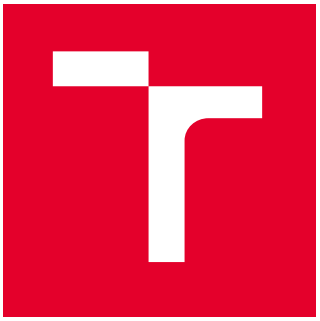


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ

FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY

INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

PROPOJENÍ RTOS S GPOS V PRŮMYSLOVÉ VYUŽITÍ

CONNECTING RTOS WITH GPOS IN INDUSTRIAL USE

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — **Bc. Tomáš Thér**

AUTHOR

VEDOUCÍ PRÁCE — **Ing. et Ing. Stanislav Lang, Ph.D.**

SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav automatizace a informatiky
Student:	**Bc. Tomáš Thér**
Studijní program:	Aplikovaná informatika a řízení
Studijní obor:	bez specializace
Vedoucí práce:	**Ing. et Ing. Stanislav Lang, Ph.D.**
Akademický rok:	2023/24

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 diplomové práce:

Propojení RTOS s GPOS v průmyslové využití

Stručná charakteristika problematiky úkolu:

Průmyslové řídicí systémy jsou neodmyslitelnou součástí automatizace. Důležitými zástupci zařízení v této oblasti jsou průmyslové počítače, které se od běžných liší zejména hardwarovým provedením a způsobem řízení programů (tedy fungováním operačního systému). V práci se student bude muset seznámit zejména s problematikou využití operačních systémů reálného času (RTOS) a jejich případnou kombinací/komunikací s operačními systémy pro běžné použití (GPOS). V praktické části student nastuduje a zprovozní produkt exOS společnosti B&R Industrial Automation, pomocí něhož zajistí komunikaci dvou operačních systémů (RTOS Automation Runtime a GPOS Linux). Výhody takového spojení pak student ukáže na vhodně zvoleném příkladu.

Cíle diplomové práce:

Nastudujte problematiku operačních systémů, stručně shrňte hlavní rozdíly a aplikační využití RTOS a GPOS.
Proveďte průzkum v oblasti možností využití kooperace obou typů systémů, uveďte příklady dostupných produktů na trhu.
Zaměřte se na produkt exOS společnosti B&R.
Zprovozněte pomocí exOS komunikaci mezi RTOS (Automation Runtime) a GPOS (Linux).
Otestujte komunikaci na reálném zařízení (na modelové aplikaci dle vlastního výběru).
Zhodnoťte dosažené výsledky.

Seznam doporučené literatury:

SROVNAL, Vilém. Operační systémy pro řízení v reálném čase. Ostrava: Vysoká škola báňská - Technická univerzita, 2003. ISBN 80-248-0503-0.

Automation Help: exOS [Elektronická dokumentace]. Eggelsberg, Austria. B&R Industrial Automation, 2023.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2023/24

V Brně, dne

L. S.

Ing. Pavel Heriban, Ph.D. ředitel ústavu	doc. Ing. Jiří Hlinka, Ph.D. děkan fakulty

ABSTRAKT

Tato diplomová práce zkoumá propojení operačních systémů reálného času (RTOS) a systémů pro obecné využití (GPOS) v průmyslové automatizaci. RTOS jsou klíčové pro deterministické řízení kritických procesů, zatímco GPOS poskytují širokou škálu nástrojů. Práce se zaměřuje na produkt exOS od společnosti B&R, jehož vlastnosti, instalace a vývojové nástroje jsou podrobně popsány. Nakonec je vytvořena modelová aplikace, která pro reálný průmyslový projekt implementuje databázi plasmové řezačky.

ABSTRACT

This thesis explores the integration of real-time operating systems (RTOS) and general-purpose operating systems (GPOS) in industrial automation. RTOS are crucial for deterministic control of critical processes, while GPOS offer a wide range of tools. The thesis focuses on the exOS product from B&R, detailing its features, installation, and development tools. Finally a model application is created, implementing a database for a plasma cutter in a real industrial project.

KLÍČOVÁ SLOVA

Operační systém, Reálný čas, Hypervisor, Automation Studio, exOS, Databáze, MariaDB

KEYWORDS

Operating system, Real-time, Hypervisor, Automation Studio, exOS, Database, MariaDB

ÚSTAV AUTOMATIZACE A INFORMATIKY

2024

BIBLIOGRAFICKÁ CITACE

THÉR, Tomáš. *Propojení RTOS s GPOS v průmyslové využití*. Brno, 2024. Dostupné také z: `https://www.vut.cz/studenti/zav-prace/detail/157707`. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky, Vedoucí práce: Ing. et Ing. Lang Stanislav, Ph.D.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že tato diplomová práce je mým původním dílem, vypracoval jsem ji samostatně pod vedením vedoucího diplomové 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 následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků.

V Brně dne 21. 5. 2024 .....

Tomáš Thér

PODĚKOVÁNÍ

Děkuji svému vedoucímu Ing. et Ing. Stanislavu Langovi Ph.D. za pomoc a ochotu při vytváření této diplomové práce. Dále chci poděkovat Ing. Kamilu Růžičkovi a Ing. Michalu Vavríkovi za konzultace a cenné rady ohledně praktické části. Nakonec děkuji i mé rodině a přátelům kteří mě podporovali během celého studia.

OBSAH

1 ÚVOD

Operační systémy reálného času (RTOS) představují klíčový prvek v průmyslové automatizaci, kde se používají pro řízení kritických procesů vyžadujících deterministické chování. Operační systémy pro obecné využití (GPOS), které se používají v každodenním životě, se přímo pro tento účel nehodí, na druhou stranu ale nabízejí velké množství užitečných nástrojů, jež nemusí být vždy pro RTOS přístupné. Propojení těchto dvou odlišných systémů může být pro některé aplikace velmi výhodné. Tato diplomová práce se zabývá možnostmi a implementací takového propojení.

V kapitole číslo dva je teoreticky rozebrána problematika operačních systémů a stručně shrnuta jejich historie. Dále jsou představeny různé typy operačních systémů. Část kapitoly se zabývá procesy a jejich plánováním, což je jeden z nejpodstatnějších rysů operačního systému.

Třetí kapitola se zaměřuje na kooperaci RTOS a GPOS, obecně představuje problematiku a poté se zabývá produkty dostupnými na trhu, které tuto kooperaci umožňují.

Následující kapitola se zaměřuje na konkrétní produkt exOS od společnosti B&R. Popisuje jeho hlavní vlastnosti, instalaci a nástroje používané pro vývoj.

Další kapitola na základě skutečné průmyslové aplikace řeší implementaci databáze pro plazmovou řezačku s využitím exOS. Jsou zde popsány potřebné konfigurace a vytvořený software. Poslední kapitola diskutuje dosažené výsledky, možnosti pro zlepšení a produkt exOS.

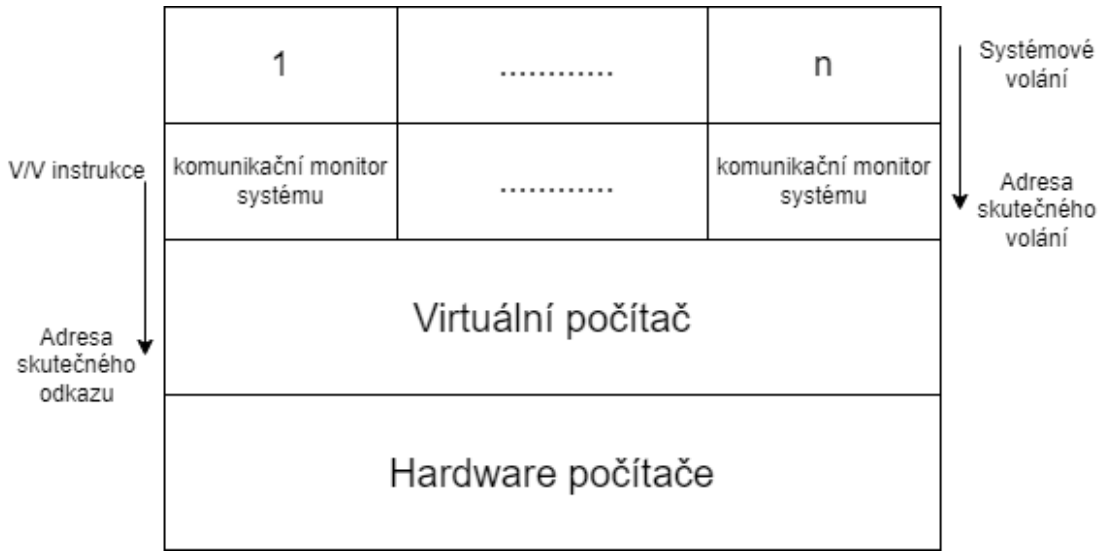
2 OPERAČNÍ SYSTÉMY

Hlavním účelem operačních systémů je organizace práce výpočetních systémů, na kterých pracují, což u prvních počítačů měla na starosti obsluha. S rychlým vzrůstem výkonnosti a složitosti výpočetních systémů přestalo být možné, aby operátor zastupoval všechny potřebné role při řízení počítače, aniž by omezil jeho rychlost. Tato skutečnost vedla k vývoji a nasazení operačních systémů umožňujících počítači samostatně provádět rozhodování, které musel předtím vykonávat člověk. [1]

Jedna z možných definic operačního systému ho popisuje jako softwarového správce prostředků dělící se na prostředky hardwarové a softwarové. Mezi hardwarové prostředky patří například procesor, operační paměť, grafická karta, disk pro ukládání dat, i vstupně výstupní prostředky jako myš, klávesnice, tiskárny a mnoho dalších. Softwarovými prostředky se rozumí procesy, paměťové regiony, soubory, grafické rozhraní, přehrávače multimédií apod. [2]

Důležitým pojmem je proces, který označuje instanci programu/aplikace. Tato instance je uložená v operační paměti, má přidělené prostředky, jakými jsou registry a oblasti paměti (například pro uložení místa zpracování při přerušení). Operační systém na základě požadavků procesu zajišťuje zpracování, získávání a ukládání dat. Paměťový region označuje souvisle adresovatelnou oblast paměti, která je přidělena jednomu či více procesům, které mají k této oblasti přímý přístup co se týče čtení i zápisu. Procesy tedy využívají paměťové regiony pro ukládání mezivýsledků, adres, strojového kódu procesu apod. V jednoduchých systémech jsou paměťovými regiony souvislé oblasti fyzické paměti, ve složitějších pak oblasti paměti virtuální, realizované množinou menších bloků fyzické paměti. [1, 2]

Operační systém v rámci správy vytváří virtuální počítač, což je jednotné rozhraní skrývající rozdíly hardwarového zpracování jednotlivých zařízení (obr. 1). Programy tak mohou fungovat na počítačích se stejným operačním systémem bez ohledu na rozdílné fyzické vybavení a zároveň se programátoři mohou vyvíjet aplikace pouze vzhledem k tomu, na jakém operačním systému poběží, což výrazně zjednodušuje psaní programů. Díky tomuto virtuálnímu rozhraní je v porovnání s fyzickým počítačem také velmi zjednodušené ovládání systému, kdy lze přes vcelku jednoduchá volání provádět komplexní akce, jako například přehrávání multimediálních souborů nebo navazování komunikace se vzdálenými zařízeními. V některých případech ale může toto rozhraní způsobit i nechtěné vedlejší účinky, jako omezení výkonu a flexibility systému, protože procesy mohou ztratit schopnost dostatečně rychlého přístupu k hardwarovým zařízením, která mohou v horším případě být pro procesy i zcela nepřístupná. Proto je ve většině operačních systémů i možnost využívání hardwarových prostředků na nižší úrovni, než je virtuální počítač, nebo i takřka přímého přístupu k těmto prostředkům. [1, 2, 3]



Obr. 1: Struktura operačního systému s virtuálním počítačem, převzato z [2]

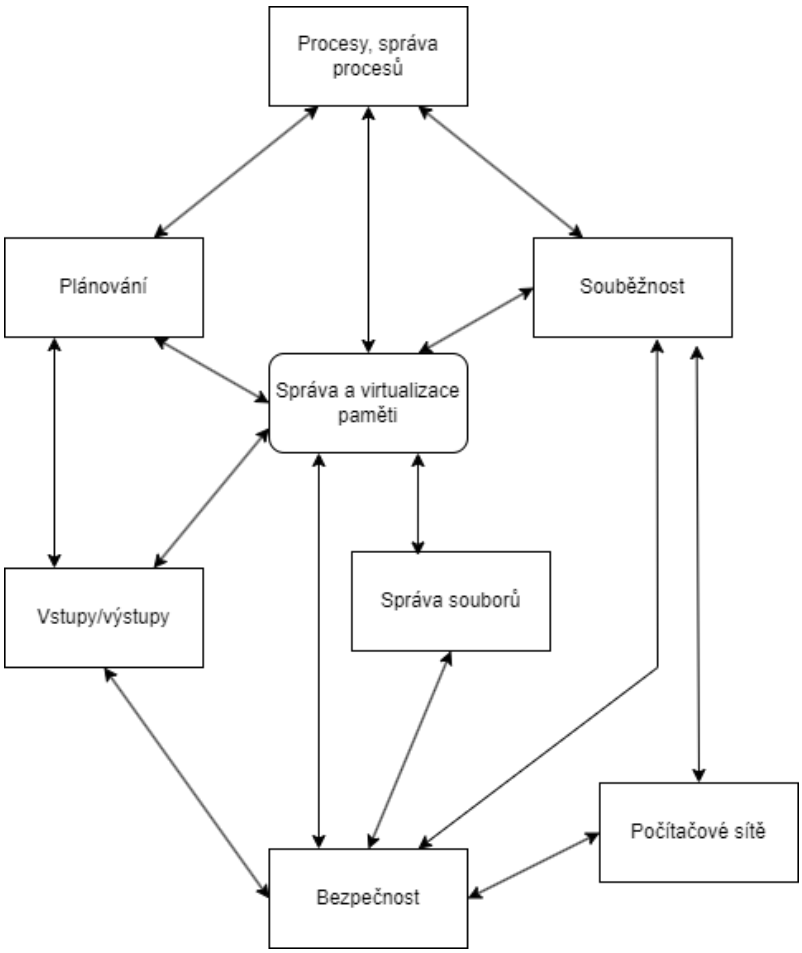
Dalším důležitým aspektem činnosti operačního systému je zaručení nezávislosti jednotlivých procesů, které tak mají z jejich pohledu iluzi, že žádné další procesy na systému neběží, i když může ve skutečnosti běžet několik procesů najednou a mohou dokonce i způsobovat přerušení ostatních. Každý proces má tedy vlastní virtuální počítač. Je nutné, aby operační systém předcházel kolizím procesů, které využívají stejné prostředky. Jsou ale i situace, kdy si mezi sebou jednotlivé procesy potřebují vyměňovat data, tudíž většina operačních systémů nabízí i možnosti meziprocesové komunikace, kterou operační systém umožňuje pouze na základě požadavku některého z procesů a tuto komunikaci také kontroluje. Přehled činností a zájmů operačního systému je zobrazen na obr. 2. [2]

2.1 Vymezení operačního systému

Operační systém se skládá z množiny rutin seskupených v několika vrstvách, přičemž nejnižší vrstva přímo přistupuje k hardwaru, další vrstva přímo přistupuje k nejnižší vrstvě atd., až do nejvyšší vrstvy. Není pevně stanoveno, které vrstvy ještě patří pod operační systém, většinou se ale používají následující vymezení. [2]

Kernel

Jádro (anglicky kernel) operačního systému je nejnižší vrstvou s rutinami bezprostředně přistupujícími k fyzickému vybavení počítače. Kernel poskytuje základní virtualizaci, souborový systém a prostředky pro meziprocesovou komunikaci. Kernel je v operační paměti přítomen takřka po celou dobu spuštění počítače, bývá



Obr. 2: Zájmy operačního systému, převzato z [3]

modulárně rozdělen na několik částí, které jsou z patřičných souborů podle potřeby načítány do paměti. [2]

Systémové procesy

Rutiny kernelu nejsou většinou samostatnými procesy, ale vykonávají se jako části procesů. Pro určité funkce operačního systému ale platí, že se snadněji vykonávají jako jeden proces. Tyto funkce obstarávají tzv. systémové procesy, na jejichž funkci závisí běh kernelu a tím pádem i celého operačního systému, při nesprávné funkci systémového procesu tedy může dojít i k zastavení celého počítače. Tyto procesy mohou běžet na úrovni kernelu anebo mohou být řízeny dalšími programy a komunikovat s jádrem pomocí systémových volání. [2]

Minimální uživatelský operační systém

Předchozí dvě vrstvy pro reálné použití operačního systému nestačí, protože neposkytují možnost komunikace s uživateli. Komunikace je realizována takzvaným shellem (z anglického slova skořápka, označující vrchní vrstvu operačního systému).

Shell je buď textový v podobě příkazové řádky (CLI - command line interface) anebo zastoupen uživatelským grafickým rozhraním (GUI - graphical user interface) a umožňuje uživateli správu souborů a procesů, přehrávání multimédií, nástroje pro administrativu a programování, apod. S shellem se již dá hovořit o takzvaném minimálním uživatelském operačním systému, který ale může být pro konkrétní účely omezen i použitím pouze jedné aplikace, například u automatů na výdej nápojů. Do minimálního uživatelského operačního systému se často také počítají i různé systémové knihovny, pomocí kterých se volají systémové procesy různými programovacími jazyky. [2]

Balíčkový operační systém

Jako balíčkový operační systém je označován soubor aplikací, který dostane zákazník při koupi operačního systému. Tento soubor může zahrnovat minimální operační systém, ale i další uživatelské aplikace, což je časté u distribucí operačních systémů pro osobní počítače či mobilní telefony. [2]

2.2 Historie

Tato podkapitola stručně shrne historii operačních systémů z chronologického pohledu, který je oproti skutečnosti dosti zjednodušen. Jak již bylo výše naznačeno, vývoj operačních systémů šel ruku v ruce s vývojem počítačů a jejich rostoucím výkonem. První pokus o vytvoření digitálního počítače je přisuzován anglickému matematikovi Charlesu Babbagovi (1792-1871), který velkou část svého života zasvětil stavění svého analytického stroje. Tento stroj byl čistě mechanický, což bylo i důvodem, proč Babbage nikdy nedokázal zajistit jeho správnou funkci, tehdejší technologie totiž neumožňovaly dostatečnou přesnost výroby součástek, jakými byla například ozubená kola. Další výrazný pokrok ve výpočetní technologii přišel až během druhé světové války, která výzkum v této oblasti rapidně urychlila. Dále se o vývoji operačních systémů mluví v několika generacích. [4]

2.2.1 První generace (1945-1955)

Hlavními součástkami počítačů této doby byly vakuové trubice (elektronky) anebo relé. Neexistovaly žádné programovací jazyky, počítače se tedy programovaly v absolutním strojovém kódu anebo zapojováním elektrických obvodů. Velká většina úloh byla v podobě přímočarých výpočtů, jako například plnění tabulek pro hodnoty sinů a logaritmů. Operační systémy v této době taktéž neexistovaly, v principu jejich činnost vykonával operátor. Začátkem 50-tých let se práce operátorů o něco usnadnila díky děrným štítkům. [4]

2.2.2 Druhá generace (1955-1965)

Velká změna přišla se zavedením tranzistorů v polovině 50-tých let. Výpočetní systémy být spolehlivější, začaly se prodávat ve formě sálových počítačů. Programy byly psány v jazyce Assembly nebo Fortran a poté převáděny do zmíněných děrných štítků. Začaly se využívat tzv. dávkové operační systémy, které programy načítaly a vykonávaly po dávkách. Příklady takového operačního systému jsou FMS - Fortran Monitoring System a IBSYS od společnosti IBM. [4]

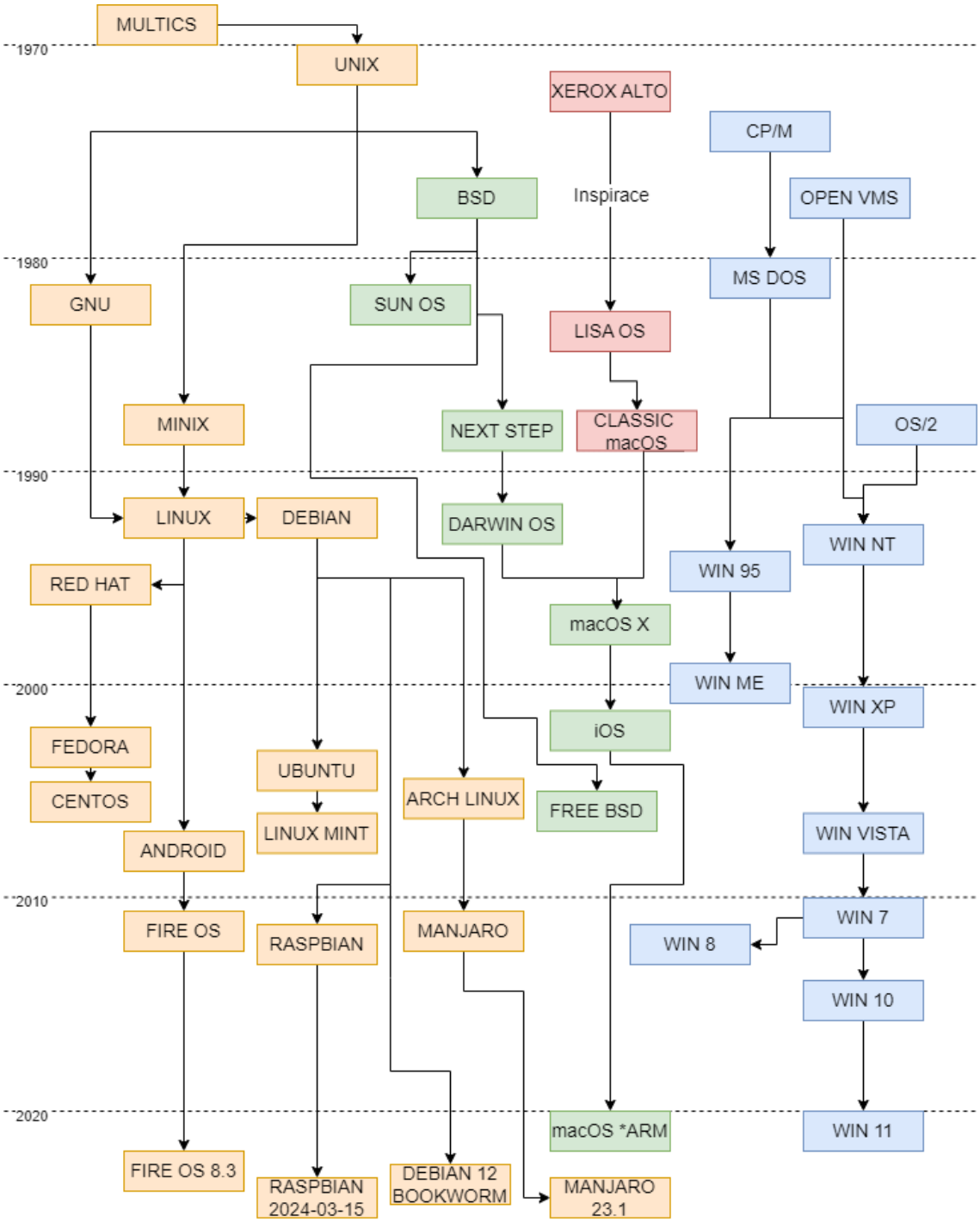
2.2.3 Třetí generace (1965-1980)

Operační systémy třetí generace přinesly mnoho průlomových inovací. Jednou z nich bylo multiprogramování. Každý program má alokovanou svou část paměti a systém může zpracovávat několik programů najednou. Například zatímco jeden program čeká na vstupní data, druhý program může mezitím využívat procesor, čímž se zvýrazně zvyšuje časová efektivita sálových počítačů, na kterých bylo přihlášeno často mnoho uživatelů naráz a každý z nich prováděl vlastní výpočty. [4]

Další novou technikou byl tzv. spooling. Do této doby když počítač dokončil výpočet daného programu, musel přijít operátor a nahrát nový program. Díky spoolingu ale bylo možné nahrát nové programy a zařadit je do fronty i před ukončením těch předchozích, po dokončení jednoho programu počítač začal automaticky zpracovávat ten další, který si načetl z disku. [4]

Výpočty na systémech této doby stále zabíraly řádově hodiny. Pro programátory bylo tedy velmi časově náročné debuggování. Aby se ušetřilo času stráveném čekáním například na program se syntaktickou chybou, byl vyvinutý tzv. timesharing. Na jeden počítač mohlo být najednou přihlášeno mnoho uživatelů, často ale pouze část z nich v jeden moment potřebovala provádět výpočty nebo zkoušet funkčnost programů. Procesor se tedy alokoval pouze pro ty programy, které právě potřebovaly být obsloužené. Programátorům se taky zpřístupnil online terminál, do kterého mohli vkládat krátké příkazy a dostávat na ně rychle odpovědi. Prvním podstatným operačním systémem využívajícím timesharing byl CTSS (Compatible Time Sharing System). [4]

Dalším významným operačním systémem této doby byl MULTICS, který podporoval práci 80 uživatelů najednou. Jeho komerční úspěchy byly smíšené, využívaly ho ale velké firmy jako například General Motors, Ford nebo například Narodní Bezpečnostní Agentura USA, které od MULTICSu odešly až během 90. let. Tento systém měl každopádně velký vliv na další vývoj, kdy Ken Thompson z Bellových laboratoří začal vytvářet MULTICS pro jednoho uživatele, ze kterého se časem stal UNIX, jenž je základem mnoha dalších operačních systémů. [4]



Obr. 3: Historie operačních systémů, podle [4] a [5]

2.2.4 Čtvrtá generace (1980-současnost)

Čtvrtou (a ve většině zdrojů poslední uvažovanou) generaci značí komerční rozšíření osobních počítačů a také zavedení grafického uživatelského rozhraní namísto pouhého textového terminálu. Vznikly dodnes velmi vlivné společnosti jakými jsou Microsoft a Apple a s nimi operační systémy jako MS-DOS, Windows a také rozšířený a otevřený Linux a jeho distribuce. Ve velmi zjednodušené verzi zachycuje

vývoj operačních systémů obr. 3, kde jsou zobrazeny vybrané systémy a jejich velmi zjednodušený vývoj. [4]

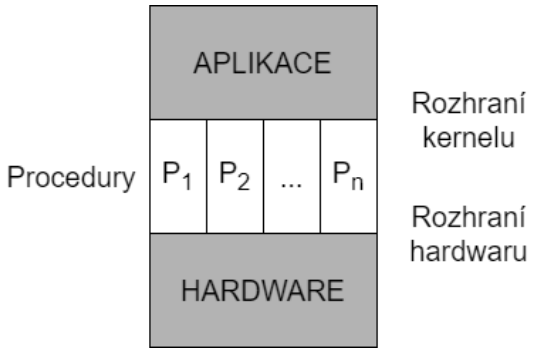
2.3 Struktura operačních systémů

Operační systémy je možné rozdělit několika různými způsoby. Jedním z nich může být například složitost. Ty nejjednodušší operační systémy zpracovávají pouze jeden program (časté u mikrokontrolérů), zatímco nejsložitější podporují běh mnoha programů najednou díky multiprogramování. [1]

Jednou ze základních klasifikací je struktura/architektura z hlediska interakcí jednotlivých rutin operačního systému. První skupinou jsou monolitické operační systémy, které mají propojení rutin velmi těsné. Dále existují tzv. hierarchické neboli víceúrovňové systémy a distribuované systémy, kde jsou rutiny rozděleny do jednotlivých modulů komunikujíc na základě modelu klient-server. Uvedené kategorie nejsou ale pevně odděleny a reálné operační systémy většinou leží na pomezí těchto architektur. [1, 2]

2.3.1 Monolitické operační systémy

Kernel monolitického operačního systému se skládá z množin navzájem minimálně komunikujících procedur (znázorněno na obr. 4). Tyto procedury poskytují rozhraní kernelu, které využívají aplikace k jejich volání, a na druhé straně procedury přímo komunikují s hardwarem. [1, 2]

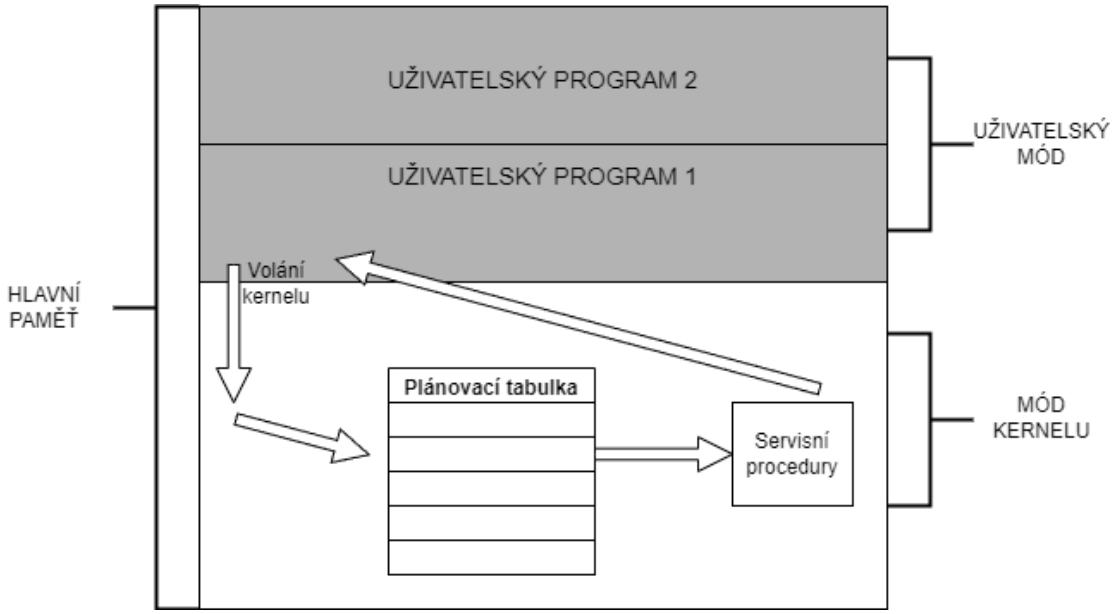


Obr. 4: Struktura monolitického systému, převzato z [2]

Tato architektura se objevuje u jednoduchých aplikací, většinou na jednodeskových mikropočítačích s jednouživatelskými systémy. Vzhledem k postupnému vývoji jednotlivých komplexnějších systémů v nich jsou poznat stopy této architektury, protože i složité operační systémy většinou vychází z jednodušších. Tato struktura je nahrazována těmi komplexnějšími, protože dnešní operační systémy jsou vyvíjeny týmy, jejichž spolupráce je u vývoje monolitického operačního systému velmi složitá. [2]

Proces může běžet v dvou různých režimech - uživatelském a v režimu jádra. V uživatelském režimu (také označovaném jako neprivilegovaném) smí proces využívat pouze vyhrazenou část operační paměti a má zakázáno provádění privilegovaných instrukcí, jakými jsou například zastavení procesoru nebo změna řídicích registrů. Proces v režimu jádra (privilegovaném režimu) vykonává jenom procedury jádra a může přistupovat k celé operační paměti i ke všem hardwarovým prostředkům. Změna režimu procesu nastává při přerušení a následně při návratu z něj. Přerušení může nastat na základě asynchronního signálu (poslaným z myši, klávesnice, disku apod.), použitím speciální instrukce na volání služby operačního systému, anebo při procesorových výjimkách nastávajících většinou v chybových stavech. [2]

Na obr. 5 je znázorněna činnost monolitického systému. Operační systém nejdříve vyhodnotí parametry volání kernelu, které určí příslušné systémové volání. V plánovací tabulce se označí procedura, která bude toto systémové volání vykonávat. V dalším kroku je tato procedura vyvolána, po jejím dokončení je ukončeno i systémové volání a uživatelský program poté opět přebírá řízení. [1]



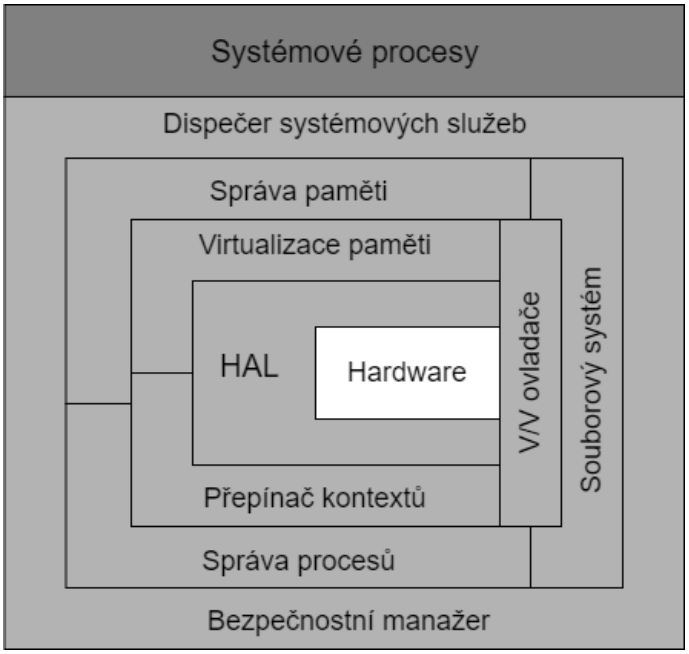
Obr. 5: Struktura služeb v monolitickém systému, převzato z [2]

V monolitickém systému se rozlišují 3 typy procedur. Hlavní procedura, kterou volá uživatelský program, ta volá servisní procedury, jež využívají služeb poskytovaných obslužnými procedurami. [1]

2.3.2 Hierarchické operační systémy

Hierarchické operační systémy jsou tvořeny několika vrstvami, které postupně obklopují hardware, zařizují jeho abstrakci a poskytují rozhraní pro vyšší vrstvy. Nejnižší vrstva opět komunikuje přímo s hardwarem, nejvyšší vrstvou je uživatelské rozhraní.

Struktura nebývá čistě hierarchická tím způsobem, že některé z vrstev mohou komunikovat nejen s vrstvami sousedícími ale i s těmi, které jsou o několik úrovní níže. Při virtualizaci je také možné, že nižší vrstvy volají vyšší. Modelová struktura kernelu hierarchického systému je znázorněna na obr. 6. Typickým a zároveň nejstarším příkladem hierarchického operačního systému je Unix. [1, 2, 3]

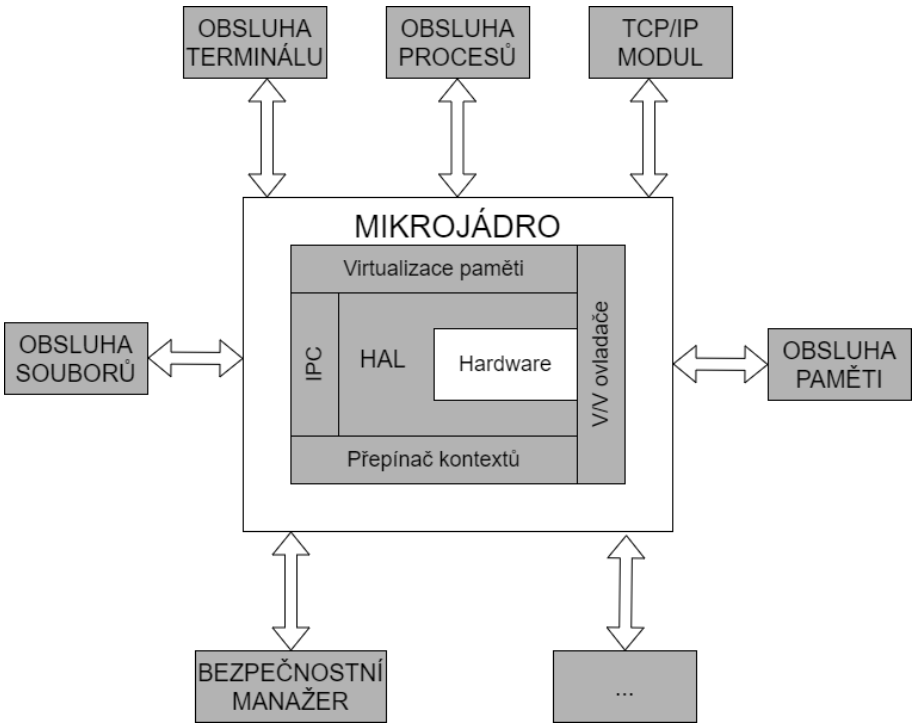


Obr. 6: Kernel hierarchického operačního systému, podle [2]

Typický příklad operačního systému rozděleného do šesti vrstev může vypadat následovně: první (nejnižší) vrstva se stará o přidělování procesoru jednotlivým procesům, přepínání kontextu procesů mezi uživatelským módem a módem kernelu a o multiprogramování. Druhá vrstva přiřazuje procesům prostor v operační paměti a na disku. Třetí úroveň poskytuje komunikaci mezi uživatelem a procesem, čtvrtá řídí vstupní a výstupní jednotky. Na páté úrovni se vyskytují uživatelské programy a šestou vrstvu tvoří shell. [1]

2.3.3 Operační systémy typu klient-server

Struktura typu klient-server (obr. 7) spočívá v oddělení služeb jádra do zvláštních systémových procedur - serverů. Takových serverů může být teoreticky neomezené množství a pro stejnou službu může existovat i více serverů. Jako klient zůstává tzv. mikrojádro, které obsahuje pouze nezbytné procedury, jakými jsou přepínání kontextu procesů, přidělování přístupu k procesoru a virtualizace paměti, hardwarová abstrakční vrstva (HAL), meziprocesorová komunikace (IPC) a ovladače vstupů/-výstupů. [1, 2]



Obr. 7: Struktura operačního systému typu klient-server, podle [1] a [2]

Příslušný server dostává požadavek na službu, tu vykoná a zpátky jako odpověď vrací data. Většina služeb běží v uživatelském módu, v módu kernelu je hlavně mikrojádro. Pokud například uživatelský program požaduje vykonat nějakou akci se souborem, tak zavolá službu mikrojádra. Mikrojádro pak pomocí meziprocesové komunikace zavolá obsluhu souborů, která vykoná daný úkol. Výhodou takovéto architektury je jednoduchá možnost vývoje jednotlivých částí operačního systému nezávisle na ostatních. Nechtěným vedlejším účinkem je ale zhoršení výkonu, operační systémy s architekturou čistě tohoto typu bývají oproti ostatním znatelně pomalejší. [2]

2.4 Operační systémy reálného času

Operační systémy reálného času (anglicky real-time operating systems – RTOS) hrají velmi důležitou roli v oblasti automatizace. Princip RTOS spočívá v tom, že jejich chování nezávisí jenom na výsledku provedených výpočtů, ale i na čase ve kterých byly tyto výpočty dokončeny a také čase, kdy počítač provádí akční zásah. Je přitom kritické dodržení časového omezení (deadlinu) daného řízeným systémem, při nedodržení tohoto požadavku může dojít k nesprávné funkci řízeného procesu nebo i k nebezpečným událostem. Mnohem důležitějším parametrem, než průměrná doba vykonání reakce je tedy doba reakce v nejhorším možném případě. [1, 6]

Systémy vyžadující spolehlivé řízení v reálném čase jsou například: řízení dopravy, chemických procesů, jaderných elektráren, armádních systémů, letadel, průmyslových procesů vyžadujících přesné chování apod. Podle důležitosti dodržení časových omezení se systémy reálného času (RT) dělí na tři kategorie. První jsou soft (měkké) RT systémy, kdy může být opožděná odezva tolerována a způsobí maximálně nárůst nákladů na provoz systému. Druhou kategorií jsou firm RT systémy, jejichž užitečnost s přibývajícím časem strmě klesá a jejich vlákna mají definovaný interval, ve kterým by se měla stihnout vykonat. Zpoždění oproti tomuto intervalu může mít za následek ztrátu kvality anebo poruchu. Poslední a nejpřísnější kategorií jsou hard (kritické) RT systémy, u kterých nelze opožděnou odezvu akceptovat, jelikož by pozdě příchozí data mohla být neplatná až nebezpečná a mohly by tak vést k vážnému selhání systému. Příkladem může být brzdicí systém u aut. [1, 6]

Další podmínkou RTOS je předvídatelná odezva na externí události. Je nutné, aby byla tato podmínka dodržena i když nastane najednou více událostí, které mohou vyžadovat stejné služby a systémové zdroje. To dále znamená nutnost programové podpory souběžného (pseudoparalelního) chodu několika úloh, což je na jednoprocesorovém počítači umožněno jeho kernelem, který přepíná mezi jednotlivými procesy a přiřazuje jim procesor (procesy a jejich plánování bude více probráno v následující podkapitole). Další možností je rozřazení programu na jednotlivá vlákna (threads), která mohou být provozována souběžně a je k nim přistupováno obdobně jako k procesům. Vláknům pak mohou být přiděleny priority, které vyjadřují důležitost vykonání dané úlohy řízené tímto procesem. Vlákna s vyšší prioritou tak dostanou přednost v případě, kdy systém musí reagovat na více událostí najednou. Dalším pozitivem při využití vláken je, že při i selhání jednoho vlákna může být celý systém dále schopen fungovat. Poslední dobou lze sledovat trend ve výzkumné i komerční sféře, co se týče více-procesorových systémů, které jsou schopné vykonávat i skutečný paralelní chod více úloh. [1, 6]

2.5 Procesy

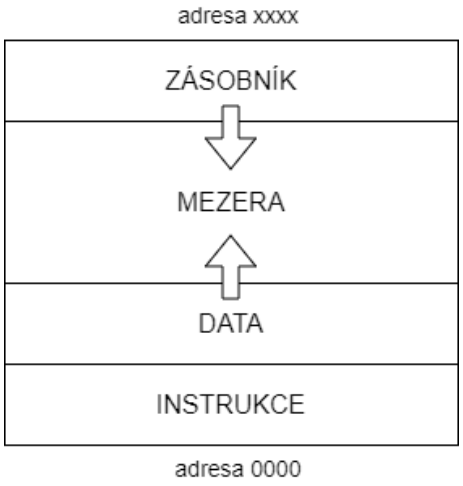
Pojem proces byl již popsán a používán výše, v kontextu této práce je ale podstatné více popsat problematiku procesů, komunikace mezi procesy a hlavně jejich plánování, což je jedním z nejpodstatnějších rozdílů mezi operačními systémy reálného času a systémy obecného využití.

Organizace procesu v paměti

Jak bylo uvedeno na začátku kapitoly, každý proces má svůj vlastní přiřazený paměťový region (obr. 8). Základem každého procesu je textový segment, v němž jsou uloženy instrukce/kód programu. [1, 7]

Další segment obsahuje data procesu, která mohou být inicializována při spuštění procesu a také přidělována dynamicky při běhu. Jsou zde obsaženy globální a statické proměnné. Tento segment nemá pevně danou velikost, jelikož je často nutné ji během života procesu měnit dle potřeby, čehož je dosaženo pomocí haldy. [1, 4, 7]

Proces má v paměti dále zásobník, do kterého se také ukládají data. Je velmi důležitý při volání funkcí. Při volání funkce je této funkci přiřazen v zásobníku rámec, do kterého se při nejmenším ukládá návratová adresa, vstupní data funkce a místo pro lokální proměnné. Rozdělení datového segmentu a paměťových rámců v zásobníku je mimo jiné důvod, proč jsou pro ostatní funkce nepřístupné proměnné deklarované uvnitř funkce jiné, zatímco ke globálním proměnným mají přístup všechny funkce. Podle konvencí se paměťový prostor zásobníku zvětšuje směrem dolů (do nižších adres), zatímco datový segment nahoru. [7]



Obr. 8: Organizace paměťového regionu procesu, podle [1] a [7]

2.5.1 Stavy procesu

S výjimkou bootovacích procesů je každý proces spouštěn a vytvářen jiným procesem. Ukončuje se buď sám anebo násilně jiným procesem či operačním systémem. Od svého spuštění až po dokončení prochází proces třemi hlavními stavy. [1, 4]

Prvním je aktivní stav (ready). Procesy v tomto stavu jsou zařazené v tzv. aktivní frontě procesů, tento stav značí připravenost procesu ke zpracování. Fronta bývá nejčastěji typu prioritní FIFO (first in first out). Do aktivního stavu se proces dostává buď po vytvoření anebo po přerušení operačním systémem například kvůli vykonávání jiného procesu. Procesů v aktivním stavu může být najednou mnoho, záleží pouze na parametrech daného systému. Jaký proces z fronty bude zpracován určuje plánovací mechanismus, který bude podrobněji popsán dále. [1, 2]

Další stav je běžící (running), při kterém proces využívá čas procesoru. Každé jádro procesoru provádí vstupní a výstupní operace, nastavuje hodnoty signálů, komunikuje s ostatními procesy (skrz IPC kernelu) apod. [1]

Třetí stav se označuje jako čekající nebo spící (waiting/sleeping). Takový proces nevyužívá procesor, ani není zařazen v aktivní frontě, je pouze uložen v operační paměti. Počet spících procesů je omezen velikostí operační paměti. Při nedostatku místa v operační paměti může být proces uložen na disk, což je ale například u systémů reálného času potlačeno. Do aktivního stavu přivádí čekající proces vždy operační systém, který na popud jiného procesu zkontroluje u spícího procesu synchronizaci a teprve po úspěšné kontrole ho zařadí zpět do aktivní fronty. [1, 2]

Údaje o všech existujících procesech jsou uloženy v registru procesů. Dají se v něm najít informace jako například priorita procesu, čas spuštění, rodičovské vztahy, atd. [1]

2.5.2 Plánování procesů

Výpočetní systém podporující multiprogamování může mít najednou i tisíce aktivních procesů. Část kernelu, která řídí vykonávání těchto procesů (přiděluje jim procesor) se nazývá plánovač (scheduler). U různých typů operačních systémů jsou také různé cíle. U dávkových systémů jde o co nejrychlejší zpracování prací, maximálního využití procesoru a zpracování co nejvíce prací za jednotku času. Interaktivní systémy se snaží splnit očekávání uživatele a rychle reagovat na jeho požadavky. V systémech reálného času je nutné splnit dané deadliny a zajistit předvídatelné chování systému (podrobněji bude vysvětleno v dalších sekcích). Existuje proto mnoho různých plánovacích algoritmů, které budou dále detailněji popsány. [1, 4, 7]

Plánovací algoritmy se dají obecně rozdělit na dvě kategorie: kooperativní a preemptivní. Při kooperativní strategii proces dobrovolně ukončí svůj běžící stav (i dříve, než je tento proces dokončen), aby mohl procesor využívat jiný proces. Může ale nastat situace (většinou kvůli chybě v programu nebo uvíznutí v nekonečné smyčce), kdy proces z běžícího stavu nikdy neodejde, tudíž při čistě kooperativním plánování hrozí, že žádný další proces již nepoběží. I když nemusí nastat tento extrémní případ, běžící proces může trvat dlouho a tudíž ostatní procesy musí čekat, i když by jejich vykonání mohlo být důležitější. Preemptivní strategie tyto problémy řeší tak, že poskytuje procesu určité časové kvantum. Pokud proces stále běží před koncem tohoto kvanta, tak je přerušen (přesunut do aktivní fronty) a plánovač znovu rozhodne, kterému z aktivních procesů přidělí procesor. [7]

FCFS

Mechanismus postupného plánování označovaný zkratkou FCFS (First Come First Served) je nepreemptivní, používaný v dávkových systémech a také ten nejjednodušší. Fronta aktivních procesů je pouze jedna a není prioritní, procesy se tedy

vykonávají v pořadí, ve kterém do fronty přijdou a není nijak omezen čas jejich vykonávání. Výhodou tohoto algoritmu je velmi jednoduchá implementace, ale jak již bylo zmíněno, má i velmi vážné nevýhody při dlouhém zpracovávání nebo v horším případě zaseknutí procesu, není ale ani vhodný, co se týče průměrné doby čekání jednotlivých procesů. [1, 4]

Pro porovnání jednotlivých plánovacích algoritmů bude uveden následující příklad:

Tab. 1: Příklad procesů

Proces	**Doba zpracování [ms]**
A	4
B	20
C	8
D	100

Průměrná doba čekání u FCFS závisí na pořadí jednotlivých procesů (a samozřejmě na době zpracování jednotlivých procesů, kterou ale plánovač nemůže ovlivnit). Při nejhorším možném scénáři přijdou procesy v pořadí od nejnáročnějšího (s nejdelší dobou zpracování), tedy D-B-C-A. Doby čekání jednotlivých procesů budou vypadat následovně: D - 0 ms, B - 100 ms, C - 120 ms, A - 128 ms. Průměrná doba čekání je v tomto případě 87 ms.

Pokud by ale procesy přišly v opačném (a nejlepším) pořadí, doby čekání by byly: A - 0 ms, C - 4 ms, B - 12 ms, D - 32 ms. Nyní je průměrná doba čekání pouze 12 ms. Pokud by všechny procesy měly víceméně stejně velkou dobu zpracování, na pořadí by tolik nezáleželo. Ve skutečnosti jsou ale procesy často různě náročné, tudíž tento algoritmus není příliš efektivní.

SJF

Algoritmus plánování podle délky procesu (Shortest Job First) přímo adresuje problém představený v předchozím příkladu. Pro minimalizaci průměrné doby čekání je optimální strategií zpracovávat procesy od toho nejméně náročného. Při vstupu procesu do fronty se jeho délka porovná s délkou ostatních procesů a zařadí se tak, aby byly procesy ve frontě vždy seřazeny od nejkratšího po nejdelší. V důsledku tedy při jakémkoliv pořadí procesů A-B-C-D bude průměrná doba čekání 12 ms. [1, 4]

Je důležité zmínit, že doby zpracování jednotlivých procesů často nemusí být známy. V tomto případě se provádí odhad, který se dále upravuje na základě předchozí skutečné doby zpracování, jejího odhadu a váhy určující, v jakém poměru má předchozí odhad a skutečná doba zpracování vliv na aktuální odhad. [1]

Preemptivní SJF

Základní forma SJF je nepreemptivní. V důsledku to znamená, že vzhledem k průměrné době čekání zaručuje optimálnost pouze tehdy, pokud jsou všechny procesy v aktivní frontě dostupné najednou. Preemptivní SJF počítá i z případem, kdy je nějaký proces v běžícím stavu a do fronty přijde další proces. Plánovač bere v potaz časy, které procesy potřebují k dokončení a vždy vybírá ten nejkratší. Tudíž pokud má nový proces kratší dobu potřebnou ke zpracování než běžící proces, běžící proces je přerušen a začne se zpracovávat nový proces. [4]

Můžeme opět vzít v potaz příklad procesů uvedený výše. Nechť je v čase 0 v aktivní frontě pouze proces D, ten se tedy začne vykonávat. V další milisekundě přijdou do fronty procesy A a C. Při nepreemptivním plánování bude proces A čekat 99 ms a proces C 103 ms. Je zřejmé, že průměrná doba čekání bude mnohem vyšší, než by musela být (konkrétně 67,33 ms).

Preemptivní SJF v tomto případě vyhodnotí, že proces A má menší dobu potřebnou k dokončení (4 ms) než proces D (99 ms), tudíž začne vykonávat tento proces. Jako další bude zpracován proces C a po jeho dokončení se do běžícího stavu vrátí proces D. Nyní proces A nečekal žádnou dobu, proces C 4 ms a proces D 12 ms. Průměrná doba čekání se zkrátila na 5,33 ms. Pokud by nastala situace, že procesu D zbývá k dokončení pouze 5 ms, a do fronty by vstoupil například proces B, který má v této chvíli delší potřebnou dobu k dokončení, je optimální nejprve dokončit proces D a až poté začít zpracovávat proces B.

Round-Robin

Jedním z nejstarších, nejjednodušších a nejpoužívanějších algoritmů v interaktivních systémech je mechanismus plánování s cyklickou obsluhou (Round-Robin). Procesu je vždy přiděleno pevné časové kvantum. Další proces se začne vykonávat buď po dokončení aktuálního procesu anebo po skončení kvanta, kdy se aktuální proces řadí na konec fronty. [1, 4]

Tento mechanismus je ze své podstaty preemptivní a nepotřebuje při chodu znát doby zpracování jednotlivých procesů. Musí ale řešit důležitou otázku a tou je, jak velké by mělo být časové kvantum. Teoreticky by bylo nejlepší mít časové kvantum co nejmenší, mezi procesy přepínat velmi rychle a tím zajistit jakousi férovost mezi všemi procesy. Ve skutečnosti ale přepínání mezi procesy také procesoru zabírá nějaký čas - data je nutné mazat z a nahrávat do paměti, načítat registry atd. Pokud by přepnutí trvalo 1 ms a délka časového kvanta by byla 4 ms, tak by čas procesoru byl využit jen z 80 %, při zmenšování časového kvanta se tento podíl bude pouze zhoršovat. Pokud by ale bylo časové kvantum příliš velké (větší než je doba nejdelších zpracovávaných procesů), tak by tento mechanismus degradoval na FCFS. [1, 4]

Pro optimální funkci algoritmu Round-Robin je tedy vhodné, aby se mezi procesy přepínalo co nejméně (tedy aby často končily samy buď ukončením zpracování nebo čekáním na jiný proces/operaci), ale zároveň aby výpočetně náročné procesy nedostávaly příliš velké kvantum času. Obecně je tento algoritmus nejefektivnější, když je velikost časového kvanta podobná průměrné délce zpracování procesu. [1]

Prioritní plánování

Round-Robin předpokládá, že jsou všechny procesy stejně důležité. To ale v praxi často nemusí být pravda. [1, 4] Zvlášť to platí u systémů reálného času, kdy některé procesy jsou mnohem podstatnější než jiné - například bezpečnostní zastavení průmyslové linky musí mít v PLC vyšší prioritu než procesy starající se o vizualizaci na panelu operátora. Různé priority procesů se ale dají najít i u systémů obecného využití. Například když si uživatel přehrává hudbu, tak nechce, aby se mu zasekávala kvůli nepodstatným aktualizacím běžícím na pozadí, videohra by se neměla zastavit při přijetí mailu apod.

Procesům jsou tedy přiřazovány priority označené přirozeným číslem. Algoritmus SJF se dá chápat jako speciální prioritní algoritmus, kdy nejvyšší prioritu mají ty nejkratší procesy a naopak. Obecně jsou procesy s vyšší prioritou zpracovávány před procesy s prioritami nižšími. Priority se dělí na interní a externí. Interní nastavuje plánovač na základě údajů, jakými jsou například paměťové nároky, časový limit apod. Externí priority nastavuje uživatel podle důležitosti jednotlivých procesů nebo třeba za účelem dosažení optimální synchronizace. [1]

Dále se priority dají nastavovat jako statické nebo dynamické. U statických priorit je uživatelem většinou priorita nastavena pouze před spuštěním úlohy a její hodnota se dále nemění, což je žádoucí právě u systémů reálného času. [1]

Dynamické priority se v průběhu úlohy mohou měnit, například procesům čekajícím na zpracování dlouhou dobu může být priorita uměle zvyšována, aby se dostaly na řadu, a naopak procesům, které již spotřebovaly velké množství procesorového času, se může priorita zmenšovat. Zajímavým příkladem jsou procesy trávící většinu času čekáním na vstupy/výstupy. Pokud takový proces požaduje přístup k procesoru, pak je často vhodné, aby ho dostal co nejdříve, protože brzy opět může při čekání na další vstupní/výstupní operaci přenechat procesor jinému procesu. [1, 4]

Více procesů může mít stejnou prioritu. Je časté tyto procesy shlukovat do skupin a v rámci nich plánovat pomocí algoritmu Round-Robin (přičemž jsou samozřejmě skupiny s nějvyšší prioritou obsluhovány jako první). [4]

MLQ

V plánování pomocí více front (Multilevel Queue Scheduling) má každá z front vlastní prioritu. Nové procesy jsou vždy přiřazeny do do fronty s nejvyšší prioritou,

uvnitř které se procesy zpracovávají pomocí FCFS. Procesu je opět přiděleno časové kvantum k dokončení. Pokud se proces v tomto kvantu nestihne vykonat, spadá do fronty s nižší prioritou. Například u zmiňovaného systému CTSS mu v této frontě byl přidělen dvojnásobek kvanta, v nižší frontě čtyřnásobek, v třetí frontě osminásobek atd. Tím byla dosažena rychlá odezva u krátkých procesů, což bylo v tomto systému, používaném mnoha uživateli najednou, velmi důležité. Také se tím minimalizoval počet přepnutí, u toho bylo potřeba ukládat rozpracovaný proces z operační paměti na disk, takže se tím ušetřilo velké množství času. [1, 4]

MLQ bylo použito v mnoha dalších systémech, které měly různý mechanismus přidělování procesů do konkrétních prioritních front. Vzhledem k dynamickému přiřazování priorit a nedefinovanému počtu procesů v jednotlivých frontách ale tento algoritmus není vhodný pro systémy reálného času. [1, 4]

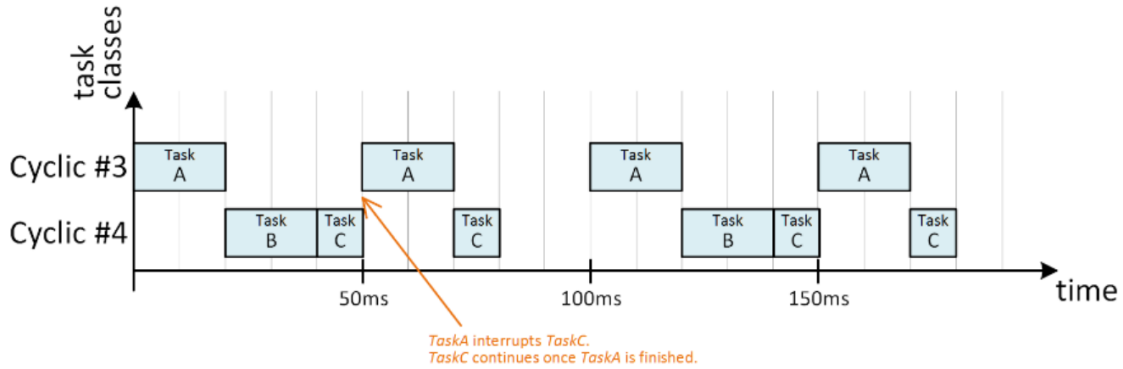
Plánování v reálném čase

U systémů reálného času je způsob plánování procesů jednou ze zásadních charakteristik. Procesy bývají vcelku krátké (s dobou zpracování v řádech maximálně stovek milisekund), ale je potřeba, aby byly zpracovány do určitého času. Procesy mohou také být buď periodické - tedy vykonávané pravidelně po uplynutí daných intervalů, anebo acyklické - nastávající nepředvídatelně. Velkou roli zde hrají priority. Může nastat i situace, kdy se všechny procesy do daných deadlinů nemohou stihnout zpracovat, v takovém případě se musí vykonat co nejvíce procesů s co největší prioritou. [1, 4]

Příkladem plánování může být RTOS s názvem OS-9. Jeho plánovač nabízí čtyři algoritmy, jež se dají používat buď samostatně a nebo v jejich kombinacích. Prvním je automatické plánování, založené na algoritmu Round-Robin upraveném tak, že procesy s vyšší prioritou dostávají častěji přiřazené časové kvantum. Druhá je metoda minimální priority procesu, kde plánovač určuje práh. Procesy mající menší prioritu než daný práh nedostávají přidělené žádné časové kvantum do té doby, dokud se práh nezmenší anebo se dynamicky nezvětší jejich priorita. Princip dalšího algoritmu funguje obdobně pomocí prahu, ale pokud není v aktivní frontě zrovna žádný proces mající prioritu větší nebo rovno danému prahu, metodou Round-Robin se plánují procesy pod prahem. Nakonec je uživateli umožněno naprogramovat si svůj vlastní algoritmus a ovládat přerušení. [1]

V praktické části bude používán RTOS s názvem Automation Runtime od společnosti B&R. Cyklické aplikace (procesy) jsou v něm označovány jako tasky. Každý task je v konfiguraci procesoru přidělen do třídy tasků. Každá třída má vlastní prioritu, přičemž nižší číslo znamená vyšší priorita. Krom priority má i přiřazenou časovou periodu, ve které se bude vykonávat. Tasky ve stejné třídě se vykonávají ve stejném pořadí, v jakém jsou seřazeny. Pokud zrovna běží task s nižší prioritou a přijde na řadu task s prioritou vyšší, tak je ten méně důležitý přerušen. Příklad

je vidět na obr. 9, kdy třída Cyclic #3 je prováděna každých 50 ms a má vyšší prioritu než třída Cyclic #4 s periodou 100 ms. Každý program (krom těch s nejvyšší prioritou) tedy může být kdykoliv během svého vykonávání přerušen, s čímž je nutno během vývoje počítat, pokud například mezi sebou jednotlivé programy komunikují pomocí globálních proměnných. [8]



Obr. 9: Plánování procesu v Automation Runtime [8]

3 KOOPERACE RTOS A GPOS

Cílem této kapitoly je probrání problematiky kooperace RTOS a GPOS. Udává obecné důvody pro toto spojení, aplikační příklady a v dalších sekcích pojednává o konkrétních komerčně dostupných produktech, které umožňují kooperaci mezi RTOS a GPOS. Jsou zde vyzdviženy jejich vlastnosti, způsob funkce a vyjmenované případné certifikáty. Zaměření této práce je hlavně na běh obou operačních systémů na jednom zařízení.

RTOS jsou deterministické a určené k řízení kritických systémů, jakými jsou roboty, frézy a podobné průmyslové stroje, dále k řízení zdravotnických zařízení, systémů v+automobilech a letadlech, vojenských systémů apod. GPOS dávají ale mnohem větší svobodu ve volbě programovacích jazyků (převážně těch vyšší úrovně), frameworků, knihoven, apod. Mohou tedy například poskytovat data pro RTOS, využít algoritmů, které by byly v RTOS obtížné na implementaci nebo poskytnout uživateli služby, které by pouze s RTOS nebyly možné. Kooperace může mít obecně dvě formy: v některých případech GPOS a RTOS obstarávají různé aplikace, které spolu přímo nekomunikují, výhoda v běhu obou systémů na jednom hardwaru je pak hlavně v konsolidaci hardwaru a tím pádem ušetření nákladů. V jiných případech mezi sebou oba systémy přímo komunikují.

3.1 Hypervisor

Velmi důležitým pojmem pro dotčené téma je hypervisor. Je to software, který umožňuje provozovat více operačních systémů na stejném počítači a poskytuje jim abstrakci hardwaru. Operační systémy běžící na hypervisoru by neměly potřebovat žádné modifikace a také obecně nemají povědomí o faktu, že neběží přímo na hardwaru. Hypervisor musí také zajistit oddělení obou operačních systémů (kromě požadované výměny dat) a měl by pouze minimálně ovlivňovat celkový výkon celého systému. [9]

Hypervisory se dělí na dva typy. Hypervisor prvního typu běží přímo na hardwaru a všechny další operační systémy jsou ve vyšších vrstvách. Hypervisor druhého typu běží na operačním systému, který ho bootuje.

3.2 Možnosti využití

Typickým příkladem využití kooperace RTOS a GPOS se zabývá článek [10]. Jedná se o sdílení zvukové karty a videokarty připojených k palubní obrazovce v automobilu. Při běžném provozu se využívá služeb GPOS jakými jsou například multimédia (přehrávání hudby, rádia), přístup k internetu nebo navigace pomocí GPS. Při cou-

vání je potřeba řízení systémem reálného času, aby byla zajištěna správná a včasná funkce přenosu obrazu z parkovací kamery a také zvukové výstrahy, když se parkovací senzory přibližují k překážkám. Článek zmiňuje dvě možná řešení. Prvním z nich je, že systémy jsou kompletně oddělené, mají svůj vlastní hardware a k palubní obrazovce se připojují za pomoci mechanického přepínače aktivovaného zařazením zpátečky. Druhé řešení, které článek navrhuje, má pouze jednu zvukovou kartu i videokartu a přístup k nim se přiděluje operačním systémům pomocí hypervisoru.

V článku [11] je popsán příklad, ve kterým RTOS sbírá data z kamer a ovládá robotickou ruku. GPOS je pak využit pro propojení s ethernetem a poskytnutí vizualizace.

V průmyslu je možné využití algoritmů umělé inteligence, často ve spojení se strojovým viděním nebo v prediktivní údržbě. Často používanými platformami pro tvorbu umělé inteligence jsou například TensorFlow nebo PyTorch využívajícími jazyka Python, který je možné použít v GPOS. V kooperaci GPOS a RTOS je takto možné neuronové sítě učit na základě sbíraných dat a poté pomocí nich data vyhodnocovat. Konkrétním příkladem může být projekt firmy B&R, kde neuronová síť na základě kamerových dat řídila regulátor čističky odpadních vod. V poslední době má umělá inteligence také vzrůstající využití při řízení autonomních vozidel.

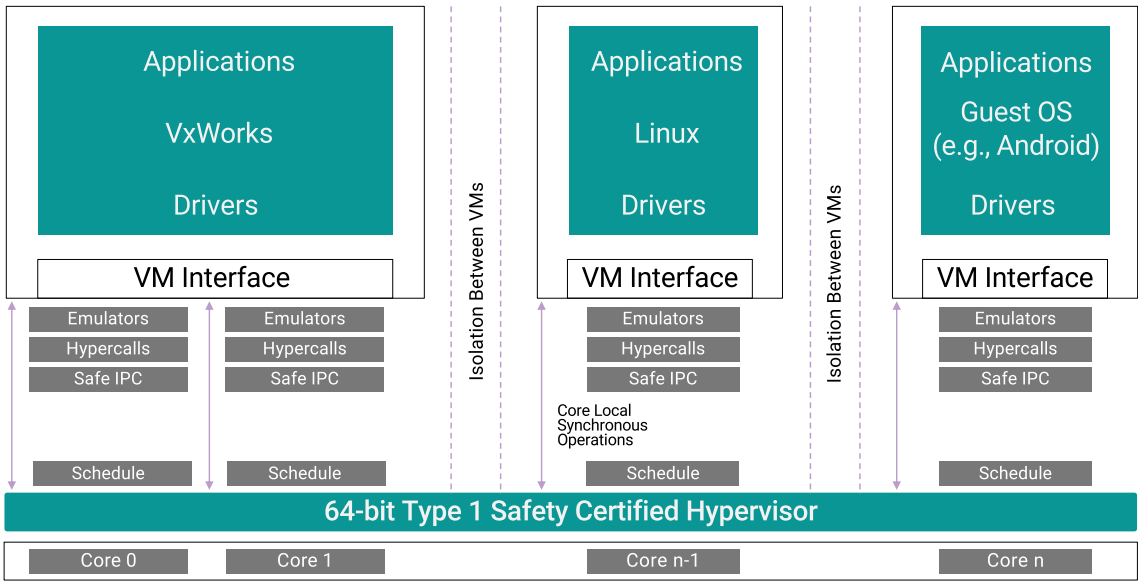
Dalším z využití jsou algoritmy plánující cestu pro mobilní roboty. Kooperace je také výhodná při využití databází dostupných na GPOS, které poskytují nebo sbírají data z RTOS, takový příklad bude řešen v praktické části této práce. Za zmínku stojí také využití konceptů Průmyslu 4.0 jako internetu věcí (IoT) či digitálních dvojčat.

Další příklady využití jsou uvedeny v některých z následujících sekcí u výrobců, kteří u svých produktů udávali konkrétní aplikační využití.

3.3 Wind River – Helix

Platforma pro virtualizaci od firmy Wind River s názvem Helix podporuje běh několika operačních systémů s různými stupni kritičnosti. Splňuje bezpečnostní normy, jakými jsou DO-178C DAL A, IEC 61508 SIL 3 a ISO 26262 ASIL-D, díky čemuž je vhodný pro použití v různých aplikacích vyžadujících bezpečnostní zabezpečení, například v letectví, automobilovém odvětví či v průmyslové výrobě. Je navržen i pro použití v systémech, kde se kombinují aplikace s bezpečnostními certifikáty s aplikacemi, které jsou necertifikované. [12, 13]

Tento produkt je rovněž možné využít pro decentralizovaný vývoj jednotlivých operačních systémů různými vývojářskými skupinami a firmami, které si mohou v daném poli konkurovat. Jsou zde dodrženy principy vývojových rolí definovaných v leteckém standartu DO-297. Poskytovatelé softwaru mohou tedy nezávisle



Obr. 10: Architektura hypervisoru Helix [12]

na ostatních vyvíjet, testovat a dodávat svoje aplikace, což je podpořeno samostatným sestavovacím (build) procesem pro jednotlivé dodavatele. Helix je proto vhodný i pro projekty velkého rozsahu. Zajímavým prostředkem je také podpora běhu legacy aplikací zároveň s aktuálními aplikacemi. Je tedy možné na jednu platformu spojit několik různě starých systémů, které byly předtím používány na samostatných zařízeních. [13]

Helix je hypervisorem typu 1 a zaručuje téměř nativní výkon i při velkém množství jader procesoru a virtuálních strojů. Výrobce udává, že na počet jader a hostujících operačních systémů není stanoven žádný limit. Pro deterministické hostující operační systémy zaručuje odezvu ve reálném čase. Procesy běžící na různých jádrech jsou na sobě plně nezávislé. [13]

Je možné použití tickless kernelu, ve kterém přerušení časovače nenastávají v pravidelných intervalech, ale pouze tehdy, když jsou potřeba. V důsledku by tedy při použití pouze jednoho virtuálního stroje nedocházelo k žádným přerušení. Další možností jsou rozvržená rozdělení v čase, kdy je na jednom jádru několik časových úseků a tyto úseky jsou přidělovány jednotlivým virtuálním strojům, které tak mají garantováno určité kvantum procesorového času. Pro celistvý systém se dá definovat několik takovýchto časových rozvrhů, mezi nimiž se může za chodu dynamicky přepínat. [12]

Další ze schopností platformy Helix je možnost využití přímých přerušení a přímého přístupu k zařízení. Při takovém přerušení není hypervisor využit a data putují z fyzického zařízení rovnou do hostujícího operačního systému, což vylepšuje rychlost odezvy. [12]

Pomocí RTOS VxWorks vyvinutého přímo firmou Wind River je zajištěn debugger, který umožňuje synchronizovaný pohled na všechny virtuální stroje, čímž je možné sledovat komplexní aktivity celého systému. Dalším nástrojem poskytovaným společně s produktem Helix je například bootovací profiler měřící čas využitý hypervisorem a hostujícími operačními systémy při inicializaci pro umožnění optimalizace bootování. [12, 13]

Podpora hostovaných operačních systémů je velmi rozsáhlá. Wind River dodává s platformou Helix tři své systémy – již zmiňovaný RTOS VxWorks, jeho verzi s bezpečnostní certifikací a jako GPOS Wind River Linux. Jako další možné hostované systémy jsou uváděny systémy s přímým přístupem k hardwaru a systémy třetích stran. Podporovanými procesory jsou řady 64bitové ARMv8 a Intel x86-64, hosté mohou být 32 nebo 64bitoví. [12]

3.4 Green Hills Software

Další firmou poskytující řešení pro kooperaci RTOS a GPOS je Green Hills Software. Prvním takovým produktem je INTEGRITY Multivisor vytvořený pro procesory typu ARM a x86. Druhý nástroj této společnosti je μ-visor určený pro mikrokontroléry.

3.4.1 INTEGRITY Multivisor

INTEGRITY Multivisor je dalším hypervisorem typu 1. Použitý systém reálného času je INTEGRITY RTOS, který je i základem architektury tohoto hypervisoru. Jako podporované GPOS Green Hills Software uvádí Android, Windows a Linux a jeho odvození, jakými jsou například Automotive Grade Linux nebo Genivi, které jsou určené pro automobilový průmysl. Multivisor zařizuje efektivní plánování procesů hostujících operačních systémů na jednom nebo více jádrech, komunikaci mezi jednotlivými systémy a uděluje přístup k periferiím jako například Ethernetu nebo grafické kartě. [14]

Multivisor splňuje množství certifikací na bezpečnost a spolehlivost, jakými jsou ISO 26262 ASIL D pro automobilovou elektroniku, NSL – certifikované zabezpečené mobilní telefony, FAA DO-178B Level A – funkce kritické pro život cestujících v avionice a pro armádní letadla, FDA Class III life – zdravotnická zařízení, EN 50128 SWSIL 4 pro železniční řídicí systémy a v průmyslu rozšířený standard IEC-61508 SIL3. [14]

Green Hills Software vyjmenovává i řadu skutečných příkladů využití Multivisoru. Jedním z nich je automobilní kokpit, kde se řídicí systém stará o multimédia na dotykovém panelu a zároveň o kritické bezpečnostní aplikace, jakými jsou varovná světla, parkovací kamera anebo monitorování hostujícího operačního systému. Na

mobilních telefonech je v dnešní době požadováno mít aplikace jako virtuální platební karty, klíče a identifikační nástroje, které vyžadují velkou míru zabezpečení a tím pádem i chod RTOS vedle mobilního GPOS. Dalším uvedeným příkladem je zobrazení stavu zbraňového systému a nastavování munice (bezpečnostně kritické aplikace vyžadující RTOS) na tabletu s grafickým uživatelským rozhraním v Linuxu. [14]

K INEGRITY Multivisoru výrobce dodává i tzv. MULTI debugger, který umožňuje nahlédnout do chodu všech použitých operačních systémů, jejich kernelů a knihoven, i přímo do řízení chodu virtuálních strojů. [14]

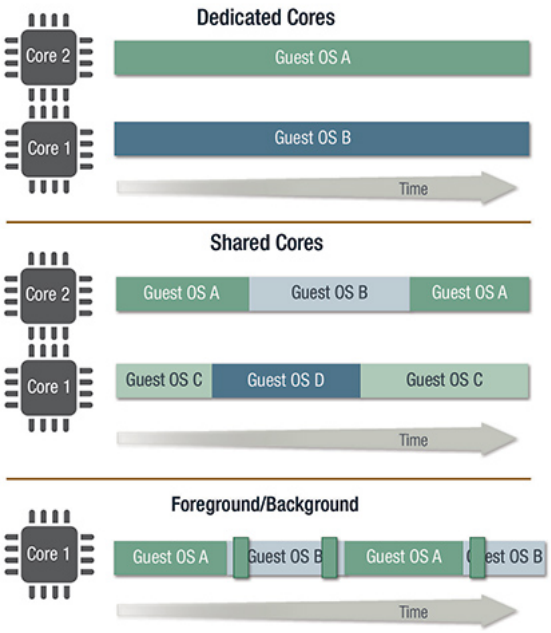
3.4.2 μ-visor

Jak již bylo zmíněno, μ-visor představuje řešení virtualizace pro mikrokontroléry s omezeným výpočetním výkonem (v porovnání například s procesory z řad ARMv8 a x86), které opět podporuje chod několika různých typů operačních systémů. [15]

Produkt podporuje provozovat operační systémy různých výrobců, jako příklady těchto systémů jsou uvedeny AUTOSAR Classic, FreeRTOS anebo μ-VELosITY. Jedny z mikrokontrolerů, na nichž lze μ-visor provozovat jsou Renesas RH850 U2A a Arm Cortex-R52. Kromě nezávislosti na konkrétních výrobcích operačních systémů a mikrokontrolérů je μ-visor flexibilní i v možnost použití standardních komunikačních kanálů a sdílení periferních zařízení procesoru. [15]

Oproti výše probranému INTEGRITY Multivosoru má μ-visor o něco méně bezpečnostních certifikací, patří mezi ně zmiňované ISO26262 ASIL D, IEC61508 SIL 3 a také standardu ISO 21434 zabývající se kybernetickou bezpečnostní v automobilovém průmyslu. Na podporovaných procesorech μ-visor poskytuje hardwarově oddělený chod operačních systémů, ochranu kritických registrů a jejich alokaci mezi virtuálními stroji, i jejich bezpečné sdílení periférií díky monitorování sběrnic. Dále je nabízeno řízení selhání, které je konfigurovatelné uživatelem a je užitečné pro certifikaci ASIL D. Tato funkce může být využita pro obnovu, analýzu a řízení virtuálních strojů při výskytu chyb. [15]

Aby bylo možné efektivně využít více jader, μ-visor poskytuje 3 různé přístupy k práci na vícejádrových mikrokontrolerech (obr. 11). První možností jsou dedikovaná jádra, kdy na jádru běží pouze jeden virtuální stroj. Sdílená jádra umožňují virtuálním strojům používat jakákoli jádra. Poslední možnost s názvem popředí/-pozadí přiřadí virtuálnímu stroji prioritu, a tudíž se jeho služby vykonají vždy před ostatními stroji s nižší prioritou. [15]



Obr. 11: Možností využití více jader μ-visorem [15]

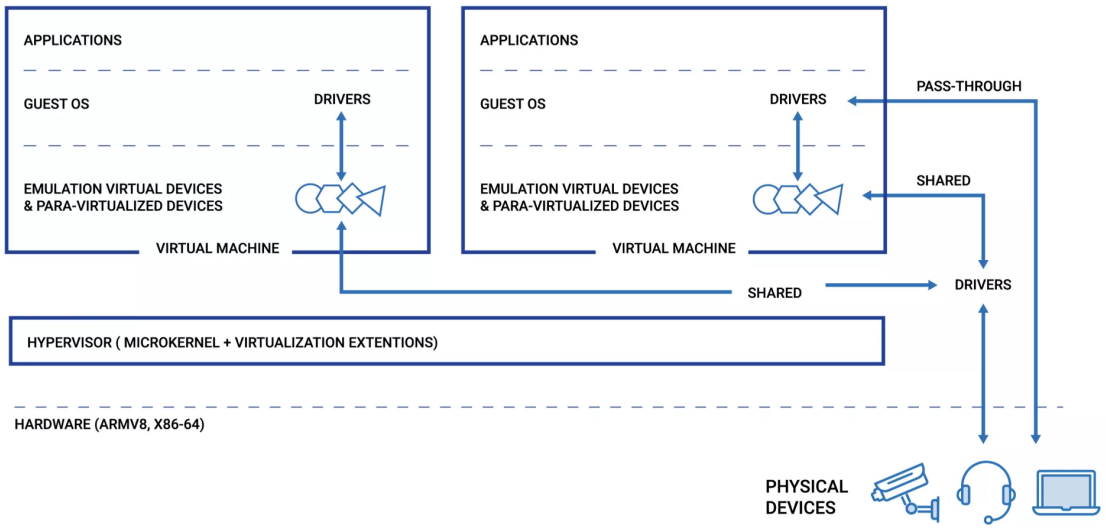
3.5 BlackBerry - QNX Hypervisor

Dalšími produkty umožňující kooperaci GPOS a RTOS, za jejichž vývojem stojí společnost BlackBerry, jsou QNX Hypervisor a QNX Hypervisor for Safety, což je verze pro aplikace vyžadující bezpečnostní certifikace, konkrétně IEC 61508 SIL 3 a ISO 26262 ASIL D. Tyto hypervisory jsou primárně inzerovány pro použití na takzvané systémy na čipu (System on a Chip - SoC), ale podporují všechny 64-bitové procesory z řad ARMv8 a x86. [16, 17]

Jako RTOS je použit buď QNX Neutrino, anebo v případě nutnosti bezpečnostní certifikace QNX OS for Safety, na nichž jsou oba hypervisory i založeny a rozšiřují jejich mikrokernely možnostmi virtualizace. Co se týče GPOS, je na výběr mezi Linux Ubuntu 18.04 nebo 16.04 a Androidem minimální verze 7.0. [16]

Při konfiguraci virtuálních strojů se jednotlivým operačním systémům přiřadí alokovaná paměť a příslušná zařízení, která mohou být z pohledu operačního systému virtuální či s přímým přístupem. QNX Hypervisory podporují i para-virtualizaci. Virtuální stroje jsou z pohledu QNX Hypervisoru pojata jako vlákna, mohou se tedy dle potřeb dynamicky přesouvat mezi jednotlivými jádry procesoru, čímž je zaručena krátká doba vykonávání procesů. [16]

Komunikace mezi jednotlivými operačními systémy může být buď síťová, nebo pomocí sdílené paměti. Pro vývoj je možné použít stejné API a frameworky jako pro RTOS QNX Neutrino. [16]



Obr. 12: Architektura QNX Hypervisor [16]

3.6 Acontis Technologies

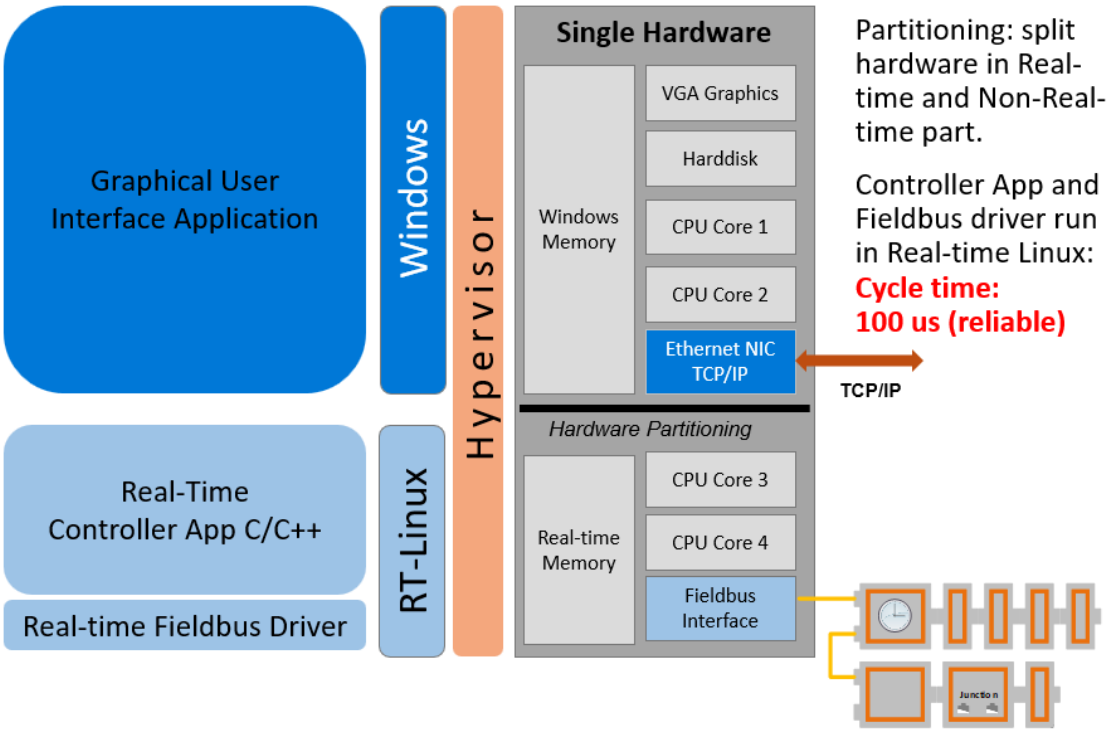
Firma Acontis Technologies nabízí několik hypervisorů typu 1 i 2, z nichž všechny podporují běh a komunikaci RTOS a GPOS. [18]

3.6.1 LxWin/VxWin

LxWin a VxWin jsou hypervisory druhého typu, oba jako GPOS používají Windows. RTOS fungující s LxWin je Real Time Linux, zatímco pro VxWin je to VxWorks firmy Windriver. [18]

Windows funguje jako hostitelský operační systém, bootuje se jako první a poté načte Real-time Hypervisor. Hypervisor poté bootuje a ovládá RTOS. Aby mohl RTOS pracovat v reálném čase, musí být ale plně nezávislý na GPOS. Toho je dosaženo hardware oddělením (partitioning) na několik částí, přičemž RTOS bude fungovat na své vlastní části. Toto rozdělení se týká procesoru, paměti i dalších zařízení, které má RTOS možnost používat. Pro přiřazování hardwaru jednotlivým OS a pro další konfiguraci poskytuje Acontis nástroj s názvem System Manager. [19, 20]

Hardwarová podpora virtualizace v procesoru není přímo vyžadována, ale Acontis na těchto procesorech zaručuje vyšší robustnost, spolehlivost a také možnost sdílení jader mezi operačními systémy. Hypervisory lze spustit téměř na jakémkoli zařízení, které podporuje Windows 7 a vyšší, jedná se tedy o vícejádrové procesory Intel a AMD, kromě jednojádrových procesorů bez podpory virtualizace. [19, 20]



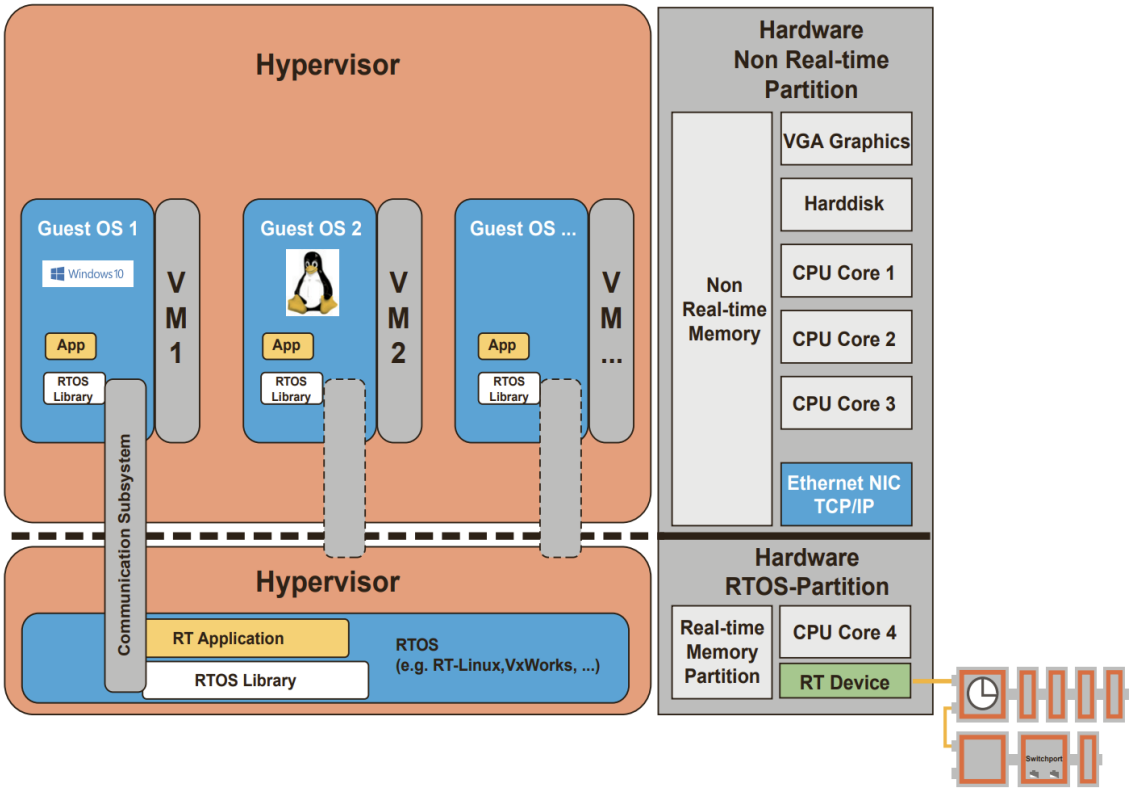
Obr. 13: Architektura hypervisoru LxWin [19]

3.6.2 RTOSVisor

RTOSVisor je hypervisorem typu 1. Umožňuje běh několika RTOS (Real-Time Linux, VxWorks, On Time RTOS-32 a další) i GPOS (Linux, Windows) najednou na stejném hardwaru, přičemž každý operační systém je plně nezávislý a oddělený, což zahrnuje možnost restartu, anebo vypnutí operačního systému bez ovlivňování ostatních. [18, 21]

Pro konfiguraci hypervisoru se používá webová aplikace, která umožňuje i vzdálený přístup k zařízení, přidávání a správu hostujících operačních systémů, alokování hardwaru a vypínání/zapínání jednotlivých operačních systémů. [21]

RTOS používají VMF (Virtual Machine Framework), který umožňuje přímý a rychlý přístup k jádrům procesoru nebo sběrnicím CAN a Ethernet bez nutnosti zásahu hypervisoru. Na straně GPOS se o přístup k hardwaru stará KVM (Kernel-based Virtual Machine), což je jeden z nejpoužívanějších standardů v bezpečnostně kritických systémech, a je využíván například firmami jako je: Google, Amazon anebo Oracle. KVM umožňuje zvýšit výkonnost tím, že na podporovaném hardwaru může být jeho část (USB, grafická karta nebo sběrnice Ethernet/PCIe) přiřazena přímo k jednomu konkrétnímu hostujícímu operačnímu systému bez zásahu hypervisoru, což ale znemožní využívat tento hardware ostatními operačními systémy. Druhou možností je takzvaná para-virtualizace, kdy operační systém nemá hardware pouze emulovaný hypervisorem, ale o hypervisoru „ví“ a může k hard-



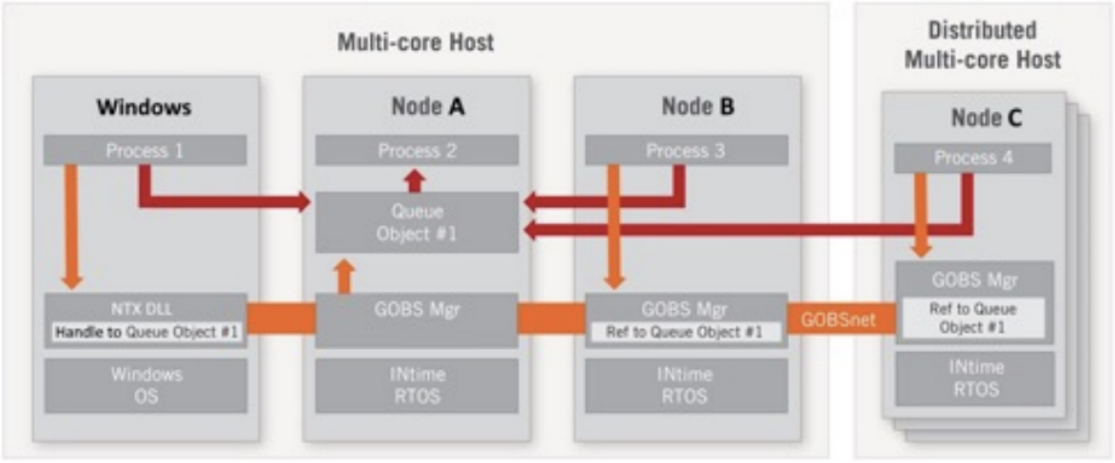
Obr. 14: Architektura RTOSVisoru [21]

waru přistupovat přímo, pokud je to hypervisorem umožněno. Nevýhodou tohoto přístupu je, že daný operační systém je nutno upravit pro specifický hardware, se kterým bude pracovat. [21, 22]

3.7 TenAsys – INtime

Řešení s odlišným přístupem, než byl v této kapitole doposud popisován, poskytuje společnost TenAsys. U svého RTOS s názvem INtime nabízí možnost fungování a komunikace s GPOS Windows buď na rozdílných zařízeních pomocí sítě, anebo na stejném zařízení, a to bez nutnosti použití hypervisoru, ale jako rozšíření Windowsu. INtime je možné provozovat i pod hypervisorem, ten ale musí být od jiné firmy, protože TenAsys žádný hypervisor nenabízí. Tento operační systém může fungovat i samostatně. Vzhledem k zaměření této práce budou dále popsány pouze možnosti, ve kterých je INtime verze 7.1 použit ve spolupráci s Windows. [23]

INtime využívá takzvaného explicitního oddělení hardwaru, kdy každý oddíl pracuje zcela samostatně. Zatímco u hypervisoru přistupují operační systémy k virtualizovaným zařízením, u INtime je přístup například ke vstupům a výstupům přímý a vstupy/výstupy používané INtimem nejsou přístupné Windowsu. Rozdělení operačních systémů na procesoru vypadá tak, že každá instance INtimu běží na



Obr. 15: Distribuovaný systém INtime [24]

jednom nebo více svých vláknech a jedna instance Windows má k dispozici všechna zbylá vlákna, přičemž INtime i Windows musí mít přiřazené alespoň jedno vlákno. V případě víceprocesorových počítačů běží každá instance INtimu pouze na jednom procesoru, což je označováno za asymetrický multiprocesing (AMP). Naopak Windows používá symetrický multiprocesing (SMP), ve kterém není použito pevné rozdělení procesorů. Windows může pracovat na více procesorech najednou, ale vždy má jenom jednu instanci v celém systému. [23, 24]

Jak již bylo zmíněno, TenAsys nabízí i řešení pro distribuované systémy, kde spolu komunikují jednotlivé platformy s Windows nebo INtime přes Ethernet, PCI nebo RS-232. Slouží k tomu nástroj s názvem Distributed Systems Manager (DSM), který monitoruje stav jednotlivých instancí operačních systémů a řídí jejich procesy. Meziprocesová komunikace je zajištěna pomocí GOBSnet tak, že procesy systému INtime na různých platformách mezi sebou komunikují, jako kdyby byly na jednom procesoru. Tato komunikace splňuje Time-Sentitive Networking standard, konkrétně IEEE 1588. Co se týče komunikace mezi INtime a Windows, je použito Windows API s názvem NTX, a to jak pro komunikaci na stejné platformě, tak při síťové komunikaci. [23, 24]

INtime vyžaduje architekturu procesoru x86 a vyhrazených 16 MB RAM pro svůj kernel. Distribuované systémy musí mít kompatibilní síťový hardware (Intel, Realtek nebo Broadcom) a jednu ze sběrnic/médií PATA, SATA, NVMe nebo eMMC na načítání INtime RTOS. Je zde podpora jazyka C++ a jeho knihoven, hlavním vývojovým prostředím je Microsoft Visual Studio. [25]

Oproti předchozím produktům založeným na hypervisorech nemá INtime žádné bezpečnostní certifikace a jeho nejnovější verze nepodporuje sdílení hardwarových prostředků, ani komunikaci pomocí sdílené paměti.

3.8 IntervalZero – RTX64

Podobným produktem jako předchozí InTime je RTX64 firmy Interval Zero. RTX64 je také RTOS rozšíření pro Microsoft Windows. Výrobce používá označení subsystém reálného času (RTSS), který funguje paralelně vedle kernelu a plánovače Windowsu, rovněž bez použití hypervisoru. [26]

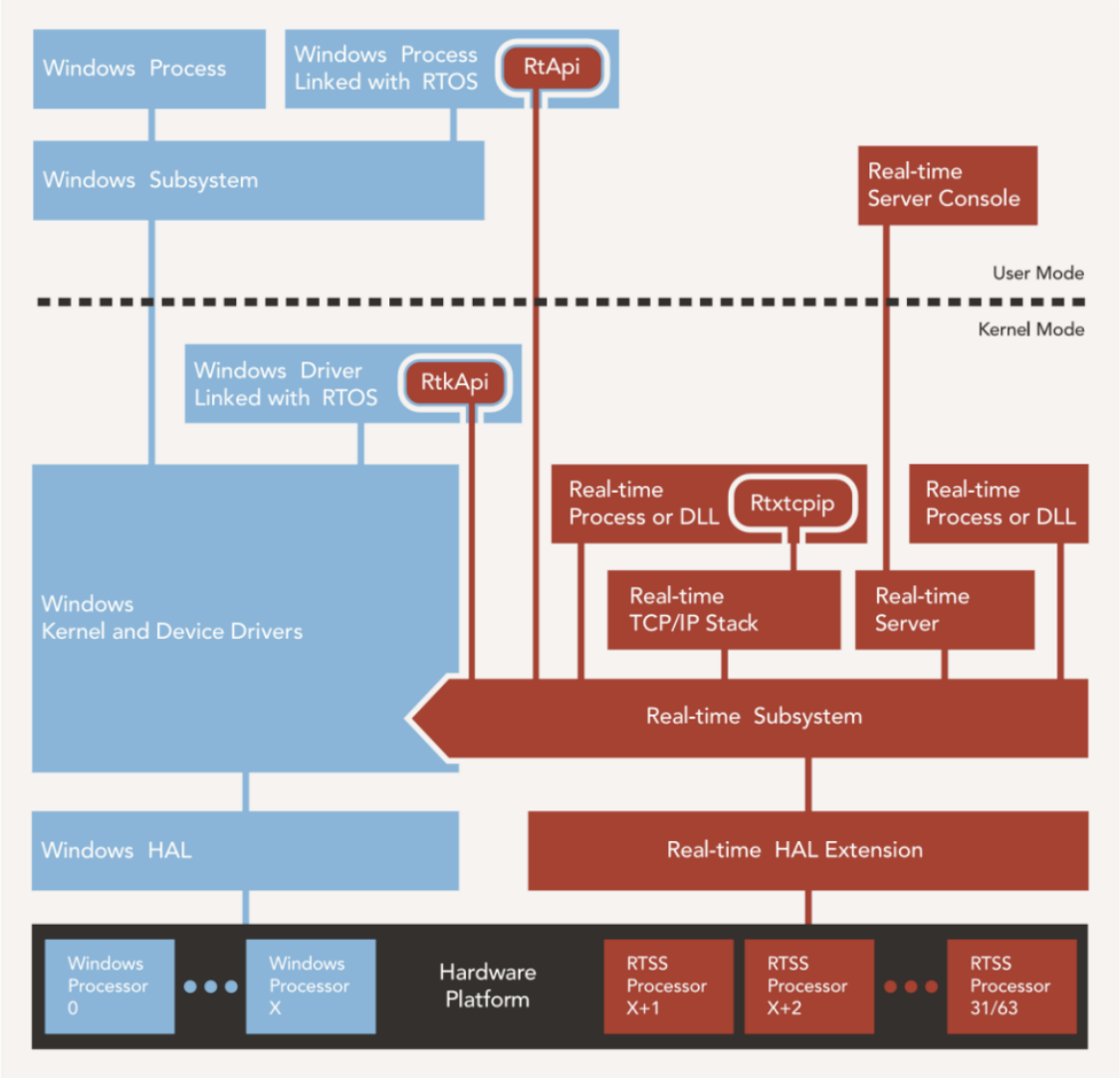
Přístup k hardwarovým zařízením je zajištěn pomocí hardwarové abstrakční vrstvy (HAL), kterou poskytuje RTX64. HAL používající RTX64 dovoluje operačnímu systému přístup k jednomu až 63 procesorovým jádrům. Plánovač pro RTX64 jádrům přímo přiřazuje vlákna reálného času. GPOS i RTOS zde používají symetrický multiprocesing a přistupují k dostupným procesorům jako k jednomu celku. Není tedy nutné mít na jednom zařízení více než jednu instanci každého ze dvou operačních systémů, tento přístup je znázorněn na obrázku. [27, 28]

RTOS plánovač podporuje prioritní obsluhu přerušení a zaručuje, aby nenastávaly konflikty s Windows. RTX64 může přistupovat k až 128 GB nestránkované paměti a k 512 GB fyzické paměti ze všech jader a bez nutnosti rozdělování paměti, což umožňuje celému systému sdílení celé paměti. Výrobce zaručuje při dostatečně výkonném hardwaru rychlost odezvy až 1 mikrosekundu při splnění požadavků kritického reálného času. Výměna dat mezi operačními systémy probíhá pomocí sdílené paměti. V případě selhání Windowsu fungují RTOS aplikace i nadále. [27, 28]

Pro vývoj RTOS i GPOS aplikací je možno použít například Microsoft Visual Studio i jiné prostředky dostupné pro Windows prostředí, podporovanými programovacími jazyky jsou C a C++. RTX64 může fungovat na všech platformách, na kterých může fungovat Windows 10. Tento systém získal dva bezpečnostní certifikáty a to SIL-3 a FDA Class 2. [27, 28]

3.9 Shrnutí

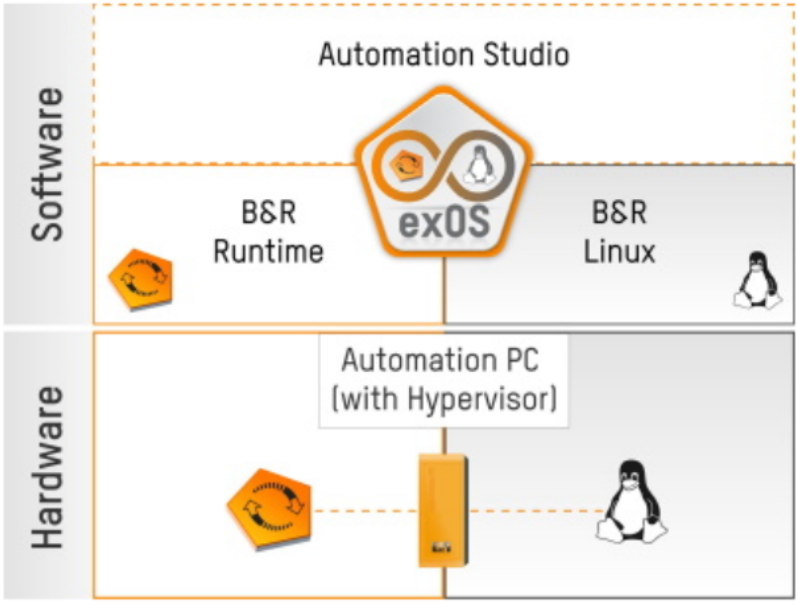
Při výběru hypervisoru nebo RTOS rozšíření záleží převážně na požadavcích konkrétní aplikace, například na dostupném hardwaru, nutnosti bezpečnostních certifikací, preferovaných operačních systémech (a možnosti jejich kombinací), výpočetní náročnosti, vývojových prostředích, programovacích jazycích apod. Řešení s hypervisory obecně umožňují sdílení hardwarových periférií, zatímco RTOS rozšíření, ve kterých běží RTOS a GPOS kernely paralelně, používají pevně nastavené hardwarové oddíly. Jejich výhodou je ale menší zatížení systému díky absenci hypervisoru.



Obr. 16: Architektura RTOS rozšíření RTX64 [28]

4 EXOS

Název softwarového produktu exOS je zkratkou pro enhanced cross-over for Operating Systems. [29] Je určen k propojení operačního systému Automation Runtime (RTOS) a Linuxu (GPOS), ilustrováno na obr. 17. Zatímco AR je systém tvrdého reálného času vhodný například pro ovládání pohybu průmyslových zařízení a podporuje programovací jazyky podle IEC, Linux nabízí velký výběr různých frameworků a programovacích jazyků, které mohou být vhodné pro použití na některých částech průmyslových projektů. Struktura obsahující proměnné, jež se používají pro komunikaci mezi oběma systémy, se označuje jako dataset. Většina informací pro tuto kapitolu je čerpána z elektronické dokumentace [29] a ze zkušeností nabytých během práce s exOS.



Obr. 17: Role exOS v automatizačním projektu [29]

4.1 Podporované platformy

Cílovými platformami jsou průmyslové počítače B&R s procesory značky Intel. exOS lze použít v Automation Studiu verze alespoň 4.10 s AR ve verzi B4.91 a vyšší. Podporovaným a otestovaným GPOS je B&R Linux 10, což je varianta Debianu 10 (busteru), která je optimalizovaná pro běh na B&R průmyslových počítačích. [29, 30]

Pro funkci na průmyslových počítačích se využívá B&R Hypervisor (1. typu). Krom oddělení obou operačních systémů se hypervisor stará i o vzájemnou synchronizaci procesů a výměnu dat pomocí sdílené paměti, která probíhá v řádu milisekund. Vývoj aplikací je možné provádět i bez použití průmyslového počítače. V takovém

případě výměna dat mezi GPOS a RTOS neprobíhá přes sdílenou paměť, ale přes TCP/IP socket. Díky abstrakčním vrstvám exOS může kód fungovat stejně ve skutečné konfiguraci i v té simulované, není tedy nutné po zprovoznění v simulovaném prostředí projekt nijak upravovat pro zprovoznění na skutečném cílovém systému. Automation Studio umožňuje pomocí systému ARsim simulovat skutečný Automation Runtime na platformě používající operační systém Windows.

Co se týče strany GPOS, tak exOS sice nepodporuje Windows, ale je možné propojení s virtuálním počítačem s nainstalovaným B&R Linuxem 10 běžícím na Windowsu. Jednou z možností, jak tohoto dosáhnout, je použití některého z dostupných virtualizačních nástrojů, jakými jsou například VMWare nebo VirtualBox. Další možností, která je doporučená v dokumentaci exOS a byla využita i v aplikaci vyvinuté v rámci této práce, je WSL (Windows Subsystem for Linux) dostupné pro Windows 10 a 11. S využití nativního hypervisoru Hyper-V běží na osobním počítači vedle Windowsu i kernel kompatibilní s Linuxem. Toto řešení se označuje jako virtuální počítač "lehké váhy", protože pro tuto funkci není použit plnohodnotný virtuální počítač, což má za důsledek menší ztrátu výkonu. Hyper-V sice není dostupný na všech distribucích Windows (není obsažen v edicích Home), ale i tak je možné použít WSL, což bude ukazáno v podkapitole 4.3. WSL má přístup k Windows souborům a může využít hardwarovou akceleraci. Další výhodou WSL je také integrace s vývojovým studiem VS Code, které se k dané instanci WSL může jednoduše připojit a programátor tak může vyvíjet Linuxové aplikace v prostředí Windows. Kromě simulace cílového zařízení se WSL využívá i pro kompilaci zdrojových programů pro Linux.

4.2 Hlavní vlastnosti

Díky exOS se k softwaru určeném pro Linux přistupuje jako k jednomu z komponent projektu v Automation Studiu a tudíž je možné jednotná správa Linux i AR kódu.

4.2.1 Přesun souborů

Jednou z vlastností tohoto komponentu je automatický přesun kódu a ostatních souborů přímo do Linuxu běžícího na cílovém systému. Pokud dojde ke změně nějakého ze souborů v rámci AS projektu, exOS nabízí možnost automatického updatu i na straně Linuxu. K této funkci je využit exOS server.

4.2.2 Vykonání příkazů

Také je možné definovat operace, které se s danými soubory mají provést v různých časových okamžicích běhu aplikace. Například lze zadat příkazy, které po přenosu projektu nainstalují potřebné knihovny, nebo automaticky spustí požadované

soubory. Stejně jako pro přesun souborů je vykonávání příkazů umožněno pomocí komunikace přes exOS server.

4.2.3 Meziprocesová komunikace

Jak již bylo zmíněno, exOS nabízí zprostředkování komunikace mezi procesy RTOS a GPOS jak ve skutečném prostředí tak i v simulovaném. Tato meziprocesová komunikace funguje na základě událostí vyvolávaných změnami proměnných. Dále exOS obsahuje buffer, díky kterému je zaručeno doručení dat.

4.2.4 Diagnostika

V průběhu vývoje aplikace je důležité využití diagnostických nástrojů. Jedním z nich je posílání zpráv do centralizovaného Loggeru v Automation Studiu. Dále je v programovacím prostředí VS Code (Visual Studio Code) k dispozici debugovací konzole, zprostředkovaná pomocí stáhnutelného rozšíření, které bude popsáno později.

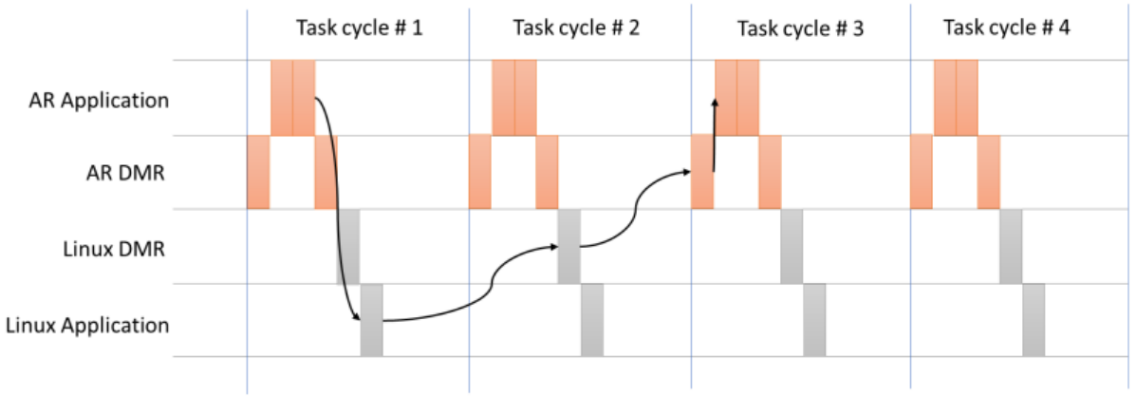
V loggeru jsou vidět různé zprávy z obou operačních systémů, které se týkají exOS komunikace i kompilačního procesu. Debugovací konzole nabízí i další informace navíc, které pocházejí z DMR (Dataset Message Router) obou operačních systémů i z programů běžících na Linuxu. Pro obecný přehled o statusech systému je možné použít diagnostické funkční bloky, které zobrazují mimo jiné i údaje o výkonnosti, jako je množství přenesených dat, přesnost časové synchronizace, čas strávený čekáním apod.

4.2.5 Synchronizace

Pro některé aplikace může být důležité mít synchronizovaný čas. V AR se základní časová stopa nazývá NETTIME, vždy inkrementuje směrem nahoru a má rozlišení jednu mikrosekundu. NETTIME je zprostředkován i Linuxu, ve kterém se podle něj mohou řídit různé operace.

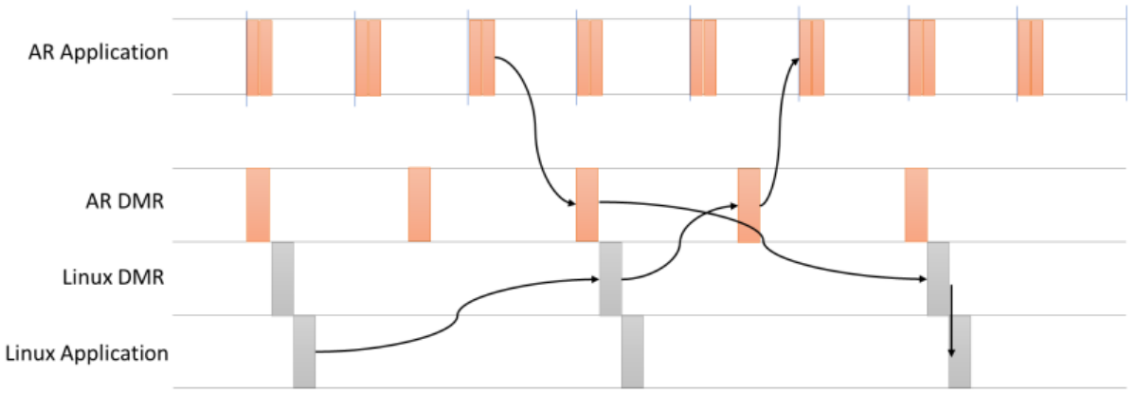
Kromě času je možné synchronizovat i procesy běžící na Linuxu s daným taskem DMR, který se stará o komunikaci. Vzhledem k nedeterminismu Linuxu nelze úplnou synchronizaci zaručit, pokud se ale DMR task nevykonává s příliš vysokou frekvencí, ve které by přetěžoval proces běžící na Linuxu, synchronizace může být obecně dosaženo.

V konfiguraci s hypervisorem dojde ke spuštění AR DMR na konci daného tasku. AR DMR obdrží od aplikace daný dataset, podle něj změní hodnoty příslušných proměnných ve sdílené paměti a nastaví synchronizační událost. Na základě této události se začne vykonávat Linux DMR, který ze sdílené paměti dodá data procesům na Linuxu a poté pošle synchronizační zprávu NETTIME. Po obdržení této zprávy vykonají Linuxové aplikace požadované funkce a případně se celý tento proces opakuje z druhého směru. Tento proces je znázorněn na obr. 18.



Obr. 18: Synchronizace procesů s hypervisorem [29]

Při použití simulace není synchronizace dosažitelná z důvodu obousměrného zpoždění TCP (obr. 19). DMR na straně AR přizpůsobuje periodu svého vykonávání podle aktuální velikosti tohoto zpoždění. DMR Linuxu se opět vykonává na základě dokončení AR DMR, tudíž běží kontrolovaně, ale asynchronně vůči systému taskových tříd v AR.



Obr. 19: Synchronizace procesů v simulaci [29]

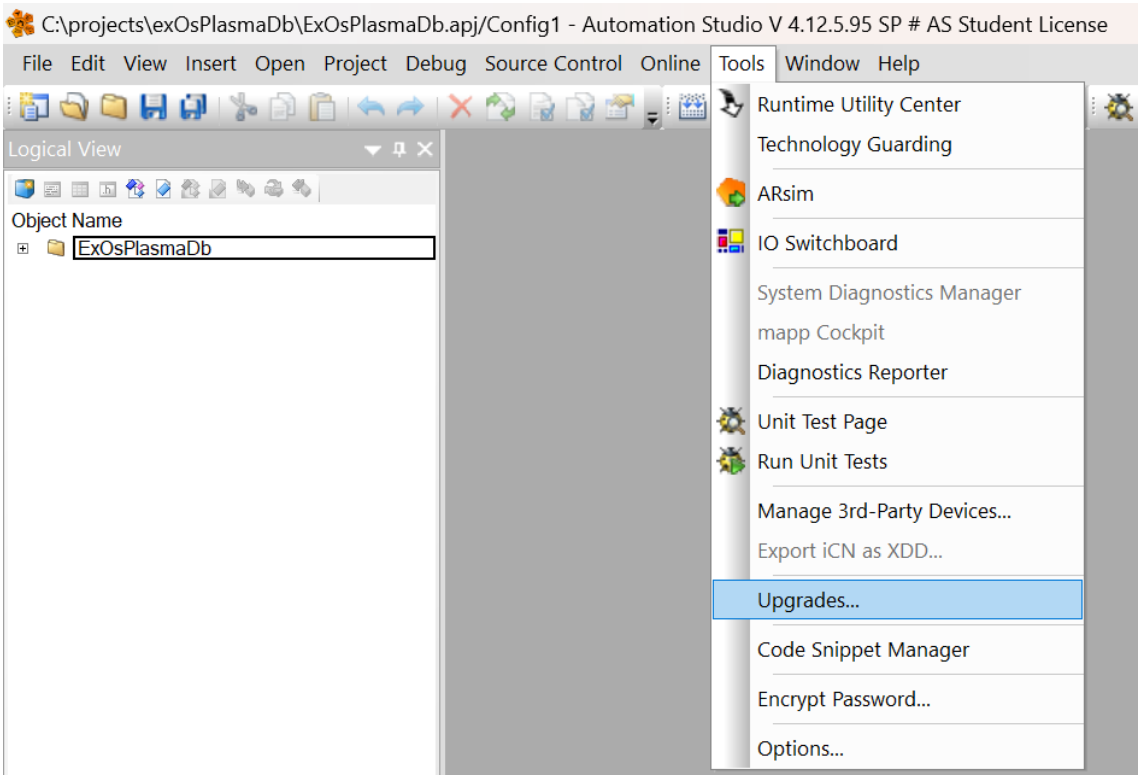
4.3 Instalace

V této podkapitole jsou popsány kroky potřebné k úspěšné instalaci produktu exOS.

4.3.1 Technologický balíček

Pro použití exOS je nejprve nutné do Automation Studia nainstalovat balíček exOS Technology. Ten je dostupný ke stažení přímo ze stránek B&R (nelze stáhnout pomocí AS). Po jeho stažení se v AS v záložce Tools vybere položka Upgrades (obr. 20). Ta otevře okno (obr. 21), ve kterém se zvolí možnost Local, vybere se adresář ob-

sahující stažený exOS balíček, ten se označí a klikne se na tlačítko Install Selected Upgrades.



Obr. 20: Otevření okna Upgrades

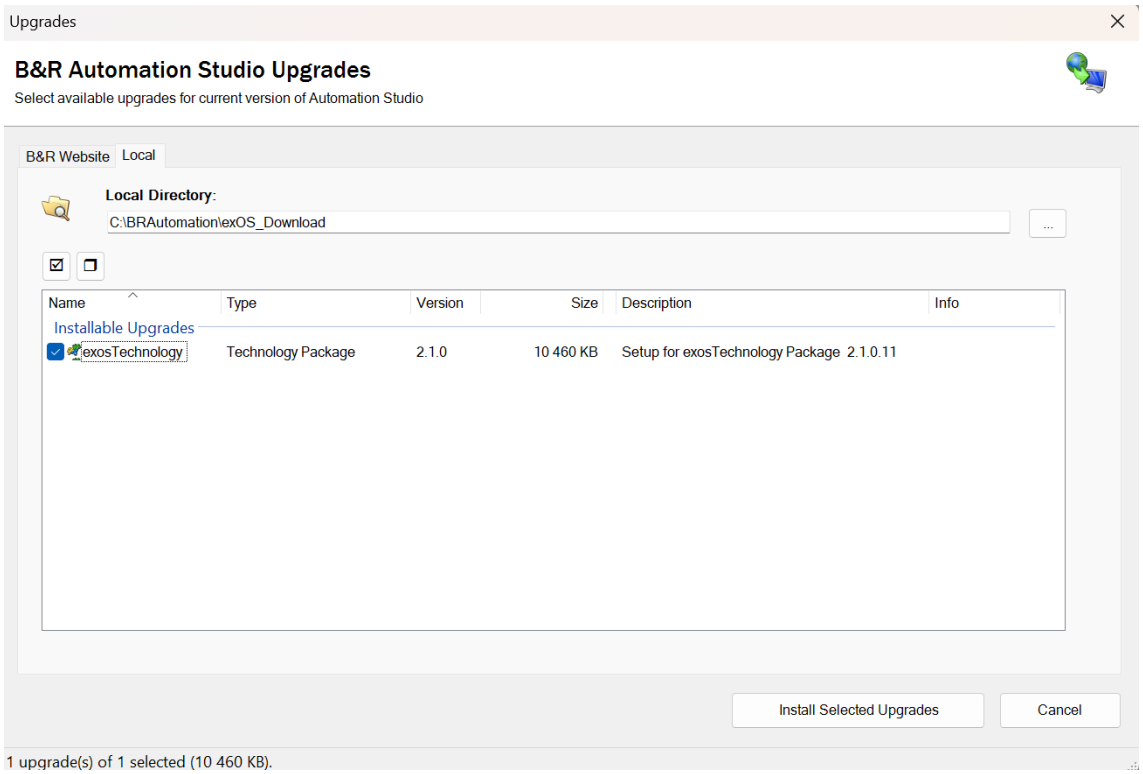
4.3.2 Server

Jak již bylo zmíněno, pro přesouvání souborů a vykonávání Linux příkazů se používá exOS server, který je potřeba nainstalovat. Tento krok je možné přeskočit při instalaci z obrazu Linuxu, ve kterém je server již obsažen. Server je využit i pro výměnu dat v simulaci, server ale je součástí oficiálního obrazu pro simulaci cílového zařízení na WSL.

Balíček pro instalaci serveru je obsažen ve složce technologického balíčku v podsložkách Modules/Server. Při instalaci na průmyslovém počítači je nutné do Linuxu tento balíček přesunout (například pomocí USB) a poté použít příkaz "sudo dpkg -i exos-server-x.y.z-b_amd64.deb", kde x.y.z-b značí verzi serveru (v době psaní této práce je nejnovější 2.1.0-1). Pro správnou instalaci musí počítač mít již nainstalované ovladače pro B&R Hypervisor. Při instalaci se vybere IP adresa virtuálního síťového rozhraní, která musí být také zadána v konfiguraci cíle.

4.3.3 WSL

V práci je použita nejnovější verze Windows Subsystem for Linux, tedy WSL 2. Samotná instalace WSL je velmi jednoduchá, provádí se příkazem “wsl –install”



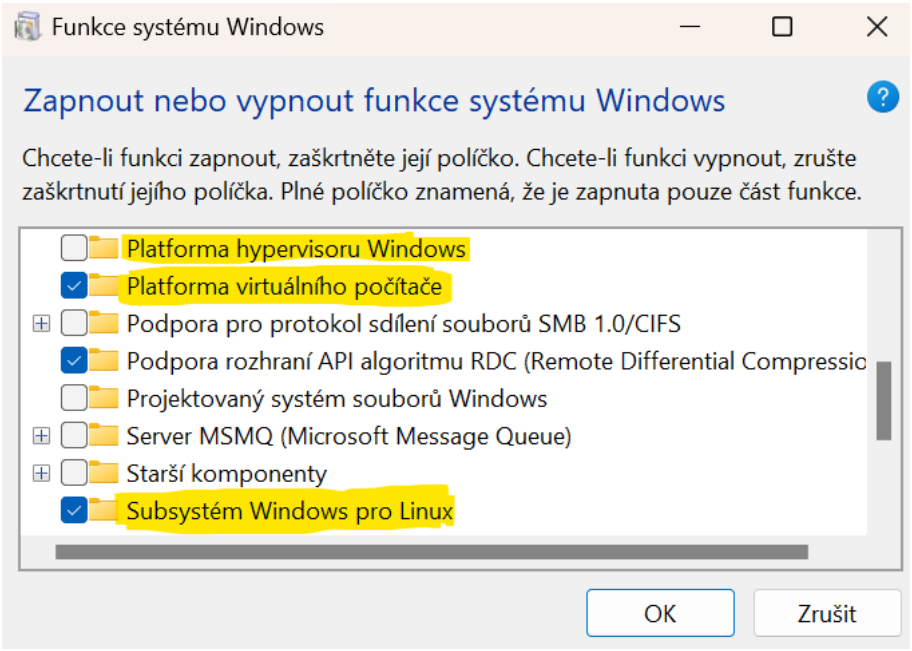
Obr. 21: Instalace exos Technology Package

zadaným do přikazového řádku nebo PowerShellu, poté je potřeba provést restart počítače.

Byl použit Windows 11 Home, který nepodporuje Hyper-V, pro funkci WSL byla tedy použita platforma virtuálního počítače. Pro toto nastavení je nutné v Ovládacích panelech Windowsu zvolit položku "Položky a funkce" a poté "Zapnout nebo vypnout funkce systému Windows". V otevřeném okně je potřeba správně nastavit funkce zvýrazněné na obr. 22.

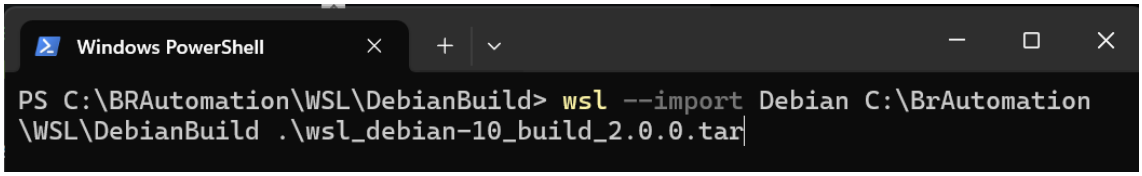
Dalším krokem po nainstalování je import Linuxových obrazů do WSL. Jeden obraz slouží pro kompilaci zdrojových kódů a druhý pro simulaci cílového zařízení. Tyto obrazy jsou k dispozici ke stažení na GitHubu br-automation-com/exOS-WSL. Po jejich stažení se ve složce s daným souborem v příkazovém řádku vykoná příkaz wsl –import, následuje pojmenování nově vzniklé instance WSL, cesta k souboru a jeho spuštění (obr. 23). Pro kompilaci souborů je dále potřeba ve WSL spustit skript na nastavení kompilačního prostředí (obr. 24).

V době implementace exOS nepodporovalo WSL SystemD (System and Service Manager), což je jeden ze základních funkčních manažerů v Linuxu, který spravuje služby a sockety [31], což je důležité pro chod serverů, jakými jsou právě například MariaDB. Kvůli zprovoznění exOS serveru byla tato absence řešena pomocí genie, což je program zastupující funkce SystemD. Sám autor programu ale genie označuje za poněkud neohrabaný [32], což je znát například při spouštění exOS

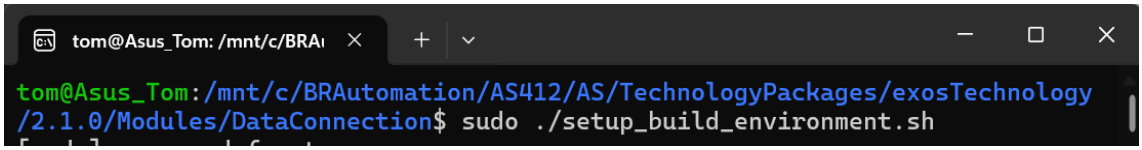


Obr. 22: Nastavení funkcí systému Windows 11 Home

serveru na WSL pro simulaci, kdy se velmi dlouhou dobu čeká právě na SystemD. I co se týče zapínáni serveru MariaDB, tak nastávaly určité problémy. Například pro spuštění serveru bylo potřeba používat jiné příkazy anebo WSL nejdříve restartovat, tudíž to znepříjemňovalo vývoj, co se týče automatické instalace z AS. SystemD už ale je ve WSL 2 nativně podporován, pouze se musí povolit jeho spuštění po startu. To se udělá zapsáním:
[boot]
systemd=true
do konfiguračního souboru wsl pomocí příkazu "sudo nano /etc/wsl.conf".



Obr. 23: Import Linux obrazu do WSL



Obr. 24: Spuštění skriptu nastavujícího kompilační prostředí

4.4 Rozšíření exOS pro VS Code

Pro jedno z nejpoužívanějších vývojových prostředí VS Code od společnosti Microsoft je k dispozici rozšíření s názvem exOS Component Generator, které usnadňuje práci s exOS. Jeho hlavní funkcí je vytváření základu (softwarových balíčků, šablon a funkčních bloků pro zajištění komunikace) pro exOS projekt podle souboru definujícího datové typy a struktury .typ, na základě kterého se vytváří i dataset. Z pohledu AR je možné vytvořit šablonu pro komunikaci buď v jazyce C nebo C++, na straně Linuxu mohou být použity uvedené jazyky, JavaScript modul používajíc NodeJS anebo Python modul, který využívá rozhraní v jazyce C, se kterým komunikuje pomocí knihovny SWIG. V případě změny v datasetu je také možné všechny potřebné soubory patřičně aktualizovat.

Další nabízenou funkcí je vytváření binárních balíčků pro Linux. Ty se dají použít již bez nutnosti kompilace, tudíž se hodí například v projektech, kde jsou týmy rozdělené na Linux a AR vývojáře. Nakonec je pomocí tohoto rozšíření také možné zapnout debugovací konzoli.

5 MODELOVÁ APLIKACE

Tato kapitola se věnuje vytvoření a zprovoznění modelové aplikace s využitím exOS. Byla vybrána aplikace založená na reálném projektu firmy B&R ovládající plasmovou řezačku.

Na základě typu plasmy, zpracovávaného materiálu a jeho tloušťce se operátorovi zobrazí výběr tzv. proudových sestav. Tyto sestavy obsahují celkem 55 různých údajů ohledně nastavení plasmové řezačky. Jsou jimi například materiál a jeho tloušťka, pro kterou je sestava nastavena, dále nastavení elektrického proudu a napětí, rychlosti pohybu řezačky v různých stavech (řezání, propal, sjetí), doby setrvání v určitém stavu, výšky řezáku nad materiálem, použité plyny a mechanické díly, nastavení tlaků a průtoků a další.

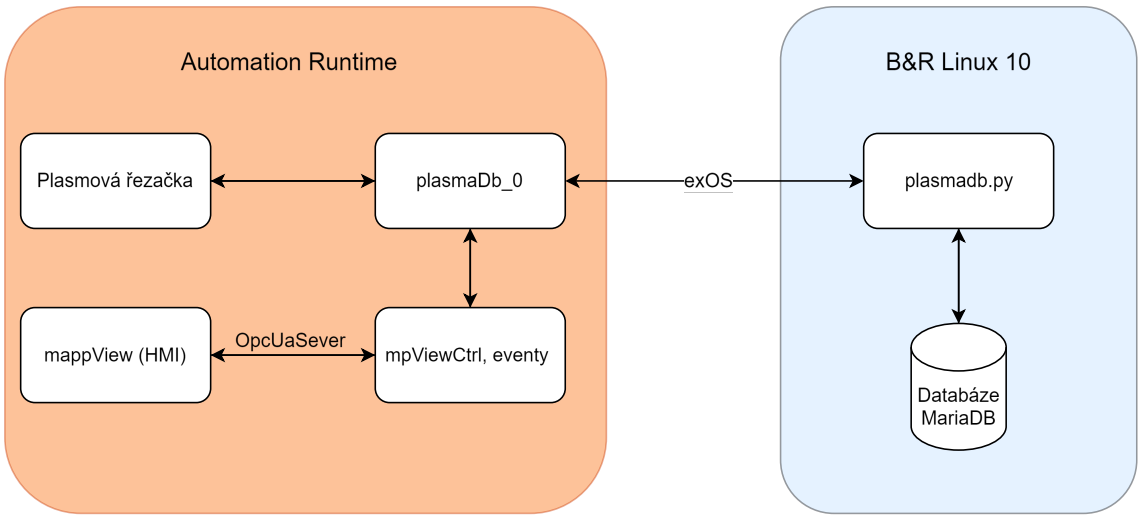
Původní řešení bylo zpracováno tak, že se na průmyslovém počítači RTOS (AR) staral o řízení řezačky a na tom samém počítači byly v rámci GPOS (Windows) uloženy tabulky ve formátu csv. Po výběru údajů určujících proudovou sestavu se tabulky přenášely do AR, kde poté byly zpracovány. To ale zabíralo velké množství času (i v řádu minut) hlavně kvůli velikosti tabulek. Mohla nastat i situace, kdy se po dlouhém čekání nakonec ani žádné údaje nepřenesly. Pro vyřešení tohoto problému je vhodné místo csv tabulek využít relační databázi, díky níž je možné zrychlit proces hledání požadovaných údajů a také výrazně zmenšit množství přenášených dat. Právě databáze patří mezi jedno z inzerovaných využití exOS, protože jejich servery běží na GPOS a práce s nimi se často provádí pomocí vyšších programovacích jazyků, které RTOS nepodporují.

V rámci této práce byla vytvořena relační databáze obsahující potřebná data, se kterou se pracuje pomocí jazyku Python na na straně Linuxu. Na straně AR pak byla vytvořena vizualizace zobrazující některá data a obslužné programy umožňující komunikaci s Linuxem pomocí exOS, obecné schéma architektury této aplikace je vidět na obr. 25. Vývojový diagram zobrazující obvyklý způsob práce s projektem a databází je poté zobrazen na obr. 26.

5.1 Vytvoření základní šablony projektu

Prvním krokem je instalace technologického balíčku exOS a Linuxových instancí ve WSL, popsána v předchozí kapitole, a vytvoření projektu v Automation Studiu, který v konfiguraci používá průmyslový počítač s podporou hypervisoru. Dále je vhodné definovat struktury datových typů, pomocí kterých bude Linux a AR komunikovat.

Na obr. 27 je zobrazen soubor .typ obsahující použité struktury. Hlavní strukturou (datasetem) je *plasmaDb*, obsahující všechny ostatní struktury a základ pro



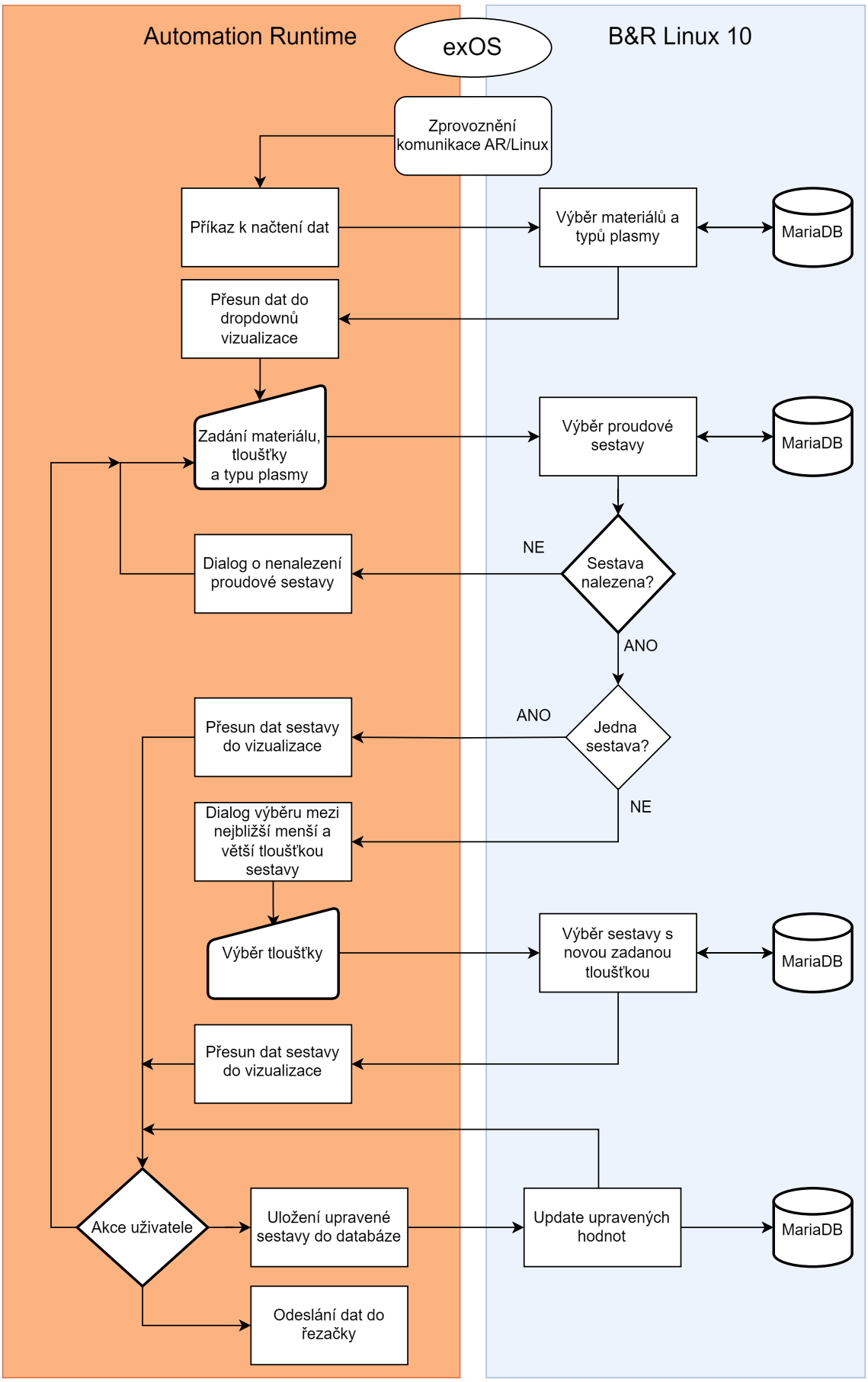
Obr. 25: Architektura aplikace

komunikaci mezi systémy. V této struktuře se definuje, které struktury jsou typu PUB (publisher) nebo SUB (subscriber) z pohledu AR. To značí, jestli se mají do Linuxu posílat (PUB) nebo přijímat (SUB). Je také možné, aby jedna struktura byla jak publisher tak subscriber.

Popis struktur a proměnných:

- **plasma_profile** obsahuje všechny jednotlivé parametry proudové sestavy, použité typy jsou REAL pro desetinná čísla (napětí, tlaky, průtoky apod.), INT pro identifikační čísla a STRING pro značení plynů a mechanických dílů
- **mpview_params** se skládá z údajů sloužících pro výběr dat z databáze, tedy unikátního id typu plasmy, materiálu a sestavy a nakonec šířky materiáliu
- **drop_down_provider** se vytváří na straně Linuxu a obsahuje stringy obsahující názvy a hodnoty pro výběr pomocí dropdown selektorů ve vizualizaci
- **choose_width_dialog** se používá pro případ, kdy zvolená šířka materiálu není dostupná mezi sestavami v databázi, uživatel je tedy vyzván k výběru mezi nejbližší větší a menší šířkou
- **refresh_data_cmd** je příkaz ze strany AR, použitý k zobrazení načtení všech aktuálně dostupných dat z Linuxu
- **wait_for_gpos** je pomocná proměnná značící probíhající operaci na straně Linuxu
- **update_table_cmd** - příkaz k přepsání změněných hodnot proudové sestavy ve vizualizaci do databáze

Ve VS Code se pomocí zmíněného rozšíření exOS Component Generator vygenerují základní soubory potřebné ke komunikaci tak, že se na daný .typ soubor klikne pravým tlačítkem a zvolí se možnost *Create exOS package*. Poté se zvolí



Obr. 26: Vývojový diagram aplikace

plasmaDb::plasmaDb.typ [Data Type Declaration]

Name	Type	&	\	Description [1]
plasma_profile_type			☑	
mpview_params_type			☑	
plasma_id	STRING[10]	☐	☑	
material_id	STRING[10]	☐	☑	
profile_dropdown_id	STRING[10]	☐	☑	
width	REAL	☐	☑	
drop_down_provider_type			☑	
material	STRING[80][0..14]	☐	☑	
plasma_profiles	STRING[80][0..14]	☐	☑	
plasma_types	STRING[80][0..29]	☐	☑	
choose_width_dialog_type			☑	
show_dialog	BOOL	☐	☑	
bigger	REAL	☐	☑	
smaller	REAL	☐	☑	
bigger_chosen	BOOL	☐	☑	
smaller_chosen	BOOL	☐	☑	
plasmaDb			☑	
plasma_profile	plasma_profile_type	☐	☑	PUB SUB
mpview_params	mpview_params_type	☐	☑	PUB SUB
drop_down_provider	drop_down_provider_type	☐	☑	SUB
choose_width_dialog	choose_width_dialog_type	☐	☑	PUB SUB
refresh_data_cmd	BOOL	☐	☑	PUB SUB
wait_for_gpos	BOOL	☐	☑	PUB SUB
update_table_cmd	BOOL	☐	☑	

Obr. 27: Soubor typu .typ popisující hlavní dataset

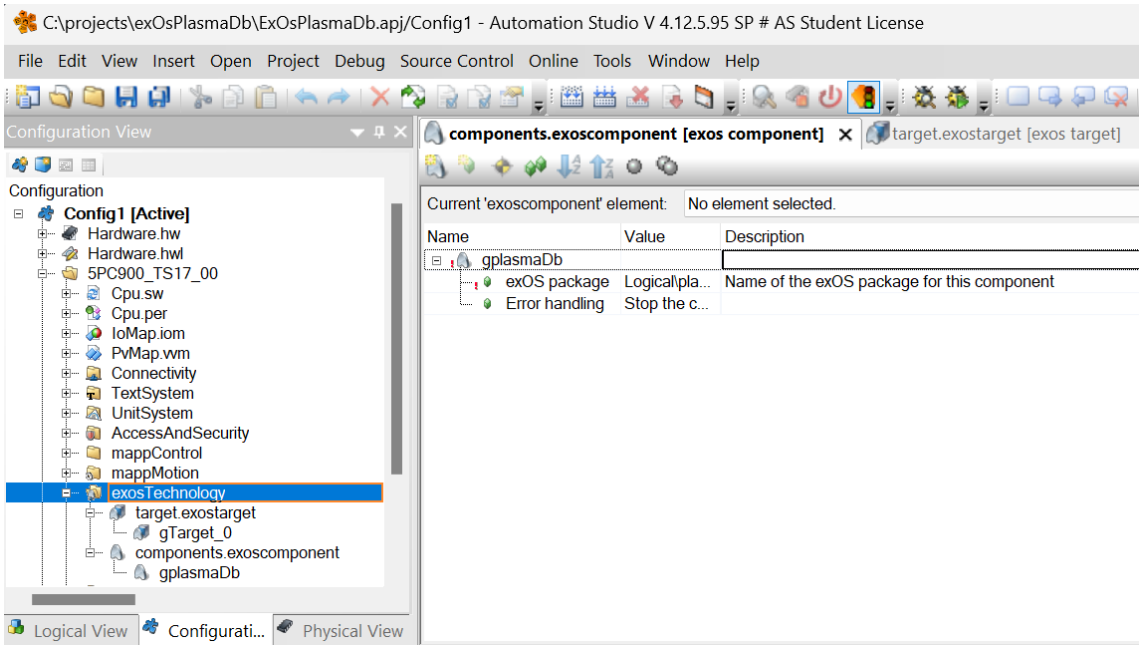
hlavní struktura datasetu (*plasmaDb*), dále se vybere šablona pro komunikaci AR (byla vybrána C++ třída) a pro komunikaci ze strany Linuxu (k tomuto účelu byl zvolen modul jazyka Python 3). U generování Python modulu je důležité, aby byla ve VS Code použita stejná verze Pythonu jako na cílovém zařízení. Aktuální verze B&R Linuxu vychází z Debianu 10, který v základu podporuje Python verze 3.7, tudíž bylo nutné v rámci VS Codu provést downgrade z aktuální verze 3.12. Druhou možností by bylo použití jednoho z nástrojů pro instalaci Pythonu vyšší verze (jako například pyenv), což by ale komplikovalo proces instalace na cílovém zařízení. Protože Python verze 3.7 byla dostatečná pro obsluhu databáze, byla použita.

5.2 Konfigurace exOS

Do souboru exosTechnology v konfiguraci AS projektu je potřeba přidat komponenty exostarget a exoscomponent.

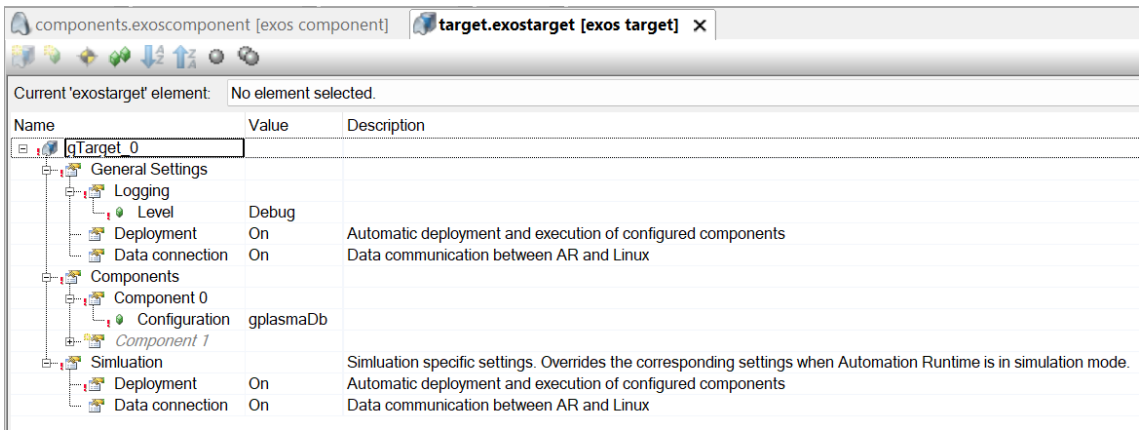
Exoscomponent (obr. 28) musí mít v kolonce *exOS package* název hlavního exOS balíčku. Obsahuje také nastavení chování při chybách vzniklých v programech

Linuxu, při přesunu souborů a vykonávání Linux příkazů. Při vybrání pokročilých parametrů jsou zde zobrazeny i soubory určené k přesunu a seznam příkazů, toto nastavení bude popsáno později.



Obr. 28: Konfigurace exoscomponent

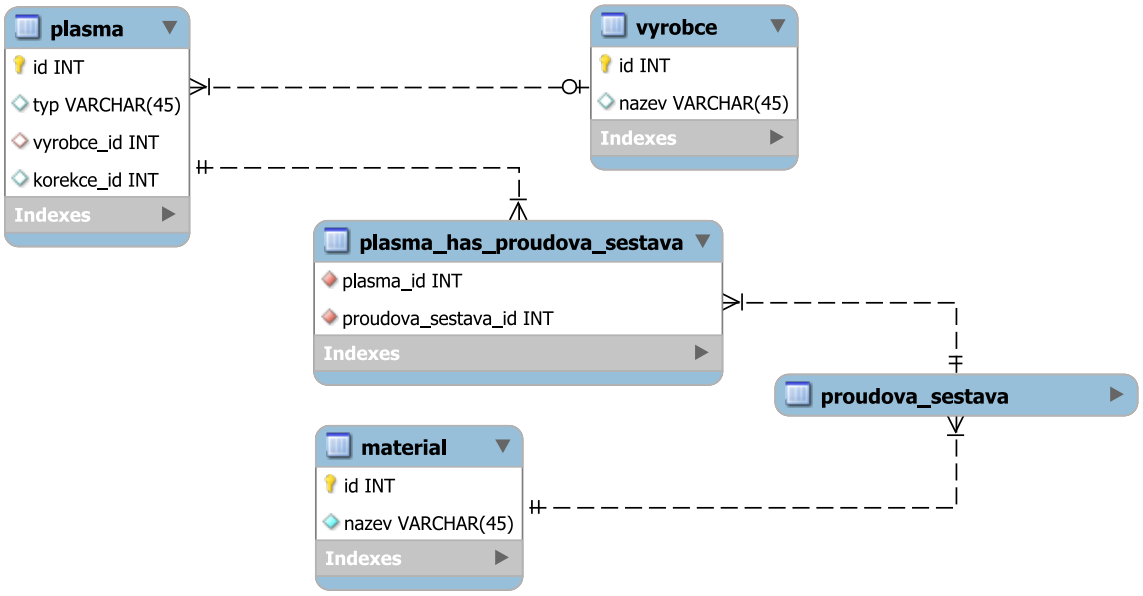
Komponent exostarget (obr. 29) definuje úroveň výpisu informací v loggeru a konzoli a nastavení pro nasazení komponentů ve skutečném i simulovaném prostředí. Při použití WSL určené k simulaci cílového zařízení je nutné mít v sekci *Simulation Deployment* i *Data connection* zapnuté.



Obr. 29: Konfigurace exostarget

5.3 Databáze

Jako relační databáze byla vybrána MariaDB, což je komunitní větev databáze MySQL. Pro vytvoření základního diagramu (obr. 30) byl využit nástroj MySQL Workbench. Všechny tabulky obsahují unikátní identifikační číslo (id).



Obr. 30: Diagram vytvořené databáze

Tabulky databáze:

- **plasma**
  - typ - označení konkrétního druhu plasmy určené výrobcem
  - vyrobce_id - odkaz na tabulku výrobce
  - korekce_id - prozatím nevyužitý parametr, který ale v budoucnu bude použit k propojení plasmy s další tabulkou obsahující mechanické korekce pro technologii 3D plasmového řezání
- **vyrobce** - obsahuje název výrobce ve formě zkratky, který se přidává před název plasmy zobrazený ve vizualizaci
- **proudova_sestava** - všechny údaje a nastavení konkrétní proudové sestavy
- **plasma_has_proudova_sestava** - tabulka realizující M:N vztah mezi tabulkou plasma a proudova_sestava, jedna konkrétní plasma totiž má mnoho různých proudových sestav, zároveň proudová sestava může být použita pro více typů plasmy (toho aktuálně není využito, ale na základě konzultace byla tato možnost ponechána)
- **material** - název zpracovávaného materiálu, který je pomocí id spojen s konkrétními proudovými sestavavami

Pro import dat z csv souborů byl vytvořen Python program, který ze složky se soubory automaticky přidá proudové sestavy (v souborech csv představuje jeden řádek jednu proudovou sestavu) a podle názvu konkrétního souboru přidává do tabulky *plasma* jednotlivé typy. Je také vytvořen m:n vztah mezi danými sestavami a plasmami. Při tomto procesu bylo zjištěno, že v dodaných souborech se vyskytují chyby, jakými jsou například nulové hodnoty tloušťky materiálu, id materiálu anebo jednoho z hlavních plynů apod. Vzhledem k tomu, že bez těchto údajů je proudová sestava neplatná (a kvůli nastavením omezení hodnot v databázi ani nejde přidat), tak program ošetřuje i tyto stavy, kdy do konzole vypíše název souboru a řádek, který obsahuje chybu. Sestava tedy do databáze není přidána.

5.4 Strana Linuxu

Hlavním programem Linuxu je *plasmadb.py*. Obsahuje základní metody a třídy ze souboru *libplasmaDb.py*, který byl automaticky vygenerován pomocí rozšíření exOS Component Generator.

5.4.1 exOS komunikace

Hlavní třídou je *plasmaDbEventHandler*. Jejími primárními metodami jsou *connect()*, *disconnect()* apod., které se starají o zajištění komunikace s AR. Metoda *process()* se volá cyklicky a reaguje na příchozí zprávy v bufferu. Pro možnosti debugování a zobrazení zpráv v konzoli a loggeru je k dispozici několik metod loggování v různých úrovních (které se dají vypnout/zapnout v konfiguraci v AS).

Každá struktura v .typ souboru, která je je obsažena v hlavní struktuře je také atributem (datasetem) hlavní třídy a podle svého typu (publisher/subscriber - z pohledu AR) má určité metody. Všechny tyto datasety mají atribut value, odkazující se na skutečnou hodnotu dané proměnné, je buď jedním ze základních datových typů anebo strukturou. Subscribeři mají metodu *publish()* posílající hodnotu přes exOS do AR.

Publisheři disponují metodou *on_change_...*, která se volá při změně hodnoty (nebo některé z hodnot) konkrétního datasetu v AR. Dále je zde atribut *nettime*, který udává časový údaj změny datasetu a může být využit k synchronizaci. Jak již bylo zmíněno, datasety mohou být typu publisher i subscriber, v takovém případu obsahuje dataset všechny popsané prvky.

5.4.2 Komunikace s databází

Dále byly naprogramovány metody pracující s databází (připojení, výběrové dotazy v jazyce SQL, update hodnot) a připravující data pro AR a vizualizaci. Ve vizualizaci je k výběru konkrétní plasmy, materiálu a proudové sestavy používáno

tzv. dropdownů, které přijímají data typu pole stringů ve formě obsahující hodnotu (zadanou jako id prvku v databázi) a název, který je zobrazen jako text.

Funkce:

- **connect_DB** - připojení se k databázi
- **provide_plasma_types** - vrací dostupné typy plasmy v databázi ve formě pro dropdown
- **provide_materials** - vrací dostupné materiály v databázi ve formě pro dropdown
- **provide_profiles** - opět vrací data ve formě pro dropdown; vybírá skupiny sestav, které splňují, že zadané parametry jsou určeny pro daný materiál a typ plasmy, dále že tloušťka zpracovávaného materiálu leží v rozmezí tlouštěk skupiny sestav (sestavy se dají rozdělit do skupin, kde používají stejné plyny a mají stejný elektrický proud, ale jsou pro různé tloušťky materiálů). Id skupiny sestav je v dropdownu uloženo jako id první nalezené sestavy.
- **find_plasma_profile** - na základě id skupiny sestav a zadané tloušťky materiálu vybírá sestavu ze skupiny, která je na tuto tloušťku dimenzována. V případě nalezení soustavy rovnou volá funkci *select_profile_params*. Pokud taková sestava v databázi není, vrátí sestavy s nejbližší vyšší a nižší tloušťkou (uživatel je poté vyzván k výběru).
- **select_profile_params** - na základě id sestavy vrátí všechny její parametry
- **update_table** - aktualizuje v databázi hodnoty proudové sestavy, které byly změněny ve vizualizaci

5.4.3 Zprovoznění komponentů

V Linuxu je nutno nainstalovat Python 3 a MariaDB server, na kterém se také musí vytvořit uživatel s dostatečnými přístupovými právy, přes kterého se k databázi připojuje hlavní Python program. Dále je potřeba vytvořit databázi a vyplnit ji hodnotami. Jak bylo naznačeno v předchozí kapitole, exOS umožňuje automatický přesun souborů z projektu v AS do Linuxu a vykonání příkazů. To se provádí zadáním požadovaných příkazů a souborů k přesunu do souboru .exospackage v XML formátu (v konfiguraci je možné toto nastavení zobrazit i jako ostatní konfigurace AS, ale tímto způsobem do něj nelze zapisovat).

Při implementaci byla nalezena chyba v exOS. Příkazy se nevykonávaly v pořadí, v jakém byly do souboru zapsány, na pořadí ale u některých příkazů velmi záleží (např. není možné vytvořit databázi předtím, než je nainstalován databázový server). To bylo vyřešeno pomocí bash skriptu, který se spustí a příkazy v něm budou vykonány ve správném pořadí vždy. Jsou v něm použity příkazy jako *apt update, apt upgrade, apt install* a *wget*, které vyžadují připojení k internetu. Bylo by možné

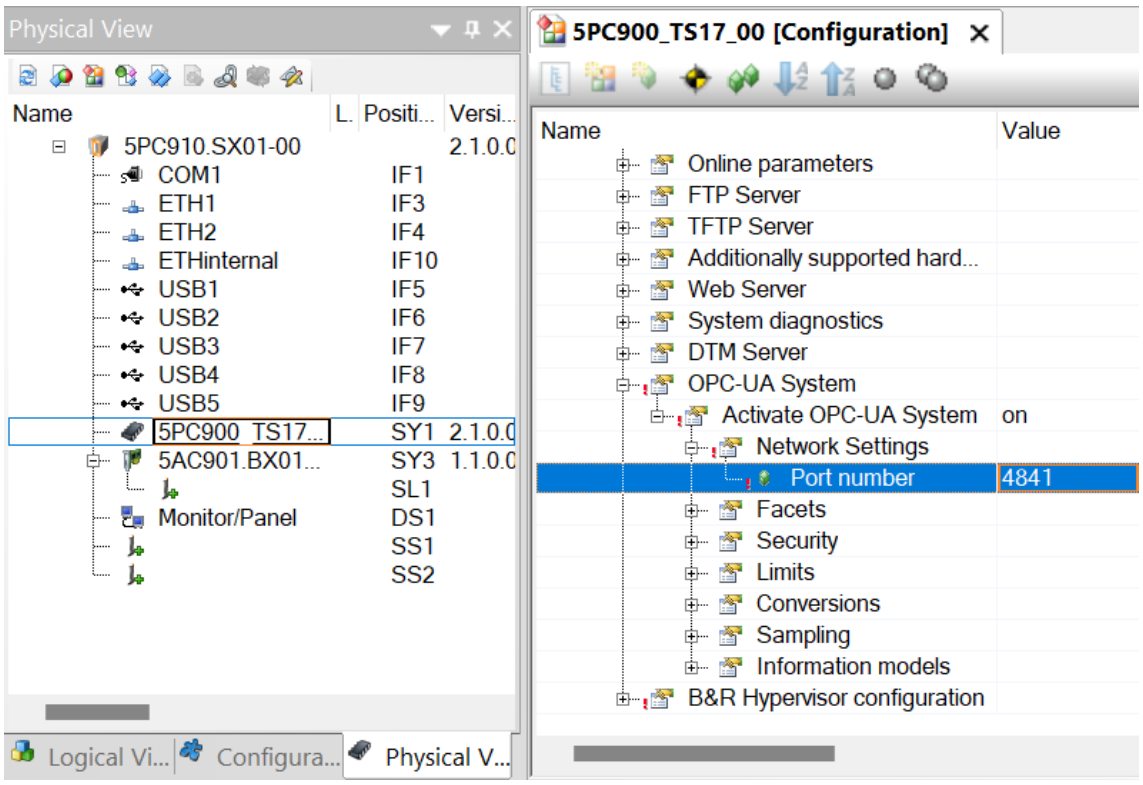
všechny potřebné balíčky předem stáhnout, vložit do složky projektu a při instalaci přesunout do Linuxu, aby při instalaci připojení k internetu nebylo vyžadováno, velikost projektu by se tím ale velmi zvětšila. Exospackage také nepodporuje přesun složek, ale pouze souborů, což je vcelku nepraktické, protože by do něj bylo potřeba vepsat velmi velký počet souborů.

5.5 Vizualizace

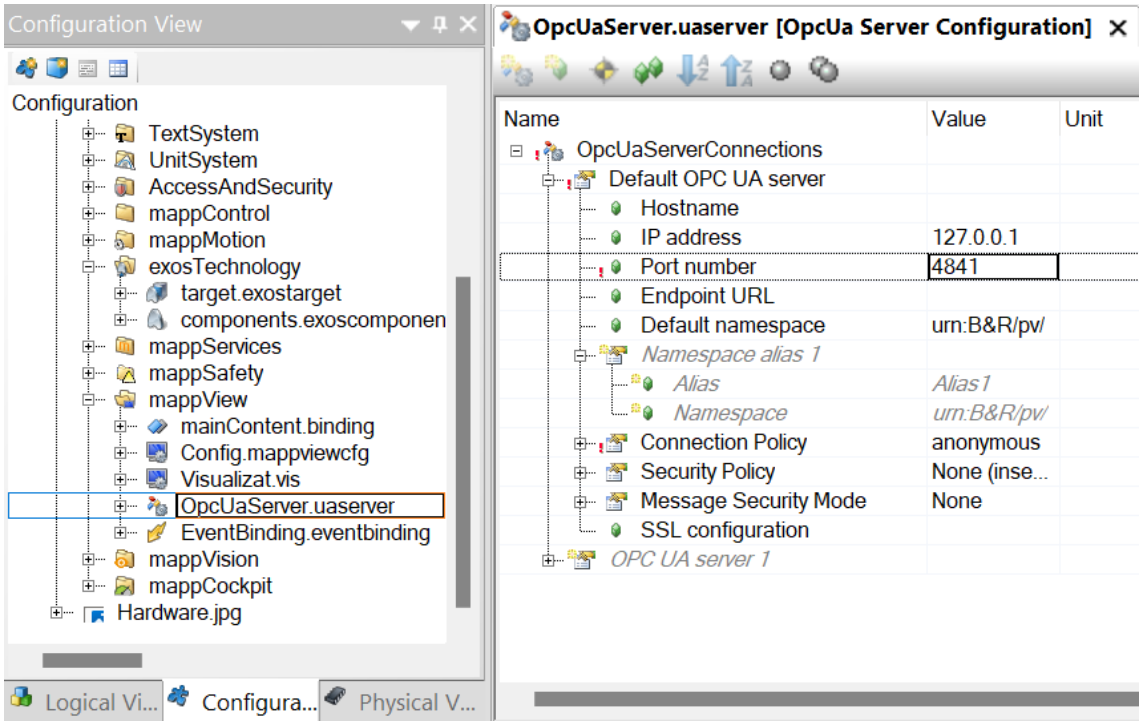
Protože cílem práce byla hlavně demonstrace využití exOS komunikace a implementace databáze, nebylo řešeno ovládání plasmové řezačky, které již na skutečné aplikaci funguje. Co se týče AR, tak kromě komunikace exOS byla vytvořena vizualizace pomocí mappView a její obslužný program. Nastavení plasmové řezačky je ukládáno do globální struktury, která bude předávána programu ovládajícímu řezačku.

5.5.1 Konfigurace serveru

Server exOS používá pevně daný port 4840. Při použití simulaci ve WSL je tento port na IP adrese localhost (127.0.0.1), na stejném portu běží ze základu ale i ARsim. Port ARsim tedy musí být změněn v konfiguraci procesoru (obr. 31). MappView má ze základu také nastaven port na 4840, tudíž do konfigurace musí být přidán prvek *OPC UA Remote server configuration* (obr. 32), do kterého musí být zadaná metoda autentizace a stejný port, jaký byl nastaven v konfiguraci procesoru.



Obr. 31: Konfigurace OPC-UA serveru

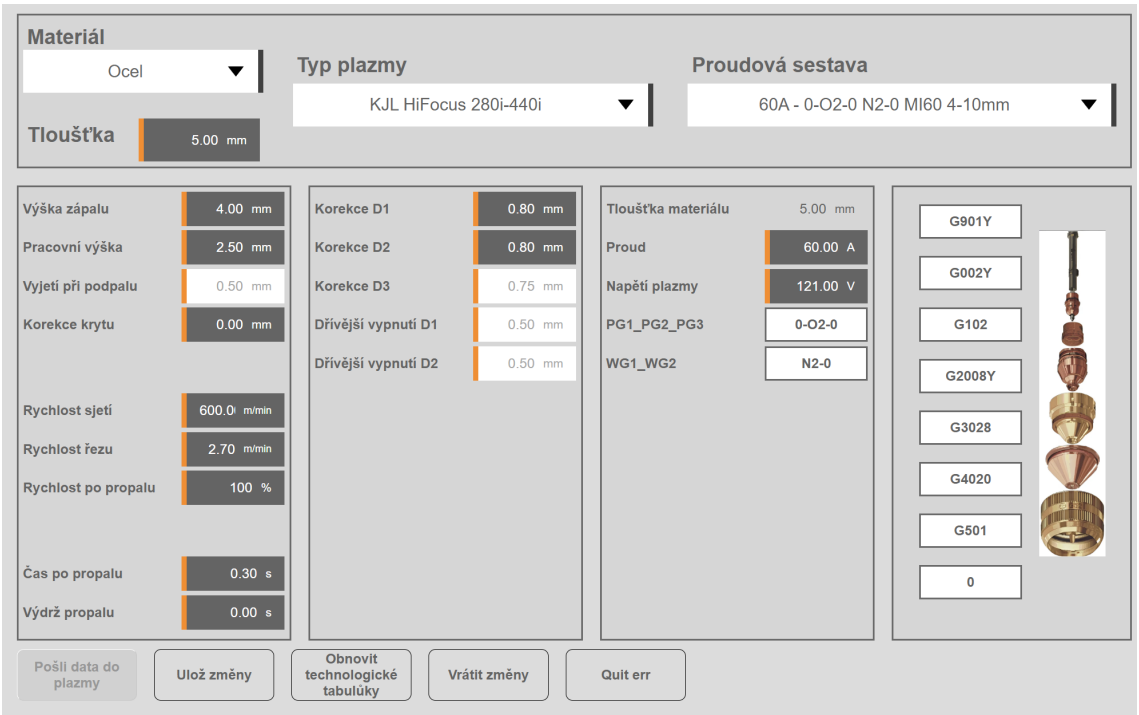


Obr. 32: Konfigurace mappView OPC-UA připojení

5.5.2 mappView

Vizualizace byla zpracována v mappView, což je nástroj pro vytváření HMI aplikací založený na HTML5, CSS3 a JavaScriptu. [33]

Stránka vizualizace (obr. 33) obsahuje několik výběrových dropdownů, které dostávají data přímo z Linuxu přes exOS a vstup pro zadání tloušťky zpracovávaného materiálu. Dále jsou k dispozici požadované údaje nastavení plasmového profilu, které se dají měnit zadáním nové hodnoty. Vstupy jejichž hodnotu změnil uživatel mají pro přehlednost jinou barvu. Tyto změny se poté dají vrátit, poslat přímo do programu ovládání plasmy anebo i uložit do databáze pomocí příslušných tlačítek (která vyvolají potvrzovací systémový dialog). Výstupy zobrazující tloušťku materiálu, pro který je sestava nastavena, použité plyny a mechanické díly, změnit z vizualizace nelze. Aby se zabránilo přehlcení programu Linuxu a databáze, tak pokud se aktuálně čeká na příjem dat do AR, jsou vstupy uživateli dočasně znepřístupněny.



Obr. 33: Hlavní stránka vizualizace

6 ZHODNOCENÍ A DISKUZE

Celý projekt byl úspěšně otestován v simulaci pomocí ARsim a WSL a je připraven k implementaci ve skutečné aplikaci. Rychlost přenášení dat z relační databáze do AR je oproti původnímu řešení s csv tabulkami mnohonásobně rychlejší. Test proběhl na počítači s výkonným procesorem AMD Ryzen 9. Dá se ale předpokládat, že při nasazení nativního Linuxu by bylo dosaženo ještě lepších výsledků (co se týče výkonu), protože WSL nedosahuje výkonnosti nativního systému (zvlášť na operačním systému Windows 11 Home, který nepodporuje Hyper-V). Výběrové dotazy trvaly řádově milisekundy. V simulačním prostředí trvala nejdéle komunikace mezi Linuxem a AR, od několika stovek milisekund až po nižší jednotky sekund. Na průmyslovém počítači by komunikace díky využití hypervisoru a sdílené paměti trvala podle dokumentace mikrosekundy až milisekundy. Při vkládání dat z csv tabulek do databáze se celkový objem dat mírně zmenšil, protože nebyly přidány neplatné proudové sestavy.

Databáze a její struktura by mohla mít vyšší úroveň normalizace, neřeší se zde například uložení jednotlivých mechanických dílů a druhů plynů do samostatných tabulek spojených s tabulkou proudových profilů. Zlepšením by mohlo také být seskupení proudových profilů podle elektrického proudu a použitých plynů již v databázi (navržené řešení vytváří skupiny pomocí výběrových dotazů). Struktura databáze by se pak ale začala značně lišit od původního řešení.

Při používání produktu exOS bylo nalezeno několik možných vylepšení. Od vydání produktu již WSL začlo podporovat SystemD, na základě tohoto faktu by bylo vhodné upravit připravené obrazy pro WSL. Příkazy zadané v .exospkg se ne vždy vykonávaly v pořadí, ve kterém byly zapsány, což je neočekávané a v určitých případech nechtěné chování (tento problém byl ale v práci vyřešen, jak je popsáno v kapitole 5.4.3). Vhodná by také byla implementace přesunu celých složek z projektu AS do Linuxu. ExOS také nepodporuje datový typ WSTRING, což znemožňuje například přesun znaků s diakritikou.

B&R Linux 10 je založený na Debianu 10, kterému ale již 30. 6. 2024 končí dlouhodobá podpora (což neznamená "end of life", který nastane až v roce 2029). [34] Z praktického hlediska to znamená obtížnější instalaci některých knihoven/modulů. Například u použitého serveru MariaDB musela být instalována starší verze a navíc provedena aktualizace konektoru jazyka C (na kterém je závislý konektor použitého Pythonu), oproti instalaci pomocí jednoho příkazu na nejnovějším Debianu 12. Některé nové nástroje by již nemuselo být možné na starším systému Debian 10 použít.

7 ZÁVĚR

Práce se zabývala možnostmi a implementací propojení RTOS s GPOS. Cílem byl průzkum výhod takového propojení a následná demonstrace na modelové aplikaci pomocí produktu exOS od firmy B&R.

První část práce se zaměřila na rozbor operačních systémů, jejich historii, typy a klíčové vlastnosti, zejména procesy a jejich plánování. Následně byla představena problematika kooperace RTOS a GPOS a popsány vybrané produkty dostupné na trhu, které tuto kooperaci umožňují.

Druhá část se soustředila na produkt exOS, byly popsány jeho vlastnosti a kroky potřebné pro jeho zprovoznění. Poté byla implementována databáze běžící na operačním systému Linux, který pomocí exOS komunikuje s operačním systémem Automation Runtime, jenž v reálné aplikaci řídí plazmovou řezačku. Bylo popsáno vytvořené programové vybavení aplikace jak na straně RTOS tak GPOS.

Vytvořená aplikace byla úspěšně otestována v simulačním prostředí a je připravena k využití ve skutečném průmyslovém projektu (buď přímo nebo jako modelové řešení). Byly diskutovány dosažené výsledky demonstrující zlepšení rychlosti přenosu dat oproti původnímu řešení a navržena možná vylepšení týkající se hlavně samotné databáze. Také byly zvýrazněny některé aspekty exOS, které by mohly být zlepšeny s cílem zjednodušení vývoje aplikací pro exOS, zlepšení programátorského komfortu a potenciálně univerzálnějšího využití.

SEZNAM POUŽITÉ LITERATURY

[1] SROVNAL, V. *Operační systémy pro řízení v reálném čase*. Ostrava: Vysoká škola báňská - Technická univerzita, 2003. ISBN 80-248-0503-0.

[2] FILŠER, J. *Principy operačních systémů I*. Ústí nad Labem: Univerzita J.E. Purkyně v Ústí nad Labem, 2016. ISBN 80-7044-505-X.

[3] KLIMEŠ, C. a BURIÁNOVÁ, E. *Základní pojmy z operačních systémů*. Ostrava: Ostravská univerzita, 2003. ISBN 80-7042-862-7.

[4] TANENBAUM, A. S. *Modern operating systems*. 3. vyd. Prentice Hall, leden 2008.

[5] JIHOUN, H. *History of OS* [https://github.com/chococigar/cup-of-cs/blob/main/img/history_of_os.md]. GitHub, leden 2022 [cit. 2024-04-5].

[6] ARM, J. *Detekce anomálií běhu RTOS aplikace [online]*. 2020 [cit. 2024-05-10]. Disertační práce. Vysoké učení technické v BrněBrno. Dostupné z: https://theses.cz/id/hbc2t5/.

[7] WIENAND, I. *Elements of a process*. 2013. Dostupné z: https://www.cs.swarthmore.edu/~kwebb/cs31/s15/bucs/elements_of_a_process.html.

[8] *Automation Help: Real-time operating system [Elektronická dokumentace]*. Eggelsberg, Austria. B&R Industrial Automation, 2023.

[9] *Embedded Hypervisor | Ultimate Guides | BlackBerry QNX*. [cit. 2024-05-10]. Dostupné z: https://blackberry.qnx.com/en/ultimate-guides/embedded-hypervisor.

[10] KATZENBEISSER, S., WEIPPL, E., CAMP, L. J., VOLKAMER, M., REITER, M. et al. *Trust and trustworthy computing*. Springer, červen 2012.

[11] MAIN, C. a TENASYS. *Allowing for GPOS and RTOS: The unique virtualization needs of mission-critical embedded systems*. Září 2009 [cit. 2024-03-10]. Dostupné z: https://militaryembedded.com/comms/communications/allowing-gpos-rtos-unique-needs-mission-critical-embedded-systems.

[12] GREEN HILLS SOFTWARE. *Wind River Helix Virtualization Platform Datasheet*. Březen 2023 [cit. 2024-04-5]. Dostupné z: https://www.windriver.com/resource/helix-datasheet.

[13] Green Hills Software. *Helix Virtualization Platform Product Overview*. Březen 2022 [cit. 2024-04-5]. Dostupné z: https://www.windriver.com/resource/helix-platform-product-overview.

[14] Green Hills Software. *INTEGRITY Multivisor*. Santa Barbara, us: [b.n.], 2018.

[15] Green Hills Software. *µ-visor - Safe and Secure Hypervisor*. Santa Barbara, us: [b.n.], 2024.

[16] BlackBerry QNX. *QNX Hypervisor*. 2021 [cit. 2024-04-5]. Dostupné z: https://www.blackberry.com/us/en/pdfviewer?file=/content/dam/qnx/products/qnx-hypervisor-brief-bb-20-0800-2-update-v6nv.pdf.

[17] BlackBerry QNX. *QNX Hypervisor for Safety*. 2022 [cit. 2024-04-5]. Dostupné z: https://www.blackberry.com/us/en/pdfviewer?file=%2Fcontent%2Fdam%2Fqnx%2Fproducts%2FQNX-Hypervisor-for-Safety.pdf.

[18] Acontis Technologies. *Type 1 and type 2 Real-time Hypervisor Solutions by Acontis*. [cit. 2024-04-5]. Dostupné z: https://www.acontis.com/en/realtime-hypervisor.html.

[19] Acontis Technologies. *LxWin Real-time Hypervisor: Windows + Real-time Linux*. [cit. 2024-04-5]. Dostupné z: https://www.acontis.com/en/windows-linux-realtime.html.

[20] Acontis Technologies. *VxWin Real-time Hypervisor: Windows + VxWorks*. [cit. 2024-04-5]. Dostupné z: https://www.acontis.com/en/vxworks-hypervisor.html.

[21] Acontis Technologies. *RTOSVisor: Type 1 Real-time Hypervisor*. [cit. 2024-04-5]. Dostupné z: https://www.acontis.com/en/acontis-hypervisor.html.

[22] BlackBerry QNX. *What is paravirtualization?* 2023 [cit. 2024-04-5]. Dostupné z: https://blackberry.qnx.com/en/ultimate-guides/automotive-hypervisor/paravirtualization.

[23] TenAsys. *INTime System Description*. 2024 [cit. 2024-04-5]. Dostupné z: https://support.tenasys.com/7-0/About_SystemDescription.

[24] TenAsys. *INtime SDK Help [Elektronická dokumentace]*. 2024 [cit. 2024-04-5]. Dostupné z: https://support.tenasys.com/7-1/webframe#toc.

[25] TenAsys. *INtime® Software Development Kit*. 2023 [cit. 2024-04-5].

[26] IntervalZero. *RTX64 ProductBrief*. Březen 2023 [cit. 2024-04-5]. Dostupné z: https://www.intervalzero.com/library/product_briefs/RTX64_44ProductBrief.pdf.

[27] IntervalZero. *INtime SDK Help [Elektronická dokumentace]*. 2024 [cit. 2024-04-5]. Dostupné z: https://help.intervalzero.com/product_help/RTX64_4/RTX64_4x_Help.htm.

[28] IntervalZero. *RTX64- The Ideal RTOS Platform for the IoT Era*. [cit. 2024-04-5]. Dostupné z: https://www.intervalzero.com/library/white_papers/RTX64-Ideal-RTOS-Platform-for-IoT.pdf.

[29] *Automation Help: exOS [Elektronická dokumentace]*. Eggelsberg, Austria. B&R Industrial Automation, 2023.

[30] *Operating systems and drivers (GPOS) [Elektronická dokumentace]*. Eggelsberg, Austria. B&R Industrial Automation, 2023.

[31] *System and Service Manager*. [cit. 2024-05-10]. Dostupné z: https://systemd.io/.

[32] Arkane Systems a Young, A. *GitHub - arkane-systems/genie: A quick way into a systemd "bottle"for WSL*. 2020 [cit. 2024-05-10]. Dostupné z: https://github.com/arkane-systems/genie.

[33] *Visualization [Elektronická dokumentace]*. Eggelsberg, Austria. B&R Industrial Automation, 2023.

[34] *Debian Wiki - Debian Long Term Support*. Dostupné z: https://wiki.debian.org/LTS.

SEZNAM ZKRATEK A SYMBOLŮ

GPOS operační systém obecného využití – General Purpose Operating System

RTOS operační systém reálného času – Real Time Operating System

CLI příkazová řádka – Command Line Interface

GUI uživatelské grafické rozhraní – Graphical User Interface

HAL hardwarová abstrakční vrstva – Hardware Abstraction Layer

IPC meziprocesová komunikace – Inter-Process Communication

FIFO první dovnitř, první ven – First In First Out

FCFS první přišel, první obsloužen – First Come First Served

SJF nejkratší práce první – Shortest Job First

PLC programovatelný logický automat – Programable Logic Controller

MLQ plánování několikaúrovňových front – Multilevel Queue Scheduling

AMP asymetrický multiprocesing – Asymmetric Multiprocessing

SMP symetrický multiprocesing – Symmetric Multiprocessing

RAM paměť s náhodným přístupem – Random-acess Memory

IEC mezinárodní elektrotechnická komise – International Electrotechnical Commission

WSL windowsový podsystém pro Linux – Windows Subsystem for Linux

VS Visual Studio

AS Automation Studio

AR Automation Runtime

DMR dataset message router

TCP Transmission Control Protocol

SWIG Simplified Wrapper and Interface Generator

HMI rozhraní člověk-stroj – Human-Machine Interface

SEZNAM OBRÁZKŮ

A SEZNAM PŘÍLOH

Elektronické přílohy:

Logical (adresář obsahující programy a softwarové komponenty projektu)

Physical (adresář obsahují hardwarové konfigurace projektu)