, které Qt pro tuto komunikaci využívá. Jedná se o systém signálů a slotů.

Signály jsou oznámení vysílaná widgety poté, co je provedena nějaká akce. Může se jednat například o stisknutí tlačítka. Slot je potom python funkce. Pokud je signál propojen s konkrétním slotem, pak je daný slot (tzn. příslušná metoda) zavolán v případě, že je daný signál vysílán. Část kódu, která vysílá signál žádným způsobem neověřuje, jestli byl signál přijat a zda je využíván. Signál může být propojen s více sloty, a naopak slot může být propojen s více signály. Signál je atribut třídy. Proto se pro propojení využívají tyto základní metody: *.connect()*, *.disconnect()* a *.emit()*. V ukázce kódu 6 lze vidět propojení tlačítek na pomocí signálu a slotu. [22, 23]

Výpis 6: Příklad propojení tlačítka s metodou pomocí signálu a slotu

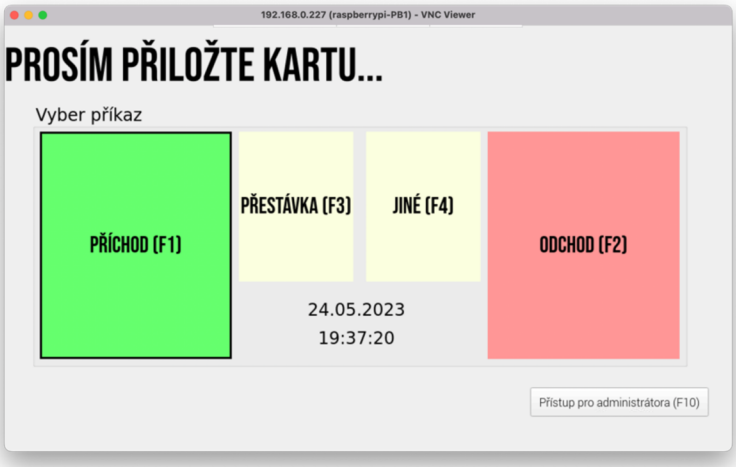
```
class MainWindow(QtWidgets.QMainWindow):
    def __init__(self, *args, obj=None, **kwargs):

        self.pushButton_1.clicked.connect(self.button_pressed)
        self.pushButton_2.clicked.connect(self.button_pressed)


    def button_pressed(self, checked):
        button = self.sender()
        if checked:
            if button.objectName() == "pushButton_1":
                self.Request = "CheckIn"
```

4.5 Hlavní stránka aplikace

Jak již bylo zmíněno, tato výchozí stránka aplikace, jejíž okno lze vidět na obrázku 16 slouží pro zápis příchodů a odchodů. Obsahuje tlačítka (QPushButton), která při stisknutí uživatelem nastaví, jaký příkaz odeslat.



Obr. 16: Hlavní okno aplikace

Tlačítko je propojeno se slotem definovaným metodou button_pressed. Tato metoda je společná pro všechna tlačítka definující status příchod, odchod či přestávku. Nejdřív se zkontroluje, které tlačítko bylo zmáčknuto, poté se nastaví do globální proměnné „Request“ jaký požadavek uživatel stisknul. Dále je také nutno nastavit stav ostatních tlačítek jako nezakliknutý (unchecked) a nastavit správný vzhled tlačítka – tedy zobrazit u zvoleného tlačítka rámeček. U ostatních je nutné rámeček vždy odstranit. Jak lze vidět v ukázce výpisku kódu 7, nejdříve se v metodě nastaví nezakliknutý styl všem tlačítkům a poté nastavíme styl pouze u stisknutého tlačítka. Speciálním tlačítkem je „Jiné“, které nejdříve vyzve uživatele k zadání důvodu průchodu. To může být například odchod k lékaři.

Výpis 7: Ukázka z metody button_pressed

```
self.pushButton_2.setChecked(False)
self.pushButton_2.setStyleSheet("border :0px solid black;"
                                "font: 24pt Bebas Neue;"
                                "background-color: rgb(255, 149, 150)")
. . .

if button.objectName() == "pushButton_1":
    self.Request = "CheckIn"
    self.pushButton_1.setChecked(True)
    self.pushButton_1.setStyleSheet("border :3px solid black;"
                                    "font: 24pt Bebas Neue;"
                                    "background-color: rgb(101, 255, 109)")
```

4.6 Připojení čtečky karet

Principem fungování vyvíjené aplikace je čtení karet a čipů zaměstnanců. Pro čtení karet jsem se rozhodl pro opětovné využití knihovny evdev z kapitoly 3.4.1. Jelikož se dle výrobce naše konkrétní čtečka karet chová jako virtuální klávesnice, je možné ke čtečce přistupovat právě jako ke klávesnici pomocí linux input subsystému. Nyní ovšem nebude do systému zapsáno zmáčknutí klávesy, ale naopak budeme z klávesnice číst.

Jelikož se čtečka chová jako klávesnice, tak by v případě otevření libovolného textového pole v aplikaci a přiložení karty, bylo vyplněno číslo karty do tohoto textového pole. Jak z hlediska bezpečnosti, tak z hlediska praktičnosti je žádoucí tento jev vyloučit. Jak lze vidět v ukázce kódu níže, tak nejdříve proskenujeme připojená vstupní zařízení a vybereme právě to s názvem naší čtečky karet. Poté nastavíme na daném vstupním zařízení možnost GrabDevice. Tím se zajistí, že žádný jiný ovladač nemůže inicializovat stejné zařízení, a také se zabrání tomu, aby zařízení odesílalo události jinam než do naší aplikace. [24]

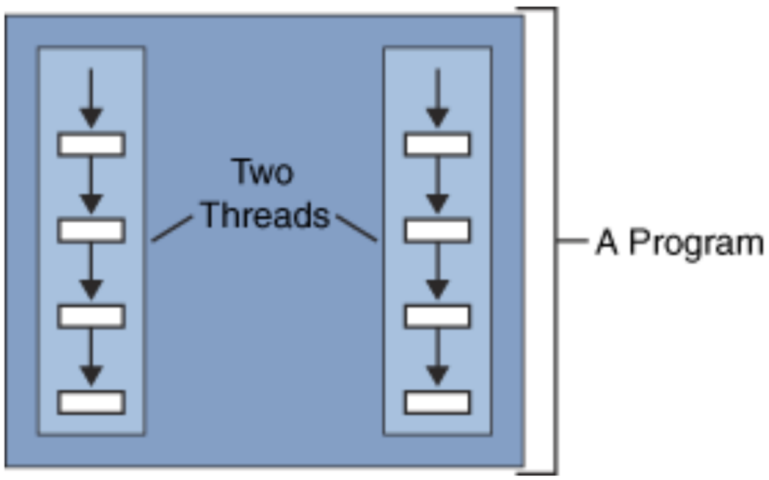
Výpis 8: Načtení čtečky karet

```
devices = [evdev.InputDevice(path) for path in evdev.list_devices()]
        for device in devices:
            if device.name == "IC Reader IC Reader":
                reader = device
        reader.grab()
```

4.6.1 Čtení karet s využitím QThread

Nyní se dostáváme k samotnému čtení čipů, případně karet. Algoritmus po inicializaci zařízení čeká na přiložení karty. Problémem nicméně je princip smyčky událostí popsaný v kapitole 4.3. Jak zde bylo zmíněno, smyčka událostí může být v rámci jedné QApplication pouze jedna. Pokud bychom tedy v kódu aplikace čekali na přiložení čipu zbytek uživatelské rozhraní by zamrzl. Respektive zásahy uživatele by čekali ve frontě.

Jednou z možností řešení tohoto problému je využití více vláken (Threads). Jedná se o metodu paralelního zpracování programů. Standartně probíhá provádění kódu sekvenčně. To znamená, že každý program má začátek, sekvenci vykonávání a konec. V průběhu běhu programu existuje vždy jediný bod vykonávání. Vlákno je podobné sekvenčním programům, jelikož má tedy také začátek, sekvenci a konec. I v průběhu běhu vlákna existuje vždy jediný bod vykonávání. Rozdíl je ovšem v tom, že vlákno samo o sobě není programem. Nemůže tedy běžet samo o sobě. Následující obrázek 17 ukazuje vztah programu a vláken [25].



Obr. 17: Schéma programu se dvěma vlákny

V PyQt tedy můžeme označit vlákno ve kterém je vytvořeno naše grafické rozhraní popsané v kapitole 4.3 jako hlavní. Dále poté můžeme vytvářet libovolné množství dalších, takzvaných pracovních vláken s využitím třídy QThread. Každé pracovní vlákno může mít vlastní smyčku událostí a podporovat mechanismus signálů a slotů PyQt pro komunikaci s hlavním vláknem [26].

V naší aplikaci bylo tedy vytvořeno nové vlákno tag_read, jehož úkol je právě čtení ze čtečky karet. Toto vlákno je poté pomocí signálu napojeno na metodu hlavního vlákna onPunch, které dále pracuje s načteným čipem či kartou. K této metodě se vrátíme v následující podkapitole.

V ukázce kódu výpis 9 je již algoritmus pro čekání na načtení čipu. Každý čip, který čtečka přečte generuje sekvenci jednotlivých číslic zakončených klávesou enter. Jelikož z principu funkce modulu evdev můžeme číst pouze jednotlivé znaky na klávesnici,

vytvoříme algoritmus, který postupně zapíše řetězec s kódem čipu a po zaznamenání enteru vyšle signál s načteným kódem čipu do příslušného slotu.

Výpis 9: Čtení čipů

```
for event in reader.read_loop():
            if event.type == evdev.ecodes.EV_KEY and event.value == 1:
                if event.code != 28:
                    tag_list.append(str(event.code-2))
                else:
                    tag = "".join(tag_list)
                    self.signal.emit("RFID", tag)
                    tag_list = []
```

4.6.2 Zpracování načteného čipu

Po naskenování čipu následuje přímo odeslání průchodu do serverové části a zápis průchodu do lokální historie. Jak bylo zmíněno výše, po načtení RFID čipu je zavolán slot, tedy metoda hlavní třídy onPunch. Touto metodou jsou nejdříve uložena načtená data do vhodné struktury typu slovník. Dále je pak zkontrolováno, zda existuje soubor s historií a v případě že neexistuje, dojde k jeho vytvoření.

Dále se pak aplikace pokusí o odeslání požadavku na server. K tomuto slouží knihovna Requests [27]. Tato knihovna umožnuje velmi jednoduše odesílat standartní http požadavky mezi aplikacemi. Samotné http je protokol sloužící k získávání požadavků ze serverů. Počítá se s využitím architektury Klient-Server. Klientská aplikace je ta, se kterou uživatel reálně pracuje, tedy taková aplikace, která zobrazuje obsah. Serverová aplikace je poté program, který je spuštěn na pozadí, naslouchá a čeká na požadavek. Pro propojení kiosku se serverovou částí využijeme metodu POST. Tato metoda je využívána zejména k vytváření nových položek na straně serveru. [28]

Pokus o odeslání je vložen do výrazu try. Jedná se o takzvané zpracování výjimek neboli systémových chyb v programu. Může se totiž stát, že se knihovně requests nepodaří doručit zprávu na server. Důvodem může být například jeho aktuální nedostupnost. V tomto případě vrátí funkce vykonávající odeslání chybové hlášení, které je potřeba v programu zachytit a ošetřit.

Příkaz try, který se skládá ze dvou základních bloků – try a except. Příkaz, nejčastěji volání funkce, pro který chceme ošetřit výjimky je vložen do bloku try. V tomto bloku postupně procházíme jednotlivé řádky a objeví-li se výjimka, zastaví se provádění kódu a jsou provedeny příkazy v příslušné větvi except. Ve výpisu 10 je vypsána část metody onPunch, která odesílá data na server. [29]

Výpis 10: Metoda pro odeslání dat na server

```
#odeslání dat na server
error = False
ip = self.Server + "/kiosk1"
try:
    requests.post(ip, json={"state" : data["state"], "user": data["user"],
                  "timedate": data["timedate"] }, timeout = 3)
except requests.exceptions.ConnectTimeout:
    print("Connection TimedOut!")
    error = True
except requests.exceptions.ConnectionError:
    print("Connection Erorr!")
    error = True

#zobrazení hlášky pro uživatele o odeslání průchodu, či chybě
if not error:
    data["send"] = "true"
    self.label_page0_accept.setStyleSheet(
                "background-color: rgb(73, 186, 70);"
                "color: rgb(255, 255, 255);"
                "font: 36pt Andale Mono;")
    self.label_page0_accept.setText("Přijato... " + str(tag))
else:
    self.label_page0_accept.setStyleSheet("background-color: rgb(255,54,80);"
                                          "color: rgb(255,255,255);"
                                          "font: 20pt Andale Mono;")
    self.label_page0_accept.setText("Chyba spojení!!! Tag " + str(tag) +
                                    " byl zaznamenán offline.")
```

Pokud tedy knihovna request nevrátí žádnou chybu, tak můžeme prohlásit, že informace byla v pořádku odeslána na server a zobrazíme uživateli hlášku s potvrzením. V opačném případě vypíšeme uživateli oznámení o chybě. Hlášky zobrazíme pomocí QLabel, zobrazené po dobu 2 sekund. V obou případech nicméně dojde k uložení do lokální historie. Ve výpisu 11 je příklad souboru s vypsanými průchody. Jedním z ukládaných parametrů přístupu je i logická hodnota (True/False) výsledku odeslání daného průchodu na server.

Výpis 11: Soubor s historií

```
{
"state": "CheckIn",
"user": "2631825417",
"timedate": "20.05.2023 19:45",
"send": "true"
}
{
"state": "CheckIn",
"user": "9208732834",
"timedate": "20.05.2023 19:46",
"send": "false"
}
```

Lokání historii je poté možné zobrazit na stránce pro administrátora. Pro zobrazení tabulky s historií průchodů byl využit QTableView. Tento widget vygeneruje při každém otevření stránky se zobrazením administrátora tabulku ze souboru log.txt. Data jsou seřazena od nejnovějších. Jsou tedy čteny jednotlivé řádky a jsou postupně zapisovány do tabulky. Jelikož je soubor v textovém formátu, kde na každém řádku je uložen právě jeden zápis historie ve formátu JSON, je nutné nejdříve textový vstup převést do vhodného formátu. Tento převod je opět ošetřen pomocí výrazu Try. V případně chybného řádku (prázdný, či s formátem jiným než JSON), je tento řádek přeskočen. Algoritmus je vypsán ve výpisu kódu 10.

Výpis 12: Algoritmus vytvoření tabulky historie

```
self.tableWidget.setRowCount(0)
#vytvoření tabulky historie
with open(fle, "r") as f:
    line = f.readline()
    while line:
        try:
            history = json.loads(line)
            self.tableWidget.insertRow(0)
            self.tableWidget.setItem(0, 0, QTableWidgetItem(history["user"]))
            self.tableWidget.setItem(0, 1, QTableWidgetItem(history["state"]))
            self.tableWidget.setItem(0, 2, QTableWidgetItem(history["timedate"]))
            self.tableWidget.setItem(0, 3, QTableWidgetItem(history["send"]))
        except json.decoder.JSONDecodeError:
            pass
        finally:
            line = f.readline()
```

4.7 Bezpečnost kiosku

Přístupové systémy obecně musí zajistit provozovateli dostatečný stupeň zabezpečení před případným pokusem o nabourání do systému. Jedním z mechanismů je znemožnění jednoduše aplikaci kiosku ukončit standartním uživatelem. Jelikož aplikace běží v režimu celé obrazovky a má zakázána systémová tlačítka pro ukončení aplikace, není možné jednoduše aplikaci ukončit. Pouze na stránku administrátora, zabezpečenou čipem administrátora, bylo přidáno tlačítko pro ukončení aplikace. V této fázi je čip administrátora pouze jeden. V této věci je prostor pro budoucí vylepšení aplikace s přidáním správy administrátorů systému.

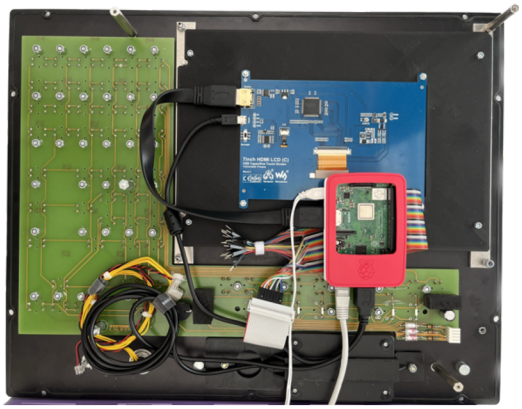
Druhým způsobem napadení může být pokus připojit vlastní flash-disk či myš a pokus o spuštění vlastního programu, který by převzal kontrolu nad kioskem. Momentálně je tento bezpečnostní problém řešen pomocí fyzického zabránění útočníkovi připojit své zařízení. RaspberryPi je schováno pod krytem přístroje a pro připojení cizího zařízení by bylo nutné kiosek rozebrat. Bylo by možné také kontrolovat jaká zařízení jsou připojena a případně blokovat jejich použití.

4.8 Výsledné zapojení a spuštění kiosku

V práci byla navržena multiplatformní kiosková aplikace. Pomocí nástroje PyInstaller byla zkompilována do spustitelného souboru pro systém linux, který je kromě samotného spustitelného souboru vytvořen složkami s veškerými potřebnými knihovnami. Předpokládá se nastavení aplikace jako výchozí automaticky spuštěné po startu systému. Na obrázcích 18, 19 a 20 si lze prohlédnou výsledný produkt. V příloze k této bakalářské práci je přiloženo video s ukázkou funkce aplikace i zdrojové kódy aplikace.

Obr. 18: Výsledný přední panel kiosku

Obr. 20: Kiosek z bočního pohledu



Obr. 19: Vnitřní zapojení kiosku

4.9 Další možná vylepšení

Jedním z vylepšení, které by zlepšilo uživatelský zážitek z aplikace by byla stránka s možností načtení průchodů uložených na serveru. Zodpovědný vedoucí by poté měl možnost zobrazit historii průchodů přímo na kiosku. Vhodným vylepšením by také byla možnost pokusit se znovu odeslat průchody, které se díky chybě odeslat nepodařilo.

5 ZÁVĚR

Předložená práce rozebírá přestavbu průmyslového počítače, který byl v ne příliš dobrém stavu vyřazen z laboratoře. Práce se nejdříve zabývá historií průmyslových počítačů a také problematikou elektronického odpadu. Vzhledem ke špatnému stavu jednotlivých komponent bylo možné z původního počítače využít pouze přední panel s odolnou hardwarovou klávesnicí a prostorem pro display. Původní základní deska s procesorem byla nahrazena mikropočítačem RaspberryPi.

Velká část práce se věnuje návrhem ovladače pro propojení hardwarové klávesnice. Jelikož kontrolér, který zajišťoval komunikaci původního počítače s klávesnicí pomocí konektoru DIN, není přímo kompatibilní se zvolenou platformou RaspberryPi, dovolili jsme si navrhnout vlastní softwarový ovladač klávesnice s využitím GPIO. Vzhledem k neexistující dokumentaci, bylo nutné proměřit plošný spoj tlačítek a navrhnout propojení klávesnice s GPIO piny. Samotný ovladač byl vytvořen v programovacím jazyce Python jako nová služba běžící na pozadí při spuštění počítače. S využitím vstupního modulu evdev poté zpracovává vstupní data z klávesnice. V této části práce je rozebrán princip funkce Linux input substému a systémového démona systemd, které jsou nezbytné pro funkci navrhnutého ovladače. Výhledově by bylo vhodné převést program do jazyka C zejména z důvodu snížení zatížení procesoru.

Poslední část práce navrhuje jednoduchou kioskovou aplikaci, jejíž cílem je evidence docházky zaměstnanců s využitím RFID čtečky karet. Pro tvorbu grafického rozhraní bylo využito multiplatformního nástroje pro tvorbu aplikací PyQt5. K návrhu uživatelskou rozhraní byl využit software Qt Designer. Aplikace využívá programovacího jazyka Python. Pro čtení uživatelů byla přidána ke kiosku čtečka karet a pro její propojení se softwarem bylo opětovně využito knihovny evdev, stejně jako u klávesnice. Čtení ze čtečky karet využívá další vlákno programu, tedy paralelního zpracování. Kiosek je napojen na serverovou část pomocí HTTP requests. Video ukázka použití aplikace v praxi je součástí příloh.

I přes nefunkční původní komponenty počítače, díky čemuž nebylo možné provést přestavbu původní základní desky s procesorem a bylo nutné nahradit tyto komponenty RaspberryPi, byl využit kvalitní odolný přední panel s klávesnicí, který může dále plnit svou funkci.

SEZNAM POUŽITÉ LITERATURY

[1] Beijer Electronics Acquires Elektronik-Systeme Lauer. *Fierce Electronics - Sensors* [online]. 2007 [vid. 2023-05-17]. Dostupné z: https://www.fierceelectronics.com/components/beijer-electronics-acquires-elektronik-systeme-lauer

[2] ZHANG, Peng. Industrial computers. In: *Advanced Industrial Control Technology* [online]. B.m.: Elsevier, 2010 [vid. 2023-03-21], s. 345–359. Dostupné z: doi:10.1016/b978-1-4377-7807-6.10009-9

[3] RAUTELA, Rahul, Shashi ARYA, Shilpa VISHWAKARMA, Jechan LEE, Ki Hyun KIM a Sunil KUMAR. E-waste management and its effects on the environment and human health. *Science of The Total Environment* [online]. 2021, **773**, 145623 [vid. 2023-05-10]. ISSN 0048-9697. Dostupné z: doi:10.1016/J.SCITOTENV.2021.145623

[4] BALDÉ, C P, E D'ANGELO, V LUDA, O DEUBZER a R KUEHR. Global Transboundary E-waste Flows Monitor 2022. nedatováno.

[5] PARVEZ, Sarker M., Farjana JAHAN, Marie Noel BRUNE, Julia F. GORMAN, Musarrat J. RAHMAN, David CARPENTER, Zahir ISLAM, Mahbubur RAHMAN, Nirupam AICH, Luke D. KNIBBS a Peter D. SLY. Health consequences of exposure to e-waste: an updated systematic review. *The Lancet Planetary Health* [online]. 2021, **5**(12), e905–e920 [vid. 2023-05-24]. ISSN 25425196. Dostupné z: doi:10.1016/S2542-5196(21)00263-1/ATTACHMENT/2901E9D1-0D90-4541-9FBC-07D39CCADFBC/MMC1.PDF

[6] UPTON, Eben a Gareth HALFACREE. *Raspberry Pi User Guide*. 4th Edition. 2016.

[7] *Difference Between SOC (System on Chip) and Single Board Computer*.

[8] Raspberry Pi 3 Model B+. *RaspberryPi official Product page* [online]. [vid. 2023-05-22]. Dostupné z: https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus/

[9] *Raspberry Pi Documentation - hardware* [online]. [vid. 2023-05-17]. Dostupné z: https://www.raspberrypi.com/documentation/computers/raspberry-pi.html

[10] *PiJuice Documentation* [online]. [vid. 2023-05-22]. Dostupné z: https://github.com/PiSupply/PiJuice

[11] DUDÁČEK, Karel. *Maticové snímání klávesnice* [online]. [vid. 2023-05-24]. Dostupné z: http://home.zcu.cz/~dudacek/PZ/klavesnice.pdf

[12] BUTLER, Sydney. *What Is GPIO, and What Can You Use It For?* [online]. 2022 [vid. 2023-05-22]. Dostupné z: https://www.howtogeek.com/787928/what-is-gpio/

[13] AWATI, Rahul. *What is an interrupt request (IRQ)* [online]. 2023 [vid. 2023-05-22]. Dostupné z: https://www.techtarget.com/whatis/definition/IRQ-interrupt-request

[14] *GPIO Interfaces — The Linux Kernel documentation* [online]. [vid. 2023-05-24]. Dostupné z: https://www.kernel.org/doc/html/latest/driver-api/gpio/intro.html#gpio-interfaces

[15] DMITRIY. *Linux Device Drivers* [online]. 2022 [vid. 2023-05-24]. Dostupné z: https://www.apriorit.com/dev-blog/195-simple-driver-for-linux-os

[16] PAVLÍK, Vojtěch. Linux Input Subsystem userspace API — The Linux Kernel documentation. *The Linux Kernel documentation* [online]. 2001 [vid. 2023-05-11]. Dostupné z: https://docs.kernel.org/input/input.html

[17] NEGUS, Christopher. Starting and Stopping Services. In: *Linux® Bible* [online]. B.m.: Wiley, 2020, s. 369–402. Dostupné z: doi:10.1002/9781119578956.ch15

[18] FITZPATRICK, Martin. *Create GUI Applications with Python & Qt5*. 2022.

[19] RAMOS, Leodanis. *Qt Designer and Python* [online]. [vid. 2023-05-25]. Dostupné z: https://realpython.com/qt-designer-python/

[20] Qt Designer Manual. *Qt Documentation* [online]. [vid. 2023-05-25]. Dostupné z: https://doc.qt.io/qt-6/qtdesigner-manual.html

[21] QMainWindow — Qt for Python. *Qt Documentation* [online]. [vid. 2023-05-18]. Dostupné z: https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QMainWindow.html#more

[22] Support for Signals and Slots. *Riverbank Computing Limited - PyQt Documentation* [online]. [vid. 2023-05-18]. Dostupné z: https://www.riverbankcomputing.com/static/Docs/PyQt5/signals_slots.html

[23] FITZPATRICK, Martin. *PyQt5 Signals, Slots and Events* [online]. [vid. 2023-05-18]. Dostupné z: https://www.pythonguis.com/tutorials/pyqt-signals-slots-events/

[24] HØGSBERG, Kristian a Peter HUTTERER. EVDEV. *The X.Org project* [online]. [vid. 2023-05-18]. Dostupné z: https://www.x.org/releases/X11R7.6/doc/man/man4/evdev.4.xhtml

[25] *Threads: Doing Two or More Tasks At Once* [online]. [vid. 2023-05-18]. Dostupné z: https://www.iitk.ac.in/esc101/05Aug/tutorial/essential/threads/definition.html

[26] RAMOS, Leodanis. *Use PyQt's QThread to Prevent Freezing GUIs* [online]. [vid. 2023-05-22]. Dostupné z: https://realpython.com/python-pyqt-qthread/

[27] REITZ, Kenneth. *Requests: HTTP for Humans™ — Requests 2.31.0 documentation* [online]. [vid. 2023-05-23]. Dostupné z: https://requests.readthedocs.io/en/latest/

[28] GARZON, Camila. *HTTP Request Methods – Get vs Put vs Post Explained with Code Examples* [online]. 2022 [vid. 2023-05-23]. Dostupné z: https://www.freecodecamp.org/news/http-request-methods-explained/

[29] PECINOVSKÝ, Rudolf. *Python : kompletní příručka jazyka pro verzi 3.10*. První vydání. Praha: Grada Publishing, 2021. Knihovna programátora (Grada). ISBN 978-80-271-3442-5.

[30] *LAUER VS386 ; VS486 C33002 INDUSTRIAL PC - Samm Electronics* [online]. [vid. 2023-05-25]. Dostupné z: https://samm-electronics.com/uk/lauer/VS386VS486C33002-used

[31] *PiJuice HAT - A Portable Power Platform For Every Raspberry Pi* [online]. [vid. 2023-05-22]. Dostupné z: https://uk.pi-supply.com/products/pijuice-standard

SEZNAM ZKRATEK, SYMBOLŮ, OBRÁZKŮ A TABULEK

5.1 Seznam zkratek

PLC	Programmable logic controller
IoT	Internet of things
PCB	Printed circut board
RAM	Random-access memory
CPU	Central processing unit
HTTP	Hypertext Transfer Protocol
GUI	Graphic user interface
GPIO	General-purpose input/output
SBC	Single board computer
SIM	Subscriber identity module
USB	Universal Serial Bus
UPS	Uninterruptible Power Supply
IRQ	Interrupt Request
SoC	System on Chip
RFID	Radio-frequency identification
ISIC	International Student Identity Card
XML	Extensible Markup Language
JSON	JavaScript Object Notation

5.2 Seznam obrázků

5.3 Seznam výpisů kódu

5.4 Seznam tabulek

SEZNAM PŘÍLOH