VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

ÚVOD DO MAINFRAME

INTRODUCTION TO MAINFRAME

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **TOMÁŠ NAJMAN**
AUTHOR

VEDOUCÍ PRÁCE **Ing. BOHUSLAV KŘENA, Ph.D.**
SUPERVISOR

BRNO 2025

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

164033

Ústav: Ústav inteligentních systémů (UITS)
Student: **Najman Tomáš**
Program: Informační technologie
Název: **Úvod do Mainframe**
Kategorie: Počítačová architektura
Akademický rok: 2024/25

Zadání:

1. Seznamte se s architekturou mainframe a její historií.
2. Registrujte se na výukovou platformu IBM Z Xplore a získejte v ní úroveň *Advanced*.
3. Ve spolupráci s odborným vedoucím ze společnosti Kyndryl navrhněte sadu příkladů pro demonstraci administrace systémů s využitím tradičního přístupu (zejména jazyk pro řízení úloh JCL, programovací jazyk Cobol a skriptovací jazyk REXX) a moderního přístupu (zejména programovací jazyk Python a vývojové prostředí Visual studio).
4. Tuto sadu příkladů implementujte, pečlivě zdokumentujte a důkladně otestujte, a to včetně výkonnostních parametrů.
5. Zhodnoťte dosažené výsledky a diskutujte možnosti jejich využití a dalšího rozvoje.

Literatura:

1. IBM. *Mainframe concepts*. Copyright 2005, 2008. Citováno: 2024-10-15. Dostupné na URL: https://www.ibm.com/docs/en/zosbasics/com.ibm.zos.zmainframe/zmainframe_book.pdf.
2. IBM. *Introduction to the New Mainframe*. Third Edition. March 2011. Citováno: 2024-10-15. Dostupné na URL: https://www.redbooks.ibm.com/redbooks/pdfs/sg246366.pdf.
3. IBM. *IBM Open Enterprise SDK for Python 3.12*. Last updated: 2023-11-14. Citováno: 2024-10-15. Dostupné na URL: https://www.ibm.com/docs/en/python-zos/3.12?topic=pdf-version.
4. IBM. *TSO/E REXX User's Guide*. Last updated: 2024-01-20. Citováno: 2024-10-15. Dostupné na URL: https://www.ibm.com/docs/en/SSLTBW_3.1.0/pdf/ikjc300_v3r1.pdf.
5. IBM. *Enterprise COBOL for z/OS 6.4*. Fourth edition. 28 June 2024 update. Citováno: 2024-10-15. Dostupné na URL: https://www.ibm.com/docs/en/SS6SG3_6.4.0/pdf/pgmvs.pdf.
6. IBM. *IBM Z Xplore*. Citováno: 2024-10-25. Dostupné na URL: https://ibmzxplore.influitive.com/.

Při obhajobě semestrální části projektu je požadováno:
První tři body zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Křena Bohuslav, Ing., Ph.D.**
Konzultant: Ing. Ondřej Zmund
Vedoucí ústavu: Kočí Radek, Ing., Ph.D.
Datum zadání: 1.11.2024
Termín pro odevzdání: 14.5.2025
Datum schválení: 31.10.2024

Abstrakt

Tato práce představuje komplexní přehled historického vývoje, současného využití i budoucích trendů Mainframe, pro zájemce o toto technologické odvětví. Poskytuje přehled hlavních aspektů mainframových systémů, se zaměřením na IBM Z Series a související technologie, včetně detailního popisu architektury, operačních systémů (zejména z/OS a z/VM) a specifických aplikačních metod, které zajišťují jejich bezkonkurenční spolehlivost, škálovatelnost a bezpečnost. Dále je věnována pozornost tradičním programovacím jazykům používaným na mainframech – COBOL, JCL a Rexx, přičemž je také představen způsob, jakým se na IBM Z Series uplatňuje Python. Součástí práce jsou také demonstrační úlohy, které slouží jako praktický nástroj pro zájemce o hlubší seznámení s funkcionalitami a možnostmi mainframových systémů.

Abstract

This thesis presents a comprehensive overview of the historical development, current use and future trends of the Mainframe, for those interested in this technology sector. It provides an overview of the major aspects of mainframe systems, with a focus on the IBM Z Series and related technologies, including detailed descriptions of the architecture, operating systems (especially z/OS and z/VM), and specific application methods that provide their unmatched reliability, scalability, and security. In addition, attention is given to the traditional programming languages used on mainframes – COBOL, JCL and Rexx, while the way in which Python is applied to the IBM Z Series is also introduced. The work also includes demonstration tasks that serve as a practical tool for those interested in gaining a deeper understanding of the functionality and capabilities of mainframe systems.

Klíčová slova

Mainframe, IBM Z, z/OS, COBOL, JCL, Rexx, Python, virtualizace, bezpečnost, demonstrační úlohy, transakční zpracování, historie počítačů

Keywords

Mainframe, IBM Z, z/OS, COBOL, JCL, Rexx, Python, virtualization, security, demonstration tasks, transaction processing, computer history

Citace

NAJMAN, Tomáš. *Úvod do Mainframe*. Brno, 2025. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Bohuslav Křena, Ph.D.

Úvod do Mainframe

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Bohuslava Křeny, Ph.D. Další informace mi poskytli Ing. Ondřej Zmund a Petr Drahoš Filip ze společnosti Kyndryl. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Tomáš Najman
11. května 2025

Poděkování

Rád bych poděkoval vedoucímu této bakalářské práce, panu Ing. Bohuslavu Křenovi, Ph.D., za cenné odborné rady a návrhy pro napsání této práce. Dále děkuji konzultantům ze společnosti Kyndryl, panu Ing. Ondřeji Zmundovi, za poskytnutí odborných materiálů a následnou kontrolu práce, a také panu Petru Drahošovi Filipovi, za celkovou pomoc ohledně implementační části. V neposlední řadě děkuji také rodině za neustálou podporu během celého studia.

Obsah

Kapitola 1

Úvod

Mainframy představují zcela unikátní a úzkou kategorii výpočetní techniky. Mohou být přirovnány k přísloví „Tichá voda břehy mele“, neboť navzdory své nenápadnosti jsou již po několik dekád klíčovým pilířem provozu kritických systémů finančních institucí, státní správy či zdravotnictví po celém světě. Přesto zůstávají pro veřejnost, akademickou obec i většinu běžných vývojářů téměř neviditelné.

O to víc mě tato „neviditelnost“ zaujala. V době, kdy se v akademickém prostředí často hovoří o moderních cloudových řešeních, mikroservisní architektuře či kontejnerizaci, je téma mainframů ve výuce prakticky opomíjené. Přitom se s nimi lidé nepřímo setkávají téměř každý den – ať už při výběru z bankomatu, placení kartou nebo při komunikaci s úřady. Z toho důvodu jsem se rozhodl zpracovat tohle téma – jako způsob, jak upozornit na jejich význam, rozšířit povědomí o jejich schopnostech a pokusit se přiblížit jejich architekturu i fungování nové generaci IT odborníků a zájemců o toto odvětví.

Osobně mě zajímalo, jak byly a jsou tyto systémy navrženy pro udržení stabilního provozu v řádu let, zvládající zpracování obrovského množství transakcí denně, které by běžné servery položily. Téma mainframů tak pro mě představuje kombinaci historického vývoje, inženýrské preciznosti a technologické dlouhověkosti, což považuji za výjimečné i v rámci rychle se měnícího IT světa.

Cílem práce je nejen seznámení se s odlišným systémem a architekturou. Ale také hlubší porozumění tomu, jak tyto systémy fungují pod pokličkou. Proto jsem se seznámil s programovacím jazykem COBOL a skriptovacími jazyky JCL a REXX. Tyto jazyky jsou s mainframy tradičně spojeny, a v rámci této práce jsem implementoval základní úlohy, které fungují v mainframovém prostředí. Tím jsem získal nejen teoretický vhled, ale i praktickou zkušenost. S technologickými prvky a kontrolou implementace mi pomáhali konzultanti ze společnosti Kyndryl, jež tuto práci organizovala. V rámci této práce jsem také dosáhl až na nejvyšší úroveň – **All Star**[1] na platformě IBM Z Xplore. Jde o online certifikát poskytovaný společností IBM za splnění potřebných kritérií na zmíněné platformě.

Obecné informace: Mainframe je navržený pro provoz rozsáhlých aplikací v prostředí, kde je vyžadována maximální spolehlivost a efektivita. Na rozdíl od standardních serverů je mainframe optimalizován pro vysoký výkon při paralelním zpracování úloh, což jej činí vhodným pro organizace s komplexními IT potřebami, v tomto případě mluvíme o opravdu vysoké „propustnosti“. Tato vlastnost je klíčová pro kritické sektory, kde si zprostředko-

[1] Ověření dosažené úrovně je dostupné na https://www.credly.com/badges/3de4a77a-8898-4997-84bb-37200315df53/public_url

vatel těchto služeb nemůže dovolit výpadky nebo zpoždění. Například systémy mainframů v bankách zajišťují každodenní provoz platebních služeb atd. *Díky mainframům můžeme každodenně používat kreditní karty v obchodech.* Dalším rysem mainframů je jejich schopnost udržet provoz nepřetržitě po dlouhá období, často měřená v letech, bez významných výpadků nebo selhání, což je klíčové zejména pro organizace, které vyžadují neustálý přístup k datům. IBM z16 poskytuje až 99,999 999 9% dostupnost, což v časovém kontextu znamená 31,56 milisekund neplánovaných výpadků ročně. Model IBM z13 byl testován na odolnost, kdy se zjistilo, že zvládne vydržet i zemětřesení o síle 8 Richterovy škály. Jejich schopnost zvládat miliardy transakcí denně s minimální latencí podtrhuje jejich stabilitu v prostředích, kde jsou výpadky nepřípustné. Široká škála využití a robustnost činí mainframy stále relevantními i v současnosti, kdy jsou často integrovány s moderními technologiemi, jako je cloud computing nebo analýza velkých dat [4, 28, 44, 85].

Historie mainframů se začíná psát už od poloviny minulého století, kdy byly vyvinuty první sálové počítače. Tyto počítače, jako například IBM 701 nebo IBM S/360, položily základy moderních mainframových počítačů a definovaly klíčové koncepty, které se využívají do současnosti.

Z hlediska konsolidace, díky modularitě hardwaru a softwaru mohou organizace rozšiřovat výpočetní kapacitu mainframů podle potřeby, což umožňuje jejich dlouhodobé využití bez nutnosti časté výměny systémů.

Mainframy poskytují pokročilé zabezpečovací mechanismy, včetně šifrování dat a ochrany proti kybernetickým útokům. Tyto funkce jsou obzvlášť důležité pro bankovnictví, zdravotnictví a státní správu.

Na rozdíl od mnoha jiných platforem mainframy umožňují provoz starších aplikací spolu s moderními technologiemi, což šetří náklady na migraci, zejména použití kódu, který díky zpětné kompatibilitě funguje i na nejnovějších verzích IBM z Series.

Avšak pořízení mainframového systému představuje značnou investici, která může být pro menší organizace obtížně dosažitelná. Firmy nabízejí i levnější varianty mainframů, ty se ale výkonnostně velmi liší od IBM Z.

Správa a údržba mainframů vyžaduje odborné znalosti, zejména znalost dnes už skoro nepoužívaných jazyků, jako je JCL, COBOL atd. Tohle může být problémem, jelikož se počet kvalifikovaných specialistů na mainframy snižuje.

Mainframy často vyžadují specializované datové centrum s odpovídající infrastrukturou, jako je chlazení a záložní zdroje energie [75].

Tato práce je strukturována do několika kapitol. Kapitola 2 popisuje historický vývoj mainframů. Kapitola 3 se zaměřuje na současnou architekturu IBM Z mainframů, operační systémy, využití a uvádí konkurenční výrobce a jejich systémy. Kapitola 4 se zaměřuje na uvedení do programovacích jazyků a nástrojů pro vývoj na mainframech. Kapitola 5 obsahuje přehled implementovaných úloh se slovním popisem a zjednodušenými ukázkami. V závěrečné kapitole 6 je popsáno, co tato práce pokryla a rozebrala s návrhy na rozšíření.

Kapitola 2

Historie mainframových systémů

Historie mainframů představuje období intenzivního experimentování a technologických inovací, které formovaly moderní výpočetní techniku. Od prvních počítačů určených k řešení specifických, většinou matematických, problémů až po univerzální platformy schopné zvládat široké spektrum úloh, mainframy prošly dramatickými proměnami. Tato kapitola se zaměří na klíčové momenty ve vývoji mainframů, jejich vliv na průmysl a společnost a na technologické inovace, které zásadně ovlivnily podobu dnešních mainframů od technologické společnosti IBM a jejich IBM Z Series.

2.1 Počátky výpočetní techniky

Přechod od mechanických počítacích strojů k elektronickým počítačům znamenal zásadní změnu, která otevřela cestu k vývoji univerzálních a výkonných mainframů. Vývoj elektronických počítačů sahá do 40. let 20. století, kdy byly vyvinuty první elektronické počítače.

Zajímavou kapitolou v této době byl vývoj specializovaných počítačů, jako byl britský Colossus. Tento stroj, nasazený během druhé světové války, byl vytvořen pro dešifrování německých zpráv kódovaných systémem Enigma. Colossus byl jedním z prvních prakticky využitých elektronických počítačů a ukázal, že výpočetní technika může mít zásadní vliv nejen na vědecké, ale i vojenské aplikace. Přestože byl Colossus tajným projektem, jeho koncepty ovlivnily další vývoj elektronických systémů.

Významným milníkem v této době byl ENIAC (Electronic Numerical Integrator and Computer), představený v roce 1946. Tento stroj, navržený primárně pro vojenské účely, využíval 18 000 elektronek a vážil přibližně 30 tun. ENIAC dokázal provádět složité matematické výpočty, například výpočty balistických drah, a zrychlil procesy, které by jinak trvaly týdny či měsíce [8].

Jeho konstrukce byla však extrémně náročná na údržbu – elektronky produkovaly velké teplo, což vedlo k častým poruchám. Například jeden den provozu ENIACu znamenal průměrnou potřebu vyměnit několik elektronek. Navzdory tomu ukázal ENIAC potenciál elektronických počítačů a stal se inspirací pro další vývoj.

Jedním z hlavních negativ z pohledu dnešní doby ENIACu byla absence programu uloženého v paměti. Tento nedostatek však překonal jeho nástupce EDVAC (Electronic Discrete Variable Automatic Computer), který přinesl revoluční koncept uloženého programu. Tuto myšlenku zpopularizoval John von Neumann ve své publikaci „First Draft of a Report on the EDVAC“, která položila základy moderní výpočetní architektury. Uložený program

umožnil zpracovávat různé úlohy bez nutnosti fyzického přeprogramování stroje, což značně zjednodušilo práci s počítači a přispělo k jejich univerzálnímu využití [1].

Dalším klíčovým krokem byl UNIVAC I (Universal Automatic Computer), který se stal prvním komerčně dostupným počítačem v roce 1951. UNIVAC byl navržen pro praktické aplikace, jako bylo zpracování dat pro americký sčítací úřad nebo analýza obchodních trendů. Tento počítač, vyvinutý J. Presperem Eckertem a Johnem Mauchlym, měl kapacitu uložit až 1 000 slov ve své magnetické paměti a jeho rychlost umožňovala zpracovávat až 10 000 operací za sekundu. UNIVAC se stal symbolem první generace počítačů a ukázal, že výpočetní technika může být široce využitelná v komerčním sektoru.

K rozvoji výpočetní techniky přispělo také akademické prostředí. Významným projektem byl IAS počítač (Institute for Advanced Study Computer) Johna von Neumanna, který definoval základní principy moderních počítačů. Von Neumannova architektura, založená na třech klíčových komponentech – procesoru, paměti a vstupně-výstupních zařízeních – umožnila rozvoj univerzálních počítačů, které byly schopny vykonávat širokou škálu úkolů.

Technologický pokrok v tomto období byl doprovázen významnými investicemi do výzkumu a vývoje, zejména ze strany vládních institucí. Americká vláda hrála klíčovou roli v podpoře projektů, jako byly ENIAC a UNIVAC, a položila tak základy pro vznik celého odvětví výpočetní techniky [1].

2.2 Elektronkové mainframy

Vznik mainframů jako samostatné kategorie počítačů je spojen s obdobím 50. a 60. let 20. století, kdy se na trhu objevily první komerční mainframy, které zásadně změnily způsob zpracování dat ve velkých organizacích. Společnost IBM v 50. letech vyvinula celou řadu mainframů řady IBM 700, například modely 701, 704, 709 – určeny pro vědecké účely. 702, 705, 7010 – určené pro komerční použití a zpracování dat. Tyto řady nebyly vzájemně kompatibilní – lišily se sadami instrukcí podle zaměření. Např. IBM 704 (1954) zavedl aritmetiku s plovoucí řádovou čárkou a magnetickou jádrovou paměť a byl prvním masově vyráběným strojem s hardwarem pro výpočty s plovoucí řádovou čárkou. Tyto stroje pracovaly bez operačního systému a obsluha počítače musela ručně řídit dávky úloh. Ke konci 50. let se začaly objevovat první jednoduché dávkové operační systémy vyvinuté uživateli, např. GM-NAA I/O (1956) pro IBM 704, které automatizovaly sekvenční spouštění úloh, aby drahý mainframe nezahálel [66].

IBM 701

IBM 701, představený v roce 1952, byl prvním komerčně vyráběným, vědeckým počítačem od IBM. Hlavní důvod vzniku byl začátek Korejské války, během vývoje se mu přezdívalo „Defense Calculator“. IBM 701 umožnil rychlé zpracování dat, což bylo klíčové pro vojenské a vědecké aplikace, například simulace proudění vzduchu, analýzu trajektorií raket nebo výpočty spojené s návrhem jaderných zbraní. Vědecké instituce a armádní výzkumné týmy mohly díky IBM 701 provádět výpočty, které byly dříve nemyslitelné.

Jeden z hlavních přínosů IBM 701 spočíval v jeho schopnosti provádět přibližně 16 000 sčítacích operací za sekundu[1]. Počítač využíval magnetickou bubnovou paměť, která nabízela relativně rychlý přístup k uloženým datům, a děrné štítky pro vstup a výstup dat, což

[1] Jedna sčítací operace na IBM 701 trvala přibližně 60 mikrosekund.

umožnilo automatizaci procesů. Ve své době bylo vyrobeno pouze 19 kusů tohoto modelu, což odráží jeho využití především ve specializovaných oblastech. Úspěch tohoto počítače otevřel cestu dalším modelům IBM a položil základy pro moderní mainframové systémy [18, 62].

IBM 704

IBM 704, uvedený na trh v roce 1954, byl dalším významným krokem v evoluci mainframů. Tento model byl prvním komerčním počítačem s hardwarovou podporou pro aritmetické operace s plovoucí desetinnou čárkou. Tato vlastnost umožnila mnohem přesnější a rychlejší vědecké výpočty, což bylo klíčové pro inženýrské, vědecké a akademické aplikace. IBM 704 byl z hlediska architektury a provedení výrazně lepší než jeho předchůdce IBM 701. Sice používal také elektronky. Nicméně zvětšil velikost instrukcí z 18 bitů na 36 bitů, což odpovídalo velikosti „slova“ paměti. Mezi změny oproti modelu 701 patří použití feritové paměti (paměti s magnetickým jádrem), místo Williamsových elektronek, aritmetické instrukce s plovoucí desetinnou čárkou, 15bitové adresování a přidání tří indexových registrů. Pro podporu těchto nových funkcí byly instrukce rozšířeny tak, aby využívaly celé 36bitové slovo. Nová instrukční sada, která není kompatibilní s počítačem 701, se stala základem podtřídy „vědecké architektury“ počítačů IBM řady 700/7000. IBM 704 dokázal provést až 12 000 sčítání s plovoucí desetinnou čárkou za sekundu.

Pro IBM 704 byl vyvinut i programovací jazyk FORTRAN, který byl představen v roce 1957. FORTRAN byl navržen jako jazyk srozumitelný pro inženýry a vědce, kteří mohli psát kód v podobě matematických rovnic místo použití složitých strojových kódů. Tento jazyk přinesl revoluci tím, že výrazně zkrátil čas potřebný na vývoj softwaru oproti strojovému kódu.

Kromě numerické analýzy byl FORTRAN hojně využíván i při vývoji softwaru pro kosmický výzkum, což vedlo k zásadním pokrokům v oblasti raketové techniky a návrhu kosmických lodí. IBM 704 a FORTRAN byly použity při výpočtech trajektorií pro vesmírné mise, což demonstrovalo jejich praktický dopad.

IBM 704 byl tak nejen technologickým průlomem, ale také zásadním krokem k zefektivnění vývoje softwaru. Díky kombinaci inovativního hardwaru a podpory vyššího programovacího jazyka vytvořil základ pro moderní výpočetní techniku [46, 64].

2.3 Tranzistorové mainframy

Přelomovou technologickou inovací bylo u této generace počítačů použití tranzistorů místo elektronek. Tranzistory byly mnohem menší, spolehlivější a úspornější, což umožnilo zvýšit komplexitu i rychlost počítačů. Paměť nadále tvořila převážně magnetická jádra, ale kapacita rostla (řádově stovky kB).

IBM v roce 1958 uvedlo tranzistorový IBM 7090. Řada IBM 7000 byla rozdělena na modely pro vědecké použití, jako byl IBM 7090 a s ním částečně kompatibilní IBM 7040[2], a na modely pro komerční použití, jako IBM 7070 a IBM 7080. Výkonnostně i kapacitou paměti převyšovaly předchůdce, avšak stále s odlišnou architekturou. IBM 7094 měl např. 36bitovou architekturu zaměřenou na čísla s plovoucí řádovou čárkou, zatímco IBM 7070 měl pouze 10bitovou dekadickou architekturu pro zpracování byznys dat. Tato nekompatibilita začala být významnou brzdou.

[2]Tento model vyšel v roce 1961, po IBM 7090.

V druhé generaci se začínají objevovat první skutečné operační systémy. Například pro IBM modely 7090 a 7094 vznikl v roce 1960 systém IBSYS a další, které umožňovaly dávkové zpracování. V této době to znamenalo automatizované načítání a provádění posloupnosti úloh z pásky. IBSYS podporoval i jazyky jako FORTRAN nebo COBOL [66].

Následující výčet uvádí vybrané příklady významných mainframových systémů druhé generace, jak od IBM, tak od jiných výrobců. Mimo řadu IBM 7000:

- **IBM 1401 (1959)** – první model ze série 1400, byl extrémně úspěšný a využíván i spolu s mainframy. Šlo o „kompaktnější" modely oproti jiným řadám. Tento model používal znakovou architekturu – „binary-coded decimal" [61].
- **IBM 7030 Stretch (1961)** – experiment IBM superpočítače, přinesl koncept paralelního zpracování instrukcí (předběhl dobu, měl pipelining, predikci cache, ochranu paměti – leccos z toho se plně docenilo až o dekády později). Ačkoli Stretch nesplnil přehnané výkonnostní cíle, jeho inovace ovlivnily další vývoj (např. IBM System/360 implementoval jeho principy) [63].
- **Burroughs B5000 (1961)** – první model řady Burroughs Large Systems. Používal 48 bitovou architektu a byl optimalizovaný pro překlad programů napsaných v jazyce ALGOL 60. Veškerá programová data se ukládala na zásobníku a procesor prováděl operace nad ním. Zavedl hardwarovou podporu multitaskingu, jelikož existovaly instrukce pro rychlé přepínání procesů na úrovni hardware [57].
- **UNIVAC 1107 (1962)** – tranzistorový, 36bitový mainframe s podporou překladačů FORTRAN a operačním systémem EXEC I. Místo magnetických pamětí používal rychlejší alternativu – „thin-film" paměť. Prodalo se pouze 36 kusů [82].

2.4 Mainframy s integrovanými obvody

Přechod k integrovaným obvodům představoval zásadní milník ve vývoji výpočetní techniky a definoval nástup třetí generace mainframových systémů. Integrace více elektronických prvků do jednoho modulu zvýšila spolehlivost, výkon i výrobní efektivitu a zároveň umožnila výrazné zmenšení fyzických rozměrů zařízení. Prvním významným zástupcem této nové generace se stal IBM System/360, využívající technologii SLT (Solid Logic Technology).

IBM System/360

IBM System/360, představený v roce 1964, byl milníkem, který zásadním způsobem ovlivnil celý svět výpočetní techniky. Šlo o jednotnou rodinu mainframů pokrývající celou škálu výkonů a aplikací. System/360 přinesl jednotnou architekturu pro vědecké i komerční účely – všechny modely sdílely stejnou sadu instrukcí a datové formáty. Číslo „360" symbolizovalo „360 stupňů" – univerzálnost použití. Bylo to poprvé, kdy zákazník mohl převést programy mezi počítači různého výkonu bez přepisování – nižší modely mohly, byť s výkonnostními nebo paměťovými omezeními, spouštět software z vyšších a naopak. IBM toho dosáhlo pomocí mikroprogramování – vnitřní implementace instrukcí se mohla lišit, ale navenek byly modely kompatibilní. S/360 také umožnil emulaci starších IBM počítačů – např. model 360/65 uměl v mikrokódu emulovat IBM 7094 pro ochranu investic zákazníků.

System/360 je klasifikován jako mainframe třetí generace, protože používal už integrované obvody (přesněji technologii označenou IBM jako SLT – Solid Logic Technology). SLT

představovala hybridní technologii, která kombinovala více tranzistorů a pasivních součástek (rezistory, kondenzátory) v jednom malém modulu. Nešlo tedy ještě o plnohodnotný monolitický integrovaný obvod, jaký známe dnes, ale již šlo o výrazný posun od čistě diskrétních tranzistorů, které používaly předchozí generace.

Architektura System/360 zaváděla také řadu pokročilých konceptů, které byly v té době revoluční. S/360 definoval 32bitové slovo a 24bitovou adresu s 8bitovým bajtem jako základní adresovatelnou jednotkou a tímto zajistil větší adresový prostor. Přechod od dřívějších 36bitových či 48bitových slov k bajtově orientované architektuře byl vhodný i pro textová data. S/360 jako první zavádí hardwarovou podporu operačního systému – instrukční sadu doplnil režim supervisor vs. uživatelský, a ochranu paměti pomocí klíčů přiřazených blokům paměti. To zabránilo uživatelským programům narušit paměť OS nebo jiných běžících úloh.

Z technického hlediska byl velkou inovací integrovaný vstupně/výstupní subsystém s kanálovými procesory. Namísto, aby CPU čekal na pomalé operace tiskáren nebo disků, S/360 měl samostatné kanálové jednotky, které asynchronně prováděly I/O operace a dovolovaly paralelní běh výpočtů a vstupů/výstupů. Byly dva typy kanálů – bajtové multiplexové kanály, které sloužily pro propojení pomalejších zařízení, jako v té době byly čtečky děrných štítků nebo tiskárny. Druhý typ, rychlejší varianta, byly selektivní kanály určené pro čtení z disků nebo kazet. Oba kanály se připojovaly přes standardizované rozhraní, takže široká škála periférií mohla být sdílena napříč modely.

Ekonomický přínos System/360 byl rovněž zásadní. Jeho uvedení na trh vedlo k prudkému růstu poptávky po kvalifikovaných IT odbornících, kteří mohli pracovat s touto novou platformou. Standardizace architektury také otevřela nové příležitosti pro software vyvíjený třetími stranami. Vývojáři mohli efektivněji vytvářet aplikace, které nebyly omezeny na konkrétní hardware, čímž došlo k masivnímu růstu softwarového průmyslu. Firmy tak získaly možnost volit z větší nabídky řešení, což dále podpořilo inovace a konkurenci v IT sektoru.

System/360 měl rovněž kulturní dopad. Veřejnost si začala více uvědomovat význam počítačové techniky pro moderní společnost. Počítače se staly klíčovým prvkem v médiích i populární kultuře, což přispělo k šíření znalostí o technologiích mezi širší populací. Akademické instituce začaly na základě tohoto systému rozvíjet nové výzkumné programy zaměřené na počítačové vědy, což vedlo k rozvoji celé generace odborníků, kteří posunuli hranice technologického výzkumu. Tuto filozofii – umožnění zpětné kompatibility hardwaru, rozšíření systému po koupi menšího systému i kompatibilita na aplikační vrstvě – si IBM zachovalo do nynější generace IBM Z [6, 23, 66, 67].

Konkurence na trhu mainframů nebyla zanedbatelná, přesto IBM ovládla trh s počítači. Společnosti jako Univac, General Electric, RCA a Burroughs musely reagovat na revoluční IBM System/360. Vyvíjely vlastní systémy, které nabízely alternativy k dominantním modelům IBM.

Například systém Univac 1108 a později série 1100/2200 zůstaly 36bitové. Zavedly multiprocesový OS, kdy se použilo více CPU v jednom stroji. Společnost Burroughs implementovala a poskytovala „virtuální paměť“ založenou na segmentaci. Počítač B5000 byl vybaven informační tabulkou segmentů, která se nazývá Program Reference Table (PRT) sloužící k určení, zda se příslušný segment nachází v hlavní paměti, k udržování základní adresy a velikosti segmentu [76, 82].

IBM System/370

V roce 1970 IBM uvedlo nástupnickou řadu System/370 (S/370). Všechny modely této řady používaly už monolitické integrované obvody navržené IBM. První modely neměly virtuální paměť, ale konkurence donutila IBM rychle reagovat. V srpnu 1972 IBM oznámilo rozšíření S/370 o podporu virtuální paměti a vydalo nové operační systémy. Nejpoužívanější byla modernizovaná víceadresní verze OS MVS (Multiple Virtual Storage), v níž již každý proces běžel ve vlastním adresovém prostoru – to byl zásadní krok pro izolaci úloh a stabilitu systému. S/370 s virtuální pamětí navíc umožnil rozvoj hypervizoru VM/370 (1972), který dovolil na jednom mainframu spouštět více virtuálních strojů s různými operačními systémy paralelně. VM/370 uměl v separovaných virtuálech provozovat např. kopie OS/MVS nebo jednoduchý interaktivní systém CMS, určený pro jednotlivého uživatele – tím IBM vytvořilo jeden z prvních time-sharing systémů, kde desítky uživatelů sdílely stroj, ale každý měl iluzi vlastního počítače. Tímto se položil základ dnešních TSO (time sharing option) používaných na IBM Z [68].

2.5 Mikroprocesorové mainframy a zrod IBM Z

V 90. letech IBM představila řadu IBM System/390. Až do poloviny 90. let stavěla IBM nejvýkonnější mainframy z bipolárních ECL obvodů[3]. Zlom nastal koncem 90. let, kdy model S/390 G4 (1997) s CMOS procesory dosahoval téměř stejného výkonu jako starší bipolární systémy. Model G5 (1998) zdvojnásobil výkon oproti svým předchůdcům, jeho procesory dosahovaly rychlosti až 500 MHz a mohl obsahovat až 10 procesorů. Vedle hlavního procesoru byl vždy jeden koprocesor – buď na vektorové zpracování, nebo akceleraci šifrování.

Parallel Sysplex umožnil propojit více mainframů S/390 do jednoho logického celku se sdílenými daty. To umožnilo horizontální škálování.

IBM modernizovalo také I/O subsystémy. Vyměnilo staré paralelní „Bus and Tag“ kabely za optické kanály ESCON[4] [66] [69].

Architektura a vývoj do současnosti

V roce 1988 představilo architekturu Enterprise Systems Architecture/370 (ESA/370), která zavedla koncept vícenásobných adresních prostorů – pomocí přístupových registrů mohl program pracovat současně se dvěma různými adresovými prostory. To umožnilo efektivnější multitasking a izolaci. V roce 1990 pak přišla architektura ESA/390, definující pátou generaci instrukční sady od S/360. ESA/390 rozšířila instrukční sadu dle standardu IEEE 754 pro aritmetiku s plovoucí čárkou a zvýšila počet registrů pro FPU z 4 na 16. ESA/390 používala 31 bitů pro adresaci paměti, přestože používala 32bitové registry a aritmetiku.

Tohle byla poslední generace využívající tuto architekturu, dokud IBM neuvedla v roce 2000 nový model z900 s novou architekturou z/Architecture, která je 64bitová. Tím se završila dlouhá evoluce od 24bitového S/360 k 31bit ESA/390. IBM začlenilo podporu

[3] ECL (Emitter-Coupled Logic) je typ bipolární logiky, která umožňuje velmi rychlé přepínání díky provozu v aktivní oblasti tranzistoru. Vyznačuje se extrémní rychlostí, ale také vysokou spotřebou energie a tepelnou zátěží.

[4] ESCON (Enterprise Systems Connection) je proprietární optické rozhraní vyvinuté IBM jako náhrada za starší paralelní kabeláž typu „Bus and Tag“. Umožňuje vyšší přenosové rychlosti, větší vzdálenosti a flexibilnější připojení periferií.

moderních paradigmat, např. objektově orientované jazyky Java, middleware (CICS, DB2) pro transakční zpracování a také začalo otevírat mainframe pro nové operační systémy. Významným milníkem v roce 2000 bylo uvedení oficiální podpory Linuxu na mainframe přes speciální procesory IFL [72].

Procesory IBM Z umí provozovat aplikace ve 24 bit, 31 bit i 64 bit módu. Tím se odstranily historické restrikce paměti a umožňuje to spouštět starší programy. IBM představilo první generaci IBM zSeries v roce 2000 (model z900), následovala nižší řada z800 (2002). V dalších letech přišly modely z990 (2003), System z9 (2005), System z10 (2008). Následně IBM uvedlo novou generaci mainframů pod značkou zEnterprise, která představovala další evoluci architektury s důrazem na hybridní výpočetní modely. Umožňovala například propojení mainframových a distribuovaných systémů do jednoho řízeného celku. Tato linie zahrnovala dvě generace: zEnterprise 196 (2010) a zEnterprise EC12 (2012). Za moderní modely se dají považovat IBM z13 (2015), IBM z14 (2017), IBM z15 (2019) a současný IBM z16 (2022). IBM také změnilo marketingové označení – přes System z nebo Z series až k aktuálnímu jednoduššímu názvu IBM Z. Svět mainframů a zákazníci mohou očekávat také nový model, a to IBM z17, který obsahuje nové čipy a vychází v půlce roku 2025.

Každá nová generace přinášela výrazný růst výkonu a další technologické inovace, avšak architektura IBM Z kontinuálně zachovává zpětnou kompatibilitu s aplikacemi napsanými již pro IBM System/360 z roku 1964 [65, 71, 83].

Kapitola 3

Současnost Mainframů

K pochopení, proč se dnes mainframy stále nasazují v určitých, kritických systémech, je potřeba pochopit celkovou architekturu mainframů. Ta se liší od běžných osobních počítačů nebo i serverů. Dá se říct, že každý výrobce používá své vyvinuté technologie, software i hardware. Někteří výrobci, zejména IBM, používají i své speciální procesory, zatímco menší firmy nasazují např. Intel Xeon procesory. Mimo procesorů mainframy obsahují i další speciální komponenty. Na mainframech běží i operační systémy, které jsou speciálně navržené na efektivní provoz. Tato kapitola pokrývá hlavní komponenty IBM Z mainframů, současné využití a zmiňuje i konkurenční výrobce.

3.1 Technologická architektura IBM Z

Mainframy IBM Z (dříve System z) navazují na již zmíněnou architekturu S/360 z roku 1964 a zachovávají plnou zpětnou kompatibilitu, čili aplikace napsané pro S/360, S/370 či ESA/390 lze stále provozovat i na nejnovějších IBM Z systémech [71]. Současná instrukční sada z/Architecture je 64bitová CISC architektura navržená pro maximální propustnost a spolehlivost [39]. Procesory IBM Z patří k technologické špičce. V dnešní době si společnost IBM procesory navrhuje a výroba je delegována na Samsung Electronics. **IBM Telum**, představený v roce 2021, obsahuje 8 jader s hlubokým out-of-order superskalárním pipeline s maximální frekvencí 5,2 GHz. Tento čip byl později v roce 2022 použit v nejnovějším mainframu – IBM z16. Každé jádro Telum má 32 MB vlastní L2 cache a až 32 těchto čipů lze propojit v jednom systému. Díky tomu může nejnovější mainframe obsahovat desítky procesorů – předchozí generace IBM z15 umožňovala až 190 CPU jader pro aplikační zátěž (CP nebo IFL) v jednom systému a obsluhovat až 40 TB RAM. Na rozdíl od běžných serverů jsou procesory IBM Z navrženy pro extrémní spolehlivost: čipy obsahují redundantní jednotky a logiku pro „hot-failover“ (při zjištění závady se přepnou na záložní komponenty) s cílem dosáhnout „zero downtime“ provozu. Důraz je kladen i na permanentní výkon – CPU IBM Z neaplikují termální throttling snížením frekvence, ale při přehřátí krátkodobě vkládají uspávací instrukce, aby se předešlo chybám, aniž by klesla taktovací frekvence [40]. IBM Telum navíc jako první v historii mainframů integruje AI akcelerátor pro strojové učení přímo na čipu – umožňuje provádět inference neuronových sítí v reálném čase přímo během zpracování transakcí [70].

V roce 2024 byl představen nástupce IBM Telum, a to **Telum II**. Druhá generace procesoru Telum je vyráběna pomocí 5nm technologie společností Samsung Electronics a dokáže běžet na frekvenci 5,5 GHz. O 40 % navyšuje kapacitu cache paměti. Telum II obsahuje 8

jader a každé jeho jádro má 36 MB L2 cache. Tato generace bude nasazena v nejnovější verzi **IBM z17** mainframu, který se očekává uprostřed roku 2025 [33].

Technické specifikace IBM z17
(Vydané 8. dubna 2025) Oproti předchozí generaci z16, čip v z17 – Telum II redukuje o 20 % plochu jádra procesoru, dosahuje o 11 % vyššího výkonu na jedno vlákno a o 17 % snižuje spotřebu energie. Celý systém podporuje až 208 procesorových jednotek (PU) a až 64 TB operační paměti DDR5 DDIMM. Integrovaný on-chip AI akcelerátor (AIU) poskytuje více než 24 TFLOPS pro real-time inferenci, sdílených mezi všemi jádry čipu. IBM Spyre Accelerator představuje přizpůsobitelnou architekturu schopnou integrovat více modelů umělé inteligence a LLM. Inference AI je 7,5krát větší pro „Credit Card Fraud Detection“ oproti z16. IBM z17 podporuje až 6 podsystémů logických kanálů (LCSS), 4 subchannel sety a 85 LPARů. Navíc je zde integrovaná kvantově bezpečná ochrana, která je zajištěna prostřednictvím rozhraní API pro kvantově bezpečnou kryptografii a nástrojů pro zjišťování kryptografických informací. Technologie „Quantum-safe secure boot“ pomáhá chránit firmware IBM z17 před kvantovými útoky prostřednictvím vestavěného schématu dvojího podpisu, přičemž k aktivaci nejsou nutné žádné změny konfigurace [55].

Architektura IBM Z obsahuje vedle hlavních CPU řadu specializovaných koprocesorů a jednotek. Jednou z těchto komponent je Channel Subsystem. Každý mainframe disponuje kanálovými procesory pro vstup/výstup (dále pouze I/O), které přebírají zpracování I/O operací a umožňují masivní paralelismus při komunikaci s úložišti a IP porty. Dále IBM Z integruje **CPACF** (CP Assist for Cryptographic Functions) – hardwarový šifrovací modul se speciální instrukční sadou, dostupný na každém jádře CPU, urychlující symetrickou kryptografii (DES, 3DES, AES-128, SHA-256 apod.) pro “clear-key” šifrování. Pro potřeby šifrování s utajenými klíči slouží volitelné dedikované moduly Crypto Express (HSM karty splňující FIPS 140-2 Level 4) [22], které bezpečně uchovávají klíče a zajišťují např. rychlé SSL/TLS operace [30, 71]. IBM z15 zavedl také akcelerátor Integrated Accelerator for zEDC pro hardwarovou kompresi dat, která umožnila až 6× vyšší kompresní poměr a 42× rychlejší přenos souborů oproti předchozí generaci [40]. IBM mainframy také využívají specializované procesory pro určité scénáře: např. **zIIP** (System z Integrated Information Processor) čip je určen pro databázové a analytické úlohy nebo **IFL** (Integrated Facility for Linux) čip vyhrazen pro Linuxové instance. Tyto specializované CPU se licencují levněji a umožňují efektivněji škálovat dané typy zátěže [72, 84].

Virtualizace a dělení

Hluboká virtualizace je jedním z poznávacích znaků mainframů. IBM Z má virtualizaci vestavěnou již na první vrstvě, a to hardware – každý systém obsahuje **PR/SM (Processor Resource/System Manager)**, který umožňuje rozdělit fyzický stroj až na desítky logických celků **(LPAR)** izolovaných už na úrovni mikrokódu. Například IBM z13 podporoval až 80 LPARů na jednom fyzickém systému. IBM z16 podporuje až 85 LPARů. Pro účely virtualizace provozuje IBM mainframy s hypervizorem **z/VM**. Z/VM je nástupcem historického VM/370 a je optimalizován pro hostování velkého počtu linuxových a z/OS virtuálních strojů. Běží přímo na LPARu, který dovoluje spouštět desítky až stovky virtuálních strojů v rámci jedné logické partition. Celkově tak jediný mainframe dokáže konsolidovat stovky až tisíce virtuálních serverů. Výhodou IBM virtualizace je také dynamická konfigurovatelnost – CPU, paměť i I/O zdroje lze za běhu přidávat nebo odebírat z LPARů podle

momentální zátěže, bez přerušení provozu. Případně jdou zdroje sdílet mezi více LPARy podle potřeby. Izolace je certifikována na vysoké úrovni bezpečnosti – PR/SM získal certifikaci Common Criteria Evaluation Assurance Level 5+ (EAL5+), což dokládá, že oddělení partition splňuje přísné bezpečnostní požadavky (důležité např. prostředí bank). IBM rovněž přidal na mainframe open-source hypervizor KVM, takže vedle z/VM je k dispozici i alternativní virtualizační technologie pro Linux on Z. z/VM také umožňuje tzv. second-level virtuální stroje – tj. uvnitř VM lze opět spustit z/VM (nebo KVM) a vytvořit další úroveň hostů, což svědčí o robustnosti a výkonové rezervě systému. Samotný z/VM je velmi odlehčený a optimalizovaný; získal bezpečnostní certifikaci EAL4+. Mimo Linuxu umí z/VM hostovat i další IBM OS (z/OS, z/VSE, z/TPF) pro testovací a vývojové účely – často se používá pro izolované sandboxy z/OS pro vývojáře [6, 30, 31, 71].

Bezpečnostní prvky

Mainframy jsou navrženy tak, aby splňovaly nejpřísnější, již zmíněné, bezpečnostní standardy. IBM z15 začal na svých CPACF koprocesorech podporovat i algoritmy kryptografie nad eliptickými křivkami (ECC) pro P-256, P-384, P-521, Ed25519 a Ed448. Kromě již uvedených šifrovacích akcelerátorů hardware podporuje koncept všudypřítomného šifrování. IBM Z umí šifrovat veškerá data v paměti i na discích transparentně, bez zásahu aplikací, čímž chrání citlivé informace (funkce zavedená od z14). Bezpečnost posilují i specializované firmwarové kontejnery, třeba technologie IBM Secure Service Container umožňuje provozovat citlivé aplikace (jako certifikační autority, blockchain uzly apod.) v izolovaném prostředí, kam ani administrátor nemá přímý přístup. Na úrovni operačního systému z/OS je integrován **RACF (Resource Access Control Facility)** – centrální modul řízení přístupu, který vynucuje autorizace ke všem zdrojům. Mainframy IBM z15 dále představily funkci Data Privacy Passports pro end-to-end šifrování dat sdílených mimo mainframe – data opouštějící systém jsou opatřena „pasem“ a mohou být dešifrována jen oprávněnými systémy. Celkově tak IBM Z platforma dosahuje právem pověsti nejbezpečnějšího komerčně dostupného systému, to je důvod, proč je často nasazována tam, kde jsou požadavky na bezpečnost a dodržování předpisů (certifikace) nejvyšší (banky, vládní instituce apod.) [6, 25, 30].

3.2 Operační systémy IBM Z

IBM z/OS

Hlavním operačním systémem na IBM Z mainframech je z/OS, přímý potomek OS/360 a MVS. Z/OS je plně 64bitový operační systém podporující 24bitové, 31bitové i 64bitové adresování, což umožňuje aplikacím přepínat mezi těmito režimy pro zajištění zpětné kompatibility. Jedná se o vysoce škálovatelný, zabezpečený a robustní OS navržený pro smíšené workloady – na jednom z/OS systému mohou souběžně běžet dávkové úlohy, transakční systémy, databáze, middleware i moderní aplikace. Z/OS obsahuje pokročilý **workload management (WLM),** který dynamicky přiděluje zdroje úlohám podle definovaných priorit a cílů, čímž zajišťuje optimální využití hardware. Správce systému klasifikuje práci do tzv. „služebních tříd“ (service classes) podle atributů, jako jsou názvy transakcí, identifikace uživatelů nebo názvy programů. Každé třídě jsou přiřazeny cíle, jako je doba odezvy nebo výkonnost, a úroveň důležitosti. WLM monitoruje systémové zdroje a stav úloh, aby zajistil splnění těchto cílů, a v případě potřeby upravuje přístup úloh k prostředkům systému. Samozřejmostí je virtuální paměť a bohaté možnosti škálování – z/OS podporuje Parallel

Sysplex clustering, který umožňuje propojení více fyzických mainframů do jednoho logického celku.

Klíčovou komponentou je **Coupling Facility (CF)**, která poskytuje struktury pro zámky, seznamy a cache, což umožňuje sdílení dat a koordinaci mezi systémy. To zajišťuje vysokou dostupnost a téměř nepřetržitý chod systému, i během údržby nebo při selhání jednotlivých komponent. Systém poskytuje množství subsystémů pro enterprise computing. Transakční monitor CICS a zpracování transakcí IMS TM, databázové systémy IBM Db2 a IMS DB, fronty zpráv MQ Series, bezpečnostní systém RACF, systém správy tiskových/dávkových úloh JES2/JES3, a dokonce integrované UNIX prostředí (Unix System Services), které zpřístupňuje POSIX rozhraní a umožňuje provoz skriptů, Unix utilit a některých linuxových aplikací přímo na z/OS. Moderní verze z/OS plně podporují Javu prostřednictvím IBM SDK pro z/OS, což umožňuje provozovat Java aplikace s vysokou efektivitou. Navíc z/OS obsahuje z/OS Container Extensions (zCX), které umožňují provoz Docker kontejnerů přímo na z/OS. To usnadňuje integraci moderních aplikací a mikroslužeb do mainframového prostředí bez nutnosti jejich přepisování i technologie jako Cloud Paks od IBM, což usnadňuje integraci mainframu do hybridního cloudu. AI/ML a DevOps: z/OS podporuje moderní vývojové metodiky, jako je DevOps, a integraci s nástroji pro kontinuální integraci a nasazení (CI/CD). Projekt Zowe, otevřený framework, poskytuje moderní rozhraní pro interakci se z/OS, včetně příkazového řádku (CLI), API a webového rozhraní. To umožňuje vývojářům a administrátorům efektivněji spravovat a automatizovat zdroje a služby na z/OS. Z/OS je považován za extrémně spolehlivý OS – často běží bez restartu po celé roky a zároveň za velmi bezpečný. I když si z/OS nese určité legacy prvky, jako je textové rozhraní ISPF a Job Control Language (JCL) pro dávkové úlohy, IBM jej neustále modernizuje. Například z/OS 3.1 přinesl vylepšení pro AI/ML workloady a nástroje pro DevOps integraci, což usnadňuje integraci mainframu do moderního IT prostředí [6, 31].

Job Entry Subsystem (JES)

Z pohledu řízení a správy dávkového (batch) zpracování na mainframech je velice zásadní komponentou z/OS **Job Entry Subsystem (JES)**. Dávkové úlohy jsou definovány prostřednictvím Job Control Language (JCL). JES přijímá vstupy jobů z TSO, ISPF (terminálový přístup) nebo přes síťový submit, zařazuje je do vstupní fronty, kontroluje syntaxi a přiděluje jim prostředky k vykonání. Po skončení jobu JES ukládá jeho výstupy (logy, datové výstupy, reporty) do tzv. spool prostoru (Simultaneous Peripheral Operations Online), který představuje sdílený diskový prostor optimalizovaný pro rychlé čtení a zápis mnoha paralelních úloh.
Z technického pohledu JES zajišťuje a poskytuje následující funkce:

- **Job scheduling a prioritizace** – JES organizuje joby podle definovaných priorit, plánovaných časů spuštění a dostupnosti systémových zdrojů (procesory, paměť, I/O zařízení). Tyto priority může řídit přímo uživatel v rámci JCL definicí (pomocí parametrů jako CLASS, PRIORITY), nebo mohou být dynamicky ovlivňovány pomocí Workload Manageru (WLM).
- **Správa spool prostoru** – JES alokuje a spravuje prostor pro dočasné ukládání dat z jobů a umožňuje paralelní provádění vstupních a výstupních operací. Výstupy jsou následně dostupné k tisku, prohlížení uživatelem (přes ISPF) nebo pro další automatické zpracování.

- **Správa front úloh (Job Queues)** – JES organizuje úlohy do front podle stavu (čekající na spuštění, běžící, dokončené) a řídí jejich postupné nebo paralelní zpracování dle dostupnosti systémových zdrojů a priorit. To umožňuje efektivní využití procesorových cyklů i periferních zařízení, například tiskáren nebo výstupních zařízení.
- **Podpora výstupních zařízení (Output Management)** – JES umožňuje tisk a distribuci reportů či výsledků dávkových úloh na různé typy periferií, jako jsou tiskárny, externí úložiště nebo další komunikační kanály.

Historicky existují dvě varianty JES. Obě verze se liší přístupem k řízení úloh.

JES2 (původně z HASP – Houston Automatic Spooling Priority) je jednodušší, decentralizovaný systém. Každý uzel JES2 pracuje relativně samostatně s vlastními frontami jobů, což umožňuje vysokou škálovatelnost a flexibilitu. Dnes je JES2 dominantní variantou používanou většinou organizací, zejména díky své jednoduchosti a flexibilitě správy.

JES3 (z ASP – Attached Support Processor) je komplexnější subsystém, který centralizovaně plánuje a koordinuje joby a zdroje napříč více systémy v prostředí Parallel Sysplex. JES3 umožňuje centrálně řídit pořadí jobů, rezervace zdrojů (diskových jednotek, pásek apod.) a sofistikovanější zpracování úloh v rámci rozsáhlých systémů. IBM nicméně oznámila, počínaje verzí z/OS 3.1, vydanou v září 2023, že již IBM z/OS neobsahuje JES3 a bude se dodávat se pouze s JES2. IBM umožňuje migraci z JES3 na JES2. Společnost Phoenix Software International převzala budoucí podporu a vývoj JES3 od IBM.

V kontextu Parallel Sysplex clusteringu, JES2 umožňuje tzv. Multi-Access Spool (MAS) konfiguraci, kdy více systémů může sdílet jeden společný spool prostor. Tím JES2 podporuje vyšší dostupnost a vyrovnání zátěže napříč více systémy v rámci Parallel Sysplexu. Pokud dojde k plánované či neplánované odstávce některého uzlu, joby mohou být automaticky převzaty a zpracovány jiným dostupným uzlem.

Přestože JES2 i JES3 využívají starší rozhraní (textové příkazy, JCL definice), IBM průběžně zajišťuje jejich modernizaci, například integrací JES s moderními nástroji pro správu typu Zowe CLI, REST API přes z/OSMF nebo Ansible pro automatizaci operací, což významně zjednodušuje integraci JES do moderních DevOps praktik [6, 27, 31].

Další IBM mainframe OS

Kromě z/OS existují i specializované systémy pro IBM Z. **z/VSE (Virtual Storage Extended)** je odlehčený OS určený pro menší mainframy a velmi specifické batch workloady. Původně známý jako Disk Operating System (DOS) byl prvním diskovým operačním systémem zavedeným pro mainframy System/360, dnes stále udržovaný pro některé zákazníky, ale jeho podíl je malý (často běží jako guest na z/VM) [28].

z/TPF (Transaction Processing Facility) je velmi úzce zaměřený OS pro extrémně rychlé zpracování jednoduchých transakcí – využívají jej například globální rezervační systémy leteckých společností, hoteliérů, nebo například Visa Inc. pro autorizaci plateb. V těchto odvětvích je potřeba ve zlomku sekundy zpracovat obrovské množství velmi krátkých transakcí. Z/TPF má minimální režii a je optimalizován na propustnost systému, běží přímo na LPAR bez zbytečných subsystémů [80].

Linux on IBM Z

Jedním z nejvýznamnějších trendů posledních circa 20 let je provoz Linuxu na mainframu. IBM portovala Linux kernel na architekturu z/Architecture již kolem roku 2000 a od té

doby podporuje několik distribucí (SUSE, Red Hat, Ubuntu) na svých mainframech. Počínaje verzí linuxového jádra 4.1 vydanou na začátku roku 2015 je Linux na IBM Z k dispozici pouze jako 64bitový operační systém kompatibilní s mainframy architektury z/Architecture. Linux může běžet buď přímo na LPARu, nebo uvnitř z/VM jako plně virtuální instance. Linux on Z si získal popularitu, protože kombinuje otevřený ekosystém Linuxových aplikací s hardwarovou spolehlivostí mainframe. Linux on Z těží z funkcí mainframu, jako je výše zmíněné šifrování, rychlá komunikace mezi VM (hypervizorová sdílená paměť) a možnost přímého připojení ke SAN úložištím a sálovým tiskárnám, tradičně používaným v mainframe prostředí. Linux běží na standardních mainframových procesorech, ale i na **IFL (Integrated Facility for Linux)**. IFL jsou mainframové procesory určené pro běh systému Linux, a to buď nativně, nebo pod hypervizorem (z/VM nebo KVM na IBM Z). IBM nabízí i speciální modely mainframů LinuxONE určené výhradně pro Linux (hardware je totožný s IBM Z, jen licencování je pouze pro IFL procesory). Linuxové workloady na mainframech sahají od provozu databází a ERP systémů až po cloudové aplikace. Linux on Z je dobrý pro konsolidaci. „16 IFL procesorů a 1 TB RAM v LinuxONE může nahradit mnoho x86 serverů“ uvádí například SUSE. Hlavní přínosy jsou v úspoře místa, energie a centralizované správě [26, 36, 74].

3.3 Současné využití

Mainframy si i v dnešní éře spíše cloudových řešení různých systémů, drží pevné místo tam, kde jsou nejvyšší nároky na škálovatelnost, spolehlivost a bezpečnost. Typické oblasti nasazení jsou:

- **Bankovnictví a finance** – Mainframy zpracovávají obrovské objemy finančních transakcí, např. karetní transakce, bankomaty, zúčtování plateb a core-banking systémy. Důvodem je schopnost mainframu provést desítky tisíc transakcí za sekundu se zárukou konzistence, bezpečnosti a bez výpadků. Banky také ocení dlouhý lifecycle – mainframe aplikace (často v COBOLu) běží i desítky let a stále je lze provozovat na nejnovějším hardware bez přepisování. Podle IBM využívá mainframy 45 z 50 největších bank světa a odhaduje se, že většina mezibankovních transakcí někde na pozadí projde mainframem [44].
- **Pojišťovnictví a zdravotnictví** – Velké pojišťovny používají mainframy pro správu pojistných smluv, výpočty rezerv, zpracování pojistných událostí – opět kvůli spolehlivosti a možnosti pojmout extrémní datové objemy. Zdravotní pojišťovny a státní systémy zdravotnictví, např. správa dávek, nemocniční informační systémy také často běží na mainframech. Typické jsou dávkové zpracování přes noc – mainframe zpracuje miliony záznamů v omezeném čase [34].
- **Vládní systémy a veřejný sektor** – Mnoho státních systémů jako registry obyvatel, daňové systémy, sociální zabezpečení, systémy armády jsou postaveny na mainframech pro jejich bezpečnost a dlouhou životnost. Např. v USA IRS (daňová správa) provozuje mainframe aplikace napsané ještě v 60.–70. letech, které stále fungují. V České republice mainframe využívá třeba Ministerstvo vnitra pro vyplácení důchodů [2, 49].
- **Doprava a telekomunikace** – Rezervační systémy leteckých společností tradičně běží na mainframech a využívají dříve zmíněný OS z/TPF. Železniční rezervační systémy, systémy řízení letového provozu apod. také spoléhají na mainframy. V telekomunikacích se mainframy používaly pro účtování hovorů, i když dnes je částečně

nahrazují velké Unix servery. Rezervační systém Sabre například v minulých letech přešel z mainframů od IBM na cloudové služby [3, 52].

- **Retail a výroba** – Některé velké maloobchodní řetězce a výrobní podniky využívají mainframy pro správu skladového hospodářství, zpracování objednávek a logistiku. Například společnost Walmart používá mainframe pro svůj centrální skladový systém [37].

Mezi hlavní důvody, proč si velké firmy volí mainframy, namísto výkonných serverů, patří:

- **Spolehlivost a dostupnost** – Mainframy dosahují dostupnosti větší něž 99,999 999 % (pouze milisekundy neplánovaných výpadků ročně). Mají redundanci na všech úrovních, možnost rolling upgrade (aktualizace za provozu) a odolnost vůči chybám. Pro kritické aplikace, kde neplánovaný výpadek by znamenal obrovské ztráty, je to klíčové [28].
- **Škálovatelnost a výkon** – Mainframe zvládne konsolidovat stovky serverů a obsluhovat tisíce procesů paralelně. Má extrémně vysokou propustnost I/O – díky specializovaným I/O procesorům a kanálové architektuře dokáže simultánně obsluhovat tisíce I/O operací, a to bez zatížení hlavních procesorů. Typický systém obsahuje desítky FICON či OSA-Express kanálů, přes něž komunikuje s rozsáhlým polem diskových jednotek a síťových zařízení. Např. IBM uvádí zpracování miliard webových transakcí denně na jediném systému. Hlavní síla není ani tak v hrubém výpočetním výkonu jednoho jádra (byť GHz jsou vysoké), ale v celkové propustnosti systému a optimalizaci pro typické workloady (transakce, DB) [44].
- **Bezpečnost** – Mainframy získaly důvěru díky robustnímu zabezpečení – kombinace izolace LPARů, šifrování dat, silné autentizace a auditování znamená, že citlivá data (osobní údaje, finanční data) lze zpracovávat s minimálním rizikem úniku. Např. mainframe může šifrovat celou databázi v klidu i při přenosu bez vlivu na výkon. To je důležité v odvětvích s regulací (GDPR, PCI-DSS pro platby apod.) [40].
- **Dlouhodobé zachování investic** – Organizace, které mají v mainframech desítky let vývoje (miliony řádků COBOL, PL/I), je nemusejí kompletně přepisovat – mainframe zajistí, že i 50 let stará aplikace běží na nejnovějším hardware. To šetří náklady na redesign a zároveň minimalizuje riziko chyby při přepisu osvědčeného systému [71].
- **Konsolidace a celkové náklady spojené s vlastnictvím** – Ač se mainframe může zdát nákladný, v mnoha případech se vyplatí díky konsolidaci. Jeden stroj ve skříni o velikosti lednice může nahradit celou místnost serverů – s nižšími nároky na prostor, energie a správu. Např. starší IBM z10 BC inzeroval, že má kapacitu až 232 x86 serverů při zlomku jejich spotřeby. Centralizovaná infrastruktura zároveň zjednodušuje správu a snižuje provozní složitost. V některých případech také licenční modely mainframového softwaru umožňují nákladově efektivní škálování při rostoucím počtu úloh [47].
- **Transakční integrita** – Mainframe systémy (z/OS + DB2/CICS) jsou známé tím, že zajišťují ACID vlastnosti transakcí i ve velmi komplexních scénářích, a to s vysokou efektivitou. CICS umožňuje zpracování tisíců transakcí za sekundu na jedno jádro, přičemž garantuje, že každá transakce je fyzicky zapsána na disk před potvrzením. Síla tedy spočívá v deterministické stabilitě, ochraně konzistence dat v prostředí s vysokou mírou paralelismu a transakční zátěže, než absolutnímu výkonu [35].

Samozřejmě, mainframy nejsou vhodné pro každé použití. Pro vědecké výpočty se využívají spíše superpočítače s paralelními CPU/GPU – mainframy excelují v transakčním zpracování, ne v floating-point výkonech. Také tam, kde není potřeba nonstop dostupnost, mohou vyhovovat levnější verze běžných serverů. Náklady a dostupnost expertů jsou dalším faktorem – specializovaná práce s mainframy (systémový programátor z/OS, operátor) vyžaduje zkušenosti, kterých ubývá. Proto se někdy volí přesun aplikací z mainframu na jiné systémy. Navzdory těmto tlakům však mainframy nadále přežívají a inovují – IBM integruje AI akceleraci, zlepšuje programátorskou přívětivost (podpora moderních jazyků, API) a výrobci se snaží umožnit hybridní provoz (mainframe + cloud) [75].

3.4 Konkureční mainframy

Mimo IBM, nabízí mainframové systémy i jiné společnosti, nicméně tyto dále popsané systémy ukazují, proč je IBM dominantem v tomto sektoru.

Fujitsu GS21

Fujitsu je jedním z mála výrobců klasických mainframů s vlastní architekturou mimo IBM (mimo něj existuje pouze Hitachi s minoritním podílem kompatibilních systémů v Japonsku). Fujitsu řady GS21 (původně známé jako série GlobalServer) byly po desetiletí nasazovány zejména v japonských bankách, pojišťovnách a vládních institucích. Firma vždy zdůrazňovala interoperabilitu svých mainframů s otevřenými systémy, přičemž standardně podporovala propojení s UNIX/Linux servery.

Navzdory dlouhodobé tradici Fujitsu oznámilo strategické rozhodnutí tento segment trhu opustit. Prodej mainframů řady GS21 bude ukončen do roku 2030 s garantovanou podporou do roku 2035. Firma v současnosti připravuje migraci zákazníků na alternativní platformy, zejména na cloudová řešení nebo re-hosting aplikací. Poslední generace GS21, vydaná v roce 2024, má sloužit jako přechodné řešení, než budou zákazníci definitivně migrováni.

Kromě GS21 nabízelo Fujitsu (resp. Fujitsu-Siemens) také evropské mainframy řady BS2000/OSD se samostatnou klientelou. Tyto systémy a jejich operační prostředí BS2000 budou také ukončeny do roku 2030.

Architektura mainframů Fujitsu GS21 vychází z platformy IBM ESA/390 (příbuzná architektuře IBM S/390 z 90. let), což znamená, že stále využívá 31bitovou adresaci a 32bitová data. Z tohoto pohledu je platforma technicky pozadu za současnou 64bitovou architekturou IBM Z.

Z hlediska hardwaru Fujitsu vyvinulo vlastní vícejádrové CPU s technologií „system-on-chip“. Tyto CPU mají až 8 jader na jednom čipu, každé jádro obsahuje 256 KB L1 cache, sdílenou L2 cache až 20 MB, integrovaný řadič paměti, I/O procesor a systémový kontrolér. Novější modely GS21, jako např. GS21 3600 uvedený v roce 2018, dosáhly nárůstu výkonu (cca o 20 % oproti předchůdci) a současně snížily fyzické prostorové nároky až o 40 %.

Fujitsu GS21 umožňuje propojení více skříní (clusterů) pro zvýšení celkového výkonu, avšak možnosti škálování jsou omezenější než u konkurenčních systémů IBM. Například GS21 3400 podporuje maximálně 4 CPU s 64 GB RAM, vyšší model GS21 3600 pak až 16 CPU s 256 GB RAM. Pro srovnání, IBM z17 umožňuje škálování až na 208 CPU a 64 TB RAM.

Mainframy Fujitsu využívají proprietární hypervizor s podporou virtualizace, která byla v posledních generacích významně vylepšena. Například model GS21 3400 díky zlepšením

virtualizace umožňuje provoz více než 10 virtuálních strojů na cluster (dříve byly limitem pouze 3 virtuální stroje), s podporou konfigurace až 20 000 virtuálních I/O zařízení na cluster.

Operačními systémy mainframů Fujitsu GS21 jsou OSIV/MSP (Mainframe System Platform) určený pro high-end systémy a OSIV/XSP pro systémy střední třídy. Oba OS jsou proprietární a jejich vývoj dlouhodobě navazuje na operační systémy původních sálových počítačů FACOM.

Systémy MSP i XSP umožňují jak dávkové zpracování, tak i transakční zpracování. Podporují rovněž tightly-coupled multiprocessing. Mezi klíčové subsystémy patří například databázový systém Symfoware a transakční monitor PowerAIM.

Fujitsu klade v OS důraz na zabezpečení, zejména díky rozšířenému řízení přístupu a logování administrátorských akcí přes SVPM konzoli. Softwarová výbava těchto OS dále obsahuje nástroje umožňující přímé napojení otevřených systémů (např. Linux/UNIX) k mainframovým databázím a aplikacím. Globálně jsou operační systémy MSP a XSP málo známé a nasazované především na japonském trhu [9, 10, 13, 11, 12, 41].

HPE NonStop

Hewlett Packard Enterprise NonStop (dříve Tandem Computers) představuje specifickou kategorii systémů, které jsou HPE označovány nikoli jako klasické mainframy, nýbrž jako fault-tolerant servery. Přesto jsou často zvažovány jako alternativa k tradičním mainframům v situacích, kde je zásadní absolutní dostupnost systému. NonStop systémy mají unikátní architekturu a operační systém, historicky zaměřený na extrémně spolehlivé OLTP transakce. Typické nasazení těchto systémů je v bankovnictví, platebních bránách, telekomunikacích nebo na burzách. Příkladem jsou některé platební systémy VISA.

Nejnovější generace systémů HPE Integrity NonStop X, uvedená od roku 2014, přešla na standardní procesory Intel Xeon (x86-64), čímž se otevřela možnost využít moderní hardware. Aktuální high-end model NonStop NS8 X4 (uvedený v roce 2021) přinesl zejména výrazné zvýšení výkonu interního propojení clusterových uzlů, zlepšenou konektivitu a další modernizaci hardwarové platformy.

Do budoucna stojí HPE NonStop před výzvou větší integrace s moderním softwarovým ekosystémem. Přestože NonStop zůstává úzkým segmentem v rámci celého portfolia HPE, společnost deklaruje dlouhodobou podporu této platformy.

Architektura HPE NonStop je výrazně odlišná od klasických mainframů, jelikož vychází z konceptu fault-tolerantních clusterů s tzv. shared-nothing architekturou (žádný centrální bod selhání). NonStop využívá více menších uzlů spojených do clusteru. Každý uzel obsahuje dvojici procesorů, které pracují v tzv. lock-step režimu (synchronní redundantní zpracování): v případě selhání jednoho procesoru okamžitě převezme jeho práci záložní procesor bez jakéhokoliv výpadku služby.

Dnešní NonStop systémy jsou postaveny na standardních Intel Xeon procesorech (od řady NonStop X uvedené v roce 2014), které nahradily původní proprietární CPU a pozdější Itanium procesory. Výkon těchto systémů se škáluje horizontálně[1], což je charakteristické pro tzv. shared-nothing architektury. Oproti tomu IBM Z tradičně využívá silně vertikální

[1] Horizontální škálování znamená zvyšování výpočetní kapacity přidáváním dalších nezávislých uzlů (např. serverů) do systému. Každý uzel má vlastní CPU, paměť i diskové prostředky a komunikuje s ostatními přes síťové propojení. Typické pro clusterové a cloudové systémy.

škálování[2] v rámci jednoho systému (desítky až stovky CPU, desítky TB RAM), přičemž horizontální škálování je řešeno prostřednictvím technologie Parallel Sysplex. NonStop systémy běžně dosahují dostupnosti přes 99,999 %.

Z pohledu virtualizace NonStop nepodporuje klasickou virtualizaci více různých OS na jednom fyzickém uzlu. Každý uzel v clusteru provozuje vlastní instanci NonStop OS. Odolnost aplikací je zajištěna jejich distribuovaným nasazením napříč více uzly. Každá kritická aplikace typicky běží jako primární proces spolu se záložním procesem na jiném CPU či uzlu. V případě výpadku primárního procesu systém okamžitě aktivuje záložní proces bez ztráty prováděných transakcí.

Operační systém HPE NonStop OS (původně nazývaný Guardian) je unikátní, proprietární systém vyvinutý pro clusterovou architekturu s vysokou dostupností. Nejedná se o klasický Unix-like OS, ale o specializovaný message-based systém s vlastním API označovaným jako Guardian procedures. Tento systém je od základu navržen pro distribuované, redundantní zpracování aplikací.

NonStop OS obsahuje dvě základní vrstvy:

- **Guardian** – nízkoúrovňová, proprietární část, která se stará o základní systémové funkce, message-passing, správu redundance procesů a automatické přepínání při výpadcích.
- **OSS (Open System Services)** – vrstva nad Guardianem, která nabízí POSIX kompatibilní rozhraní, čímž usnadňuje portaci aplikací a provoz UNIXových nástrojů.

Pro aplikace se používá databázový systém NonStop SQL/MX, specializovaný pro extrémně spolehlivé a výkonné OLTP operace. Aktualizace operačního systému probíhají bez odstávky, tzv. „upgrades in place“, což umožňuje kontinuální provoz bez přerušení dostupnosti služeb.

Vývoj aplikací pro NonStop OS vyžaduje specifické znalosti, například zvládnutí programovacího jazyka TAL (Transaction Application Language) nebo speciální úpravy v jazycích C/C++ s využitím Guardian API. Celkově jde o systém s úzce zaměřeným ekosystémem, který zahrnuje specifické nástroje a middleware určený pro správu, monitoring a replikaci dat ve vysoce dostupných prostředích [14, 15, 16, 77].

Bull/Atos (Eviden) GCOS

Bull byl původně tradiční francouzský výrobce mainframů a serverových řešení. Historicky šlo o firmu s kořeny sahajícími až do 30. let 20. století. Bull dlouhá léta vyvíjel své vlastní mainframy, zejména známé řady GCOS.

V roce 2014 byl Bull odkoupen společností Atos, což je velká francouzská IT firma orientovaná především na služby (outsourcing IT, cloud, kyberbezpečnost, digitální transformace apod.). Mainframe produkty (řady GCOS a později BullSequana M) nadále nesou značku Bull, ale společnost jako taková už je plně integrována do struktur Atosu.

V roce 2023 však došlo k velké restrukturalizaci firmy Atos. Důvodem byly finanční problémy a snaha zjednodušit strukturu podniku. Atos se rozdělil na dvě samostatné společnosti Atos a Eviden. Mainframy Bull (nyní BullSequana M) tedy oficiálně vyrábí a prodává Eviden. Samotný Atos (mateřská společnost) se nyní zaměřuje spíše na tradiční IT

[2] Vertikální škálování označuje navyšování výkonu rozšiřováním kapacity jediného systému – přidáváním CPU, paměti nebo I/O zdrojů do jedné výpočetní jednotky.

služby, a nikoliv na vývoj vlastních hardwarových produktů. Atos zvolil strategii převedení původně proprietární mainframové architektury na moderní x86 servery, přičemž nabízí jak fyzické mainframové systémy, tak čistě softwarová či cloudová řešení. Servery BullSequana M jsou navržené inženýry společnosti Eviden a založené na nejnovějších škálovatelných procesorech Intel Xeon. Typická nasazení GCOS zahrnují regulované sektory, například státní správu, bankovnictví nebo obranné aplikace.

Architektura BullSequana M existuje ve dvou řadách: **M7000** (určená pro OS GCOS7) a výkonnější **M9600** (pro GCOS8). Obě využívají vícejádrové Xeony a podporují konfigurace s vysokou kapacitou paměti a I/O pro zpracování velkých objemů dat. Mainframy BullSequana tak dokážou kontinuálně růst dle potřeb – od „klasického“ nasazení jednoho OS až po vysoce virtualizované konfigurace s více instancemi a různými workloady současně. Díky standardnímu hardwaru přebírají tyto mainframy výhodu široké podpory technologií: využívají moderní sběrnice PCI Express, rychlá úložiště (např. SAN) a síťová rozhraní běžná v enterprise sféře. Zároveň však zachovávají typické mainframe prvky jako redundance a vysoká dostupnost. Architektura BullSequana M například obsahuje pokročilé mechanismy pro toleranci chyb a nonstop provoz (hot-swap komponenty, modularita pro údržbu bez odstávky apod.), aby splnila nároky kritických aplikací. Výsledkem je platforma nabízející mainframe-class výkon a spolehlivost, avšak s využitím osvědčených komoditních komponent pro lepší poměr cena/výkon. Podle dostupných údajů tak Bull dosáhl výrazného snížení TCO oproti starším proprietárním systémům a usnadnil integraci těchto mainframů do moderních datových center.

Operační systémy: Bull mainframy historicky využívají rodinu OS GCOS (General Comprehensive Operating System), která sahá až do 60. let (původně vyvinuto firmou General Electric/Honeywell). Dnešní systémy BullSequana nadále podporují dvě hlavní větve: **GCOS7** a **GCOS8**, které jsou vyvíjeny paralelně. Tyto OS jsou zpětně kompatibilní s legacy aplikacemi z dřívějších mainframe řad DPS7/DPS7000 a DPS8/DPS9000. GCOS7 i GCOS8 si zachovávají tradiční mainframe schopnosti, jako je zpracování dávkových úloh, transakční zpracování OLTP a time-sharing pro více uživatelů. Například GCOS8 již od 80. let podporuje virtuální paměť a stránkování, zatímco GCOS7 vychází z jiné větve (Honeywell Level 62/ DPS7) a má odlišné interní architektury transakčního systému (TDS pro GCOS7 vs. TP8 u GCOS8). Pro uživatele však obě větve poskytují srovnatelnou úroveň robustnosti a funkcionalit typických pro mainframe OS (job scheduling, JCL skripty, vestavěné databáze atd.).

V rámci nové hardwarové koncepce došlo k portaci/emulaci GCOS na platformu Intel Xeon. Bull už dříve zavedl řadu Novascale (postavenou na Itaniu), kde GCOS7/8 běžely v režimu emulace nad Unix/Linux prostředím. Nyní jsou GCOS7 nasazovány na systémech BullSequana M7200 a GCOS8 na M9600, vždy prostřednictvím instrukční simulace na architektuře x86. Tato vrstva zajišťuje, že legacy binární kód může běžet beze změn i na zcela odlišném procesoru.

Zároveň BullSequana M umožňuje vedle GCOS provozovat i standardní operační systémy, Linux a Windows, a to buď alternativně, nebo dokonce současně v rámci jednoho systému. Každá fyzická jednotka BullSequana tak může hostovat například oddíl s GCOS8 pro core transakční systém banky a vedle toho oddíly s Linux/Windows pro doprovodné aplikace. Tato schopnost multiOS provozu je velký rozdíl proti starším mainframe generacím a demonstruje otevřenost platformy – Bull uváděl, že jeho mainframy jsou „nejotevřenější na trhu“ ve smyslu podpory standardních aplikací, což dnes v porovnání s konkurencí už není tak aktuální [7, 56, 60, 75].

Unisys ClearPath

Společnost Unisys (vzniklá spojením firem Burroughs a Sperry/UNIVAC) pokračuje ve vývoji mainframových řešení pod označením ClearPath Forward. Tyto systémy jsou specifické svou strategií využívání softwarové emulace namísto proprietárního mainframového hardwaru. Jeho současná mainframová linie ClearPath Forward staví stejně jako Bull na standardních Intel Xeon procesorech. Unisys dokončil přechod z proprietárních CMOS procesorů na platformu x86 v roce 2015 – šlo o velice složitý, desetiletý projekt, který vyústil v čistě softwarově definovanou mainframe architekturu běžící na Intel procesorech.

Společnost Unisys nabízí dvě hlavní produktové linie ClearPath:

- **Dorado** – využívá operační systém OS 2200 (navazuje na historický UNIVAC OS).
- **Libra** – využívající operační systém MCP (Master Control Program), který navazuje na původní Burroughs MCP.

Architektura je koncipována jako „fabric“: množina vzájemně propojených výpočetních uzlů (blade serverů) spojených vysokorychlostní propojovací technologií (Unisys zvolil InfiniBand z HPC světa). Tato softwarově definovaná síť spojuje výpočetní jednotky a jejich zdroje tak, že pro uživatele tvoří jednotný systém. ClearPath mainframy tak dokáží škálovat jak vertikálně (výkonné více-socketové servery), tak horizontálně (více uzlů v clusteru propojeném InfiniBandem).

Klíčovou komponentou architektury je Unisysem vyvinutá technologie **s-Par (Secure Partitioning)**. Jde o implementaci virtualizace typu „shared-nothing“, kde se o řízení všech workloadů stará softwarová vrstva hypervizoru s-Par využívající Intel Virtualization Technology hardware. s-Par vytváří na systému bezpečné oddíly (partition), z nichž každý má vyhrazené procesory, paměť i I/O kanály. Na rozdíl od běžné virtualizace zde nedochází k sdílení těchto zdrojů mezi oddíly, a tím se eliminují „noisy neighbor“ efekty, což jsou úniky mezi oddíly, které způsobují zpomalení aplikací. Unisys de facto každý oddíl pojímá jako softwarově definovaný blade s plnou izolací. s-Par umožňuje vytvořit i oddíl s obecným x86 systémem. V praxi tedy lze na ClearPath mainframu spustit i Linux či Windows Server. To je obdobný koncept jako u BullSequana. Na s-Par infrastruktuře pak běží jednotlivé operační systémy a úlohy.

Hardwarově ClearPath využívá standardní komponenty, ale doplňuje je specializovanými moduly pro akceleraci některých mainframe funkcí. Například dříve Unisys používal tzv. IOP (I/O Processor) na odkládání zátěže vstupně-výstupních operací. Nové modely s Xeonem dokázaly díky vyššímu výkonu CPU redukovat počet těchto I/O koprocesorů o 90 % a přesto dosáhnout až čtyřnásobné I/O propustnosti oproti starším CMOS strojům.

Celkově architektura poskytuje extrémně rychlé datové toky dovnitř i ven z mainframu – podporuje moderní rozhraní (Fiber Channel, 10/25/40Gb Ethernet, InfiniBand) a disková pole obvyklá v datacentrech. Unisys též zavedl koncept specializovaných partition (specialty engines), které slouží pro konkrétní účely mimo hlavní OS. Například **ClearPath ePortal** je integrovaný aplikační server ve vlastním oddílu, sloužící k obsluze webových a mobilních aplikací napojených na mainframe. Tyto pomocné „enginy“ rozšiřují schopnosti systému. Celá architektura ClearPath Forward tak představuje unikátní spojení mainframe vlastností (dostupnost, RAS, transakční výkon) s otevřeností x86 platformy [5, 43, 45, 48].

Operační systémy: Unisys mainframy historicky kombinují dvě odlišné architektury OS, pocházející z fúze Unisys (Sperry Univac a Burroughs):

- **OS 2200** – pokračovatel řady UNIVAC Exec 8/OS1100, určený pro systémy ClearPath Dorado. Jde o mainframe OS původně navržený pro 36bitovou architekturu (tzv. word machine, 36bit = 1 slovo), orientovaný na dávkové zpracování a transakční systémy ve velkých organizacích (finance, obrana apod.).
- **MCP (Master Control Program)** – operační systém z dílny Burroughs, využívaný na mainframech ClearPath Libra. Jedná se o jeden z prvních OS napsaných ve vyšším programovacím jazyce (ESPOL/NEWP), běžící na architektuře založené na zásobníkovém modelu.

Tyto dvě větve OS Unisys nadále rozvíjí paralelně, avšak od uživatelské perspektivy je spojuje jednotná platforma ClearPath Forward. Konkrétní model (Dorado vs Libra) určuje, který OS je použit pro zákazníkovy aplikace. Původní kód OS je provozován pomocí softwarové emulace. Unisys implementoval vrstvu, která na Intel hardwaru věrně reprodukuje instrukční sadu historických UNIVAC i Burroughs procesorů, takže OS2200 i MCP běží na Xeonu, aniž by si aplikace „uvědomily“ změnu platformy.

Unisys také vyvinul middleware, který umožňuje snadnější komunikaci mezi aplikacemi OS2200, MCP a externími systémy. Oba OS tak mohou podporovat moderní jazyky (Java, C, COBOL atd.), webové služby a podobně. Přímo v MCP i OS2200 je k dispozici Java VM, stejně tak lze z nich volat externí REST API služby. Unisys též uvolnil vývojové nástroje – např. ClearPath Visual IDE integruje vývoj mainframe aplikací do prostředí Visual Studio. Tím se smazává tradiční propast mezi mainframe vývojem a běžným IT [58, 81].

Z technologického a tržního rozboru vyplývá, že IBM Z Series dominuje současnému mainframe segmentu díky kontinuální inovaci (64bit architektura, specializované akcelerátory, integrace cloudu) a široké bázi nasazení v kritických systémech světa. Konkurenční mainframy existují spíše pro specifické regionální či aplikační potřeby a většinou se transformovaly k využití standardních CPU (x86) s emulací legacy prostředí. IBM Z si drží náskok v podpoře jak horizontální, tak vertikální škálovatelnosti, robustnosti a univerzálnosti nasazení. Organizace volí mainframy tam, kde potřebují kombinaci výkonu, spolehlivosti a bezpečnosti, jíž jiné platformy dosud zcela nedosáhly. Zároveň probíhá postupná evoluce – mainframy se více otevírají (Linux, cloud integration) a konkurence ubývá (ukončení Fujitsu mainframů). V dohledné budoucnosti tak mainframy zůstanou nadále klíčovou, byť specializovanou součástí IT ekosystému, zejména v odvětvích, kde je kontinuita provozu a dat prioritou.

Kapitola 4

Programovací jazyky a nástroje

Programovací jazyky používané na mainframech IBM Z tvoří vzájemně propojený ekosystém, ve kterém každý jazyk plní specifickou roli. COBOL slouží převážně k realizaci robustních podnikových aplikací a dávkového zpracování dat, zatímco JCL umožňuje řídit a plánovat spouštění těchto aplikací. Skriptovací jazyk REXX usnadňuje automatizaci rutinních operací, integraci systémových funkcí a administraci systému. V posledních letech se v prostředí IBM Z prosazuje i Python, který přináší flexibilitu a usnadňuje integraci tradičních aplikací s moderními technologiemi, jako jsou DevOps nástroje nebo datová analytika. Tato kapitola se zaměřuje na přehled těchto jazyků, jejich vzájemné vazby a ukazuje, jak společně přispívají k efektivnímu vývoji a správě aplikací na mainframech.

Při práci na IBM Z mainframech se nepoužívají klasické soubory a složky, proto je dobré uvést a popsat určitá slova nebo fráze, o kterých se dále bude psát.

- **Dataset** – Základní úložný prvek používaný na mainframech IBM Z, který představuje ekvivalent souboru na běžných operačních systémech. Datasety mohou mít různé formáty a struktury, jako sekvenční nebo indexované datasety (VSAM).
- **Sekvenční dataset** – Nejjednodušší typ datasetu, ve kterém jsou data ukládána a čtena postupně, od začátku do konce.
- **VSAM (Virtual Storage Access Method)** – Specifický typ indexovaného datasetu používaný pro rychlé vyhledávání a zpracování dat. VSAM datasety mají strukturovaný přístup k datům prostřednictvím indexů.
- **PDS (Partitioned Dataset)** – Speciální typ datasetu, který umožňuje uchovávat více položek (členů, members) pod jedním společným názvem datasetu. Každý member může být nezávisle upravován, kompilován nebo spouštěn, čímž PDS poskytuje obdobu složky s více soubory.
- **Member** – Jednotlivá položka nebo část uložená v rámci PDS nebo PDSE. Často představuje samostatný program, skript, zdrojový kód, konfiguraci či jiné specifické informace.

Datasety a jejich members mají specifický způsob zápisu a odkazování:

- **Dataset** se zapisuje ve formátu např. `USER01.TEST.DATA`, kde mezi tečkami nesmí být více jak 8 znaků a je běžné spíše používat písmena a čísla. První segment se

nazývá **HLQ** – high-level qualifier. Celkově jméno datasetu nesmí být delší jak 44 znaků. Typicky, když se v nějakém ze segmentů objeví JCL nebo COBOL, tak to znamená, že dataset obsahuje členy s kódy v těchto jazycích. Dále v kapitole bude zmíněna platforma IBM Z Xplore, kde se jako obecné HLQ používá „ZXP“ a při tvorbě uživatel používá jako svoje ID jako HLQ, např. `Z60930.JCL`

- **Member** se specifikuje za názvem datasetu v závorkách, např. `Z60930.JCL(TRXREP)`

Tradiční programovací jazyky

S IBM Z jsou tradičně spojeny jazyky jako COBOL, JCL nebo REXX. Existují i další podporované jazyky, ale v této práci se soustředím na tyto zmíněné i v rámci implementační části, proto je dobré je nejdříve rozebrat.

4.1 COBOL

COBOL je akronym z Common Business Oriented Language a patří mezi jedny z nejstarších vyšších programovacích jazyků, navržený primárně pro obchodní a databázové aplikace. Vznikl v roce 1959 pod záštitou komisí CODASYL a USA Department of Defense jako přenositelný jazyk pro zpracování dat. Při jeho návrhu se vycházelo z jazyka FLOW-MATIC (vyvinutého Grace Hopperovou) a kladl se důraz na srozumitelnost kódu v angličtině a schopnost běžet na různých hardwarových platformách té doby. První COBOL kompilátory se objevily počátkem 60. let. Postupně byl standardizován, a to poprvé v roce 1968 verzí (COBOL-68) a postupně doplňován, třeba v roce 2002 se přidala možnost objektově orientovaného programování. Nejnovější standard je COBOL 2023, kdy se přidaly asynchronní zprávy nebo příkazy jako COMMIT nebo ROLLBACK pro transakce. COBOL se přes svůj věk udržel díky zpětné kompatibilitě a obrovskému množství legacy kódu – dnešní programování v COBOLu je spíš orientované na úpravu těchto legacy kódů. Programy v jazyce COBOL se celosvětově používají vládami a podniky a běží na různých operačních systémech, jako jsou z/OS, z/VSE, VME, Unix, NonStop OS, OpenVMS a Windows. V roce 1997 Gartner Group uvedla, že 80 % světového byznysu běží na COBOLu s více než 200 miliardami řádků kódu. Modernizace stávajících systémů a strategický význam COBOLu vedl k výraznému nárůstu, bohužel přesná data nejdou přesně dohledat. Dnes celkové číslo tvoří více než 800 miliard, což z COBOLu dělá nejpoužívanější jazyk z hlediska počtu řádků nasazeného kódu [42].

Syntaxe a sémantika COBOLu

COBOL je kompilovaný imperativní procedurální jazyk s možnostmi OOP a strukturových paradigmat v novějších verzích, známý svou angličtině podobnou syntaxí. Program COBOLu je strukturován do čtyř hlavních oddílů: ***Identification Division, Environment Division, Data Division a Procedure Division***. Syntaxe je velmi blízká přirozenému jazyku – klíčová slova jsou anglická slovesa a fráze (např. `MOVE A TO B`, `IF ... ELSE ... END-IF`, `PERFORM UNTIL` atd.), což bylo zamýšleno pro snadné čtení i pro neprogramátory. Například podmínka může být zapsána jako `x IS GREATER THAN y`. Tato textová syntaxe je podpořena přes 300 klíčovými slovy. Datové typy COBOLu tradičně zahrnují čísla (často pevná desetinná čísla) a textové řetězce, reflektující zaměření na obchodní výpočty a zpracování textových záznamů. COBOL dále podporuje práci se soubory (sekvenční, indexované)

přímo v jazyce. Sémanticky COBOL umožňuje strukturované programování a je doporučeno nahrazovat dřívější hojné `GOTO` strukturami `IF`/`PERFORM`), nicméně zachovává i starší konstrukty kvůli zpětné kompatibilitě. COBOL je staticky typovaný a velmi přísný v kontrolách formátu dat, což přispívá k jeho spolehlivosti u finančních aplikací.

- **Identification Division**: Definuje metadata o programu, jako jsou název programu, autor, datum vytvoření atd.

  ```
  IDENTIFICATION DIVISION.
  PROGRAM-ID. HELLO-WORLD.
  ```

  Výše uvedená ukázka deklaruje program s názvem HELLO-WORLD.

- **Environment Division**: Specifikuje operační prostředí programu, zejména nastavení a přístup k souborům.

  ```
  ENVIRONMENT DIVISION.
  INPUT-OUTPUT SECTION.
  FILE-CONTROL.
      SELECT INFILE ASSIGN TO 'INPUT'
      SELECT OUTFILE ASSIGN TO 'OUTPUT'
  ```

  Ukázka definuje pro kompilátor COBOLu, že logický soubor `INFILE` bude vázán na fyzický soubor s názvem `INPUT`. `INFILE` se dále používá v COBOLu jako proměnná. `INPUT` je v JCL definovaný identifikátor. To stejné platí i pro `OUTFILE` a `OUTPUT`.

- **Data Division**: Obsahuje deklarace proměnných pomocí **PICTURE clause** určujících jejich datových typ a délku čísla či řetězce, včetně pracovních oblastí.

  ```
  DATA DIVISION.
  WORKING-STORAGE SECTION.
  01 STUDENT-NAME PIC X(20).
  01 STUDENT-AGE  PIC 9(2).
  ```

  V této ukázce jsou deklarovány proměnné STUDENT-NAME (řetězec délky 20 znaků) a STUDENT-AGE (číslo se dvěma ciframi).

  COBOL však umožňuje i strukturované zanoření proměnných pomocí úrovňových čísel (level-numbers), kde nižší číselná hodnota (např. 01) představuje vyšší úroveň a vyšší číselná hodnota (např. 05) označuje podřízenou část. Takto lze definovat například záznam o zaměstnanci:

  ```
  01 IN-REC.
     05 EMP-ID       PIC X(5).
     05 NAME         PIC X(19).
     05 DEPT         PIC X(15).
  ```

V tomto příkladu je `IN-REC` tzv. **group item** a jeho podřízené položky `EMP-ID`, `NAME` a `DEPT` jsou **elementary items**. Typicky se používají násobky pěti pro přehlednost. Tento způsob strukturování umožňuje pracovat jak s celým záznamem `IN-REC` jako s jedním blokem dat, tak i s jednotlivými jeho částmi. Při použití v programu můžete např. přesunout celý záznam mezi strukturami pomocí jediného příkazu.

```
MOVE IN-REC TO OUT-REC.
```

COBOL podporuje i tzv. **podmínková jména (condition names)**, která se deklarují pomocí úrovně `88`. Tato jména reprezentují konkrétní hodnoty jiné proměnné a slouží ke zlepšení čitelnosti podmínek v kódu.

```
01 GENDER        PIC X.
   88 MALE       VALUE 'M'.
   88 FEMALE     VALUE 'F'.
```

V tomto příkladu je `GENDER` standardní proměnná typu znak. Následující deklarace s úrovní `88` určují, že pokud má `GENDER` hodnotu `'M'`, pak je podmínka `MALE` pravdivá. Podobně, pokud má hodnotu `'F'`, pak platí `FEMALE`. Tyto podmínkové konstanty pak lze využít v programu místo přímého porovnávání hodnot.

```
IF MALE
    DISPLAY "Pohlaví: muž"
ELSE IF FEMALE
    DISPLAY "Pohlaví: žena".
```

COBOL podporuje různé datové typy:

- `PIC X(n)` – textový řetězec o délce `n`
- `PIC 9(n)` – celé číslo o `n` číslicích
- `PIC S9(n)V9(m)` – desetinné číslo se znaménkem, `n` číslic před desetinnou čárkou a `m` za ní

- **Procedure Division**: Zde jsou umístěny příkazy, které program provádí při spuštění.

```
PROCEDURE DIVISION.
    DISPLAY "Hello, World!".
    STOP RUN.
```

Uvedený příklad demonstruje jednoduché zobrazení textu „Hello, World!“ a ukončení programu.

V Procedure Division lze využívat například následující konstrukce:

- `MOVE` – přiřazení hodnoty do proměnné

- `IF` – podmíněné větvení
- `PERFORM` – volání procedury nebo opakování bloku kódu
- `EVALUATE` – vícenásobná podmínka (ekvivalent switch-case)
- `READ/WRITE` – práce se soubory

Využití COBOLu na IBM Z

Dnes na IBM Z běží obrovské množství COBOLových programů, které provádějí kritické dávkové úlohy a transakční zpracování. IBM neustále podporuje a rozvíjí COBOL na svých mainframech – nabízí moderní kompilátory IBM Enterprise COBOL for z/OS, které implementují nová rozšíření jazyka a optimalizace pro současný hardware. Navzdory občasným předpovědím útlumu vývoje v COBOLu je realita taková, že vývoj mainframe aplikací v COBOLu a obdobných jazycích pokračuje stálým tempem – po celém světě jsou v provozu tisíce COBOLových programů, jež tvoří páteř každodenního chodu firem, a firmy dokonce nadále vytvářejí nové aplikace nebo moduly v COBOLu.

Hlavní doménou COBOLu na IBM Z jsou:

- **Dávkové zpracování (batch)** – COBOL se využívá zejména pro noční a periodické úlohy zpracovávající velké objemy dat z transakcí, které se uskutečnily přes den. Zpracovávají se například zúčtování transakcí, výpočty mezd, generování sestav.
- **Online transakční zpracování** – COBOL programy zde fungují jako transakce vyvolané požadavkem, může jít o zaplacení rezervace hotelu nebo letenek, kde je nutná rychlá a spolehlivá odezva.
- **Databázové aplikace** – COBOL na mainframech často přistupuje k databázím – zejména k relační **DB2**. Většina robustních bankovních systémů nebo i již zmíněné rezervace letenek je kombinací COBOLové aplikační logiky a DB2 databáze.
- **Údržba legacy kódu** – Mnoho kódu napsaného v COBOLu v minulých desetiletích je stále v provozu. Úkolem současných vývojářů je tyto systémy udržovat a rozšiřovat. IBM proto poskytuje pokročilé nástroje, které usnadňují práci s rozsáhlými COBOLovými codebase.

COBOL se v prostředí z/OS spoléhá na Language Environment (LE), což je společné runtime prostředí pro jazyky na z/OS. Díky LE a dalším komponentám mohou COBOL programy koexistovat s moduly v jiných jazycích (PL/I, C, C++, assembler) v jedné aplikaci. COBOLový zdrojový kód bývá uložen v knihovnách (partitioned datasets – PDS) a jeho překlad do spustitelné podoby probíhá přes JCL kompilace a linky (viz kapitola o JCL). Celkově zůstává COBOL nepostradatelným nástrojem pro mainframe vývoj zejména díky své dlouhodobé přítomnosti v produkčních systémech, rozsáhlé základně ověřeného kódu a podpoře prostředí na platformě z/OS [17, 28, 59].

4.2 Job Control Language

Job Control Language (JCL) je specifický skriptovací jazyk používaný v IBM mainframe operačních systémech pro zadávání dávkových úloh a řízení jejich běhu. S JCL se poprvé setkáváme v 60. letech u operačního systému OS/360 pro sálové počítače System/360. Od té doby se JCL v jádru příliš nezměnil – moderní z/OS stále používá jeho strukturu

a syntaxi velmi podobnou té z dob OS/360. Na rozdíl od jazyků jako COBOL, které prošly mnoha standardizačními cykly a rozšířeními, si JCL zachovává konzervativní podobu kvůli extrémnímu důrazu na zpětnou kompatibilitu. Jakékoliv vylepšení syntaxe či funkcí JCL musí zachovat možnost provozu starých úloh beze změny, jinak by hrozilo, že kritické batch joby u zákazníků nepoběží. Tato kompatibilita vysvětluje určitou archaickou podobu JCL. IBM ve skutečnosti měla dvě odlišné varianty JCL: jednu pro systémovou větev DOS/360 (dnešní z/VSE) a druhou pro OS/360 a jeho následníky (MVS, z/OS). Spolu sdílejí jen základní syntaxi a pár konceptů, ale jinak jsou velice odlišné. Nás se týká JCL na z/OS, jelikož na IBM Z obvykle běží tento operační systém.

JCL je pro mainframe ekvivalent toho, co jsou shell skripty pro jiné systémy, avšak s důrazem na robustnost a kontrolu nad systémovými prostředky. Bez znalosti JCL se neobejde žádný mainframe programátor či administrátor, neboť právě JCL zajišťuje, že se jejich programy skutečně vykonají na z/OS a dostanou k potřebným datům [28] [73].

Syntaxe a sémantika JCL

JCL neobsahuje aritmetiku nebo komplexní logiku. Jde spíše o sadu kontrolních příkazů určených k popisu, kdy a jak spustit programy na mainframu, s jakými prostředky. Úlohou JCL je systému říci, které programy spustit, jaké vstupní a výstupní soubory (datasety) mají použít, případně definovat podmínky pro zpracování jednotlivých kroků. JCL tak slouží k sekvenčnímu provádění tzv. jobů (dávek), které se mohou skládat z více kroků – v každém kroku se spouští určitý program nebo procedura. Kromě spuštění vlastního aplikačního programu může JCL obstarat i vytvoření potřebných datasetů, nastavení výpisů (logů) na tiskárnu nebo spool, předání parametrů programům a další pomocné činnosti související se zpracováním úlohy.

Typický JCL stream (soubor) začíná příkazem `//JOB`, následuje jedna či více sekcí `//STEPn EXEC ...` a pod každou z nich příslušné `//DD ...` specifikace souborů. JCL má striktní syntaktická pravidla: každý příkaz začíná na novém řádku znaky `//` a nemůže se objevit prázdný řádek, návěští a názvy mají pevně omezenou délku (typicky 8 znaků) například začátek `//SALESJOB JOB` a parametry se zadávají ve formátu `KEYWORD=value`, oddělené čárkami. Mezery a pozice mají význam pro pokračovací řádky, rozlišování kolonek atd. Vše musí být psáno velkými písmeny.

Přestože úplná syntaxe JCL je rozsáhlá (existuje mnoho parametrů pro alokaci diskového prostoru, třídění úloh, priority, podmínky atd.), v praxi se využívá poměrně malá podmnožina nejdůležitějších konstrukcí. Téměř 90 % běžných úloh lze pokrýt několika základními JCL příkazy a parametry (`JOB`, `EXEC`, `DD`, jednoduché `COND`, definice prostorů `SPACE`, `DSN` jména datasetů apod.). Z hlediska sémantiky JCL „programuje“ spíše operační systém než CPU: při odeslání JCL úlohy systém zpracuje tyto příkazy a podle nich spustí odpovídající programy s patřičně připojenými daty. Výsledkem JCL tedy není nový výstupní datový element, ale provedený job.

V minulé kapitole byl popsán Job Entry Subsystem, který spravuje fronty jobů, plánuje jejich běh a provádí vlastní interpretaci JCL. JES tak například na základě JCL zařadí job do fronty, alokuje datové sady před spuštěním programu, po skončení uvolní zdroje a zaloguje výstupy.

Existují i nadstavby JCL:

- **Procedury (PROCs)** – předdefinované šablony JCL (makra obsahující sekvence `EXEC` a `DD`), které lze volat jedním řádkem v JCL (pomocí `EXEC PROC=`).

- **Symbolická parametrizace** – umožňuje používat proměnné, jejichž hodnoty se doplní při spuštění (umožňuje znovupoužitelnost jednoho JCL pro více podobných úloh).

Nicméně JCL záměrně nemá podmíněné větvení (kromě omezeného `COND` na základě návratových kódů) ani cykly – veškerá aplikační logika by měla být v programech, JCL slouží pouze k jejich orchestraci.

Základní stavební kameny JCL jsou tři typy příkazů:

- **JOB** – Označuje začátek dávkové úlohy. Každý job má právě jeden příkaz `//JOB`, kterým je určen jeho název a případně globální parametry nebo účetní informace.

  ```
  //MYJOB JOB (ACCNT),'JOB DESCRIPTION',
  //       CLASS=A,MSGCLASS=X,NOTIFY=USER
  ```

  Výše uvedená ukázka definuje job s názvem `MYJOB`, účetní informace `ACCNT`, třídu `A`, výstupní zprávy směřuje do třídy `X` a oznámení po skončení jobu zašle uživateli `USER`.

- **EXEC** – Definuje jednotlivý krok jobu, tedy spuštění konkrétního programu nebo procedury. Každý `//EXEC` obsahuje parametr `PGM` (název spustitelného programu) nebo `PROC` (název katalogizované procedury. Použití **DD** je vysvětleno později.

  Příklad spuštění programu `IEBGENER`, který kopíruje data mezi datasety:

  ```
  //STEP1 EXEC PGM=IEBGENER
  //SYSUT1 DD DSN=INPUT.DATASET,DISP=SHR
  //SYSUT2 DD DSN=OUTPUT.DATASET,
  //          DISP=(NEW,CATLG,DELETE),
  //          SPACE=(TRK,(10,5)),
  //          DCB=(RECFM=FB,LRECL=80)
  //SYSPRINT DD SYSOUT=*
  //SYSIN DD DUMMY
  ```

  Tato ukázka kopíruje obsah `INPUT.DATASET` do nového datasetu `OUTPUT.DATASET` s definovaným alokovaným prostorem v tracks (stopách – jednotkách místa na disku).

  Příklad použití programu `IDCAMS`, který umožňuje manipulaci s VSAM datasety:

  ```
  //STEP2 EXEC PGM=IDCAMS
  //SYSPRINT DD SYSOUT=*
  //SYSIN DD *
     DELETE (MY.VSAM.DATASET) CLUSTER PURGE
     DEFINE CLUSTER(NAME(MY.VSAM.DATASET) -
            TRACKS(15 5) RECSZ(80 80) KEYS(8 0) -
            INDEXED)
  /*
  ```

  Výše uvedený příklad nejprve odstraní existující VSAM dataset (pokud existuje) a následně vytvoří nový indexovaný VSAM dataset s definovanou velikostí klíče a záznamu.

Příklad kompilace COBOL programu s procedurou `IGYWCL`:

```
//COMP EXEC IGYWCL
//COBOL.SYSIN   DD DSN=USER.CBL(SALESREP),DISP=SHR
//LKED.SYSLMOD  DD DSN=USER.LOAD(SALESREP),DISP=SHR
```

Tento příklad kompiluje COBOL zdrojový kód `SALESREP` uložený v knihovně `CBL` a vytváří spustitelný modul v knihovně `LOAD`.

Rozšířenější ukázka s procedurou:

```
//STEP2 EXEC PROC=MYPROC,PARM='OPTIONS'
```

Zde se spouští katalogizovaná procedura `MYPROC` s předaným parametrem `'OPTIONS'`.

- **DD (Data Definition)** – Specifikuje vstupní a výstupní datové sady pro daný program. Každý příkaz `//DD` popisuje datový zdroj nebo cíl, se kterým program pracuje.

```
//OUTPUT DD DSN=MY.DATASET.NAME,
//          DISP=(NEW,CATLG,DELETE),
//          SPACE=(CYL,(5,1)),
//          DCB=(RECFM=FB,LRECL=80)
```

Výše uvedená ukázka definuje vytvoření nového datasetu `MY.DATASET.NAME`, který se uloží (`CATLG`) po úspěšném dokončení kroku, nebo odstraní (`DELETE`) při neúspěchu. Datový prostor je alokován v cylinderech (5 primárních, 1 sekundární), formát dat je pevný (FB) a délka jednoho záznamu je 80 bytů.

Další možnosti alokace prostoru:

  - `SPACE=(TRK,(primární,sekundární))` – alokace prostoru ve stopách (tracks)
  - `SPACE=(CYL,(primární,sekundární))` – alokace prostoru v cylindrech (soubor soustředných stop)
  - `SPACE=(BLK,(primární,sekundární))` – alokace prostoru v blocích (logické jednotky dat definované aplikací)

Další příklad vstupního datasetu:

```
//INPUT DD DSN=EXISTING.DATASET,
//         DISP=SHR
```

Tato ukázka definuje existující dataset `EXISTING.DATASET`, který bude sdíleně otevřen pro čtení.

Využití JCL na IBM Z

Na mainframech IBM Z je JCL nezbytným nástrojem pro provoz dávkových úloh. Prakticky každá batch práce, ať už jde o produkční zpracování nebo vývojářské úlohy, je definována pomocí JCL. Některé hlavní scénáře užití:

- **Spouštění dávkových aplikací** – V produkčním provozu bank, pojišťoven, velkých podniků běží denně stovky až tisíce jobů, které zajišťují zpracování obchodních dat. Tyto joby jsou definovány JCL skripty a řízeny plánovači úloh (scheduling systémy jako IBM Tivoli Workload Scheduler apod.). JCL určuje, jaké programy (typicky COBOL, PL/I) se mají postupně spustit, jaké soubory mají zpracovat a kam směřovat výstupy (reporty na tisk, update databází, generování nových souborů).
- **Kompilace a deployment** – JCL se hojně využívá pro překlad a linkování programu. IBM dodává vzorové JCL pro kompilace – tzv. procedury pro kompilaci, které jsou předdefinovány (např. IGYWCL pro COBOL kompilaci atd.). Vývojář tedy spustí job, který obsahuje krok EXEC volající COBOL kompilátor nad zdrojovým členem a další krok pro linkedit, čímž vznikne spustitelný load modul. Tímto způsobem se na mainframu provádí build aplikací. Také testy se mohou řídit JCL, např. spustit testovací variantu programu s testovacím vstupem definovaným v DD SYSIN.
- **Systémové úlohy a utility** – V rámci JCL můžeme použít mnoho utilit pro systémové operace. Například zálohy datasetů (utility IEBGENER, IEBCOPY), manipulace s VSAM soubory (IDCAMS), spouštění DB2 utilit (Image Copy, Reorg).
- **Propojení s ostatními platformami** – V moderním prostředí se JCL může použít za pomoci nástrojů jako Zowe CLI. Python skript běžící na mainframu (v Unixovém prostředí) může pomocí funkcí z externí knihovny odeslat JCL job pro vykonání nějaké úlohy s přístupem ke zdrojům z/OS. JCL tak zůstává integrální součástí i v hybridních workflows.

4.3 REXX

REXX (REstructured eXtended eXecutor) je vyšší procedurální skriptovací jazyk vyvinutý v IBM na přelomu 70. a 80. let. Cílem bylo vytvořit snadno použitelný jazyk pro psaní maker, skriptů a vývoje aplikací, který by sjednotil a nahradil jazyky na mainframech jako EXEC a EXEC 2. REXX byl součástí skoro všech operačních systémů na IBM mainframech, ať už VM/CMS nebo MVS TSO/E (Time Sharing Option Extended na MVS). Na konci 80. let jej IBM zařadila jako běžný skriptovací jazyk do své standardní architektury SAA (Systems Application Architecture) – v té době byl označen jako „SAA Procedure Language REXX“. První překladač pro REXX byl vyvinut už v roce 1987. Umožňuje přeložit REXX skript do binární podoby pro rychlejší provádění – to je výhodné pro produkční nasazení rozsáhlejších REXX aplikací. V praxi se však REXX skripty téměř výhradně spouštějí interpretovaně, bez předešlé kompilace.

Přestože jsou JCL a REXX oba označovány jako skriptovací jazyky, jejich účel i způsob použití se zásadně liší. K čemu se používá JCL už bylo popsáno. Naproti tomu REXX je plnohodnotný skriptovací jazyk s podporou proměnných, podmínek atd. Díky tomu je vhodný pro interaktivní nástroje, prototypování, automatizaci a zpracování výstupů. Zatímco JCL „řídí běh úloh“, REXX umožňuje „programovat logiku jejich zpracování“. V praxi se REXX

často používá k přípravě, validaci nebo dynamickému generování JCL skriptů, nebo k automatizaci správy systémových úloh.

Syntaxe REXXu

REXX je **procedurální interpretovaný jazyk**, navržený s důrazem na čitelnost a jednoduchost struktury kódu. Skripty se nazývají REXX exec a vykonává je přímo interpret, není nutné jej předem kompilovat do strojového kódu. To umožňuje rychlý vývoj a testování – uživatel může interaktivně spustit skript, opravit jej a znovu spustit bez dlouhého cyklu překladu. IBM však umožňuje i volitelnou kompilaci REXXu (IBM Compiler for REXX on System z), která převede REXX do tzv. compiled REXX objektu, jenž se pak může linkovat a spouštět jako program – to se provádí kvůli výkonu u často používaných skriptů. Nicméně kompilace není pro funkčnost nutná.

Syntaxe REXXu je relativně blízká pseudokódu – používá známá klíčová slova pro řízení toku, nemá mnoho speciálních symbolů ani složitých konstrukcí. Příkazy mohou být psány volně (není třeba dodržovat pevné sloupce jako u JCL), zpravidla jeden příkaz na řádek. Struktura programu REXX je jednoduchá. Poskytuje konvenční výběr řídicích konstrukcí. Patří mezi ně například `IF...THEN...ELSE...` pro jednoduché podmínky, `SELECT...WHEN ...OTHERWISE...END` pro výběr z řady alternativ a několik variant `DO...END` pro cykly. Instrukce `GOTO` není obsažena, ale instrukce `SIGNAL` je k dispozici pro změnu toku programu, jako jsou chybové výstupy (signály) a větvení výpočtu.

Příklad jednoduchého řízení toku pomocí `IF` konstrukce:

```
IF number = 5 THEN
    SAY 'Number is five'
ELSE
    SAY 'Number is not five'
```

Proměnné není nutné v REXXu deklarovat. Všechny proměnné jsou automaticky typu string a jejich hodnoty se interpretují podle kontextu – dynamické typování. Operátory a vestavěné funkce poskytují bohaté možnosti pro manipulaci s textem a čísly. REXX také podporuje aritmetiku s velkou přesností.

Příklad práce s textovými řetězci:

```
text = 'Hello World'
SAY SUBSTR(text,1,5)  /* Vypíše 'Hello' */
```

Příklad jednoduché aritmetiky:

```
a = 10
b = 20
c = a + b
SAY 'Výsledek je:' c  /* Vypíše 'Výsledek je: 30' */
```

Řízení běhu je strukturované – lze využít cykly jako `DO...END`, iterace s čítačem, nebo podmíněné cykly.

Příklad cyklu s čítačem:

```
DO i = 1 TO 5
    SAY 'Číslo je:' i
END
```

Příklad podmíněného cyklu:

```
i = 1
DO WHILE i <= 3
    SAY 'Iterace číslo:' i
    i = i + 1
END
```

K dispozici jsou jednoduché proměnné a dá se definovat konstrukce složené proměnné, která podporuje přidávání polí (tzv. ocásků) k proměnné (v tomto kontextu nazývané kmen) a podporuje tak datové struktury, jako jsou seznamy, pole, n-rozměrná pole, řídká nebo hustá pole, vyvážené nebo nevyvážené stromy a záznamy.

Příklad použití asociativního pole:

```
X.1 = 'Hodnota 1'
X.2 = 'Hodnota 2'
DO i = 1 TO 2
    SAY X.i
END
```

Všechny proměnné jsou globální, ale lze vytvářet i lokální pomocí procedur a také používat návratové kódy a podprogramy (internal subroutines).

Příklad definice a volání interní procedury:

```
CALL Pozdrav
EXIT

Pozdrav:
    SAY 'Dobrý den!'
RETURN
```

Komentáře v REXXu začínají dvojicí znaků `/*` a končí `*/`. Komentovat lze také pomocí klíčové sekvence `--`, která označuje komentář do konce řádku.

Příklad komentářů:

```
/* Toto je komentář na více řádků,
   který REXX interpretuje jako jeden celistvý */
SAY 'Hello'  -- Tento komentář platí do konce řádku
```

REXX také umožňuje práci se vstupy a výstupy prostřednictvím příkazů `PULL`, `SAY`, nebo `PARSE`:

Příklad práce s uživatelským vstupem:

```
SAY 'Zadejte své jméno:'
PULL name
SAY 'Vaše jméno je:' name
```

REXX umožňuje snadnou integraci s TSO (Time Sharing Option) příkazy na mainframech:

Příklad použití TSO příkazu v REXXu:

```
ADDRESS TSO "LISTCAT LEVEL('USER01')"
```

Tento příklad vypíše seznam datasetů začínajících prefixem 'USER01'.

Využití REXXu na IBM Z

REXX se na mainframech IBM Z uplatňuje v mnoha rolích, zejména jako skriptovací a automatizační jazyk v TSO/ISPF prostředí:

- **Provádění rutinních úkolů** – například sekvence TSO/E příkazů. Administrátor místo ručního zadávání příkazů (alloc, copy, delete datasetů) napíše REXX skript, který vše vykoná automaticky. Podobně vývojář může mít REXX, který mu zkompiluje a spustí testy jedním příkazem.
- **Volání jiných REXX execů** – REXX může sloužit jako wrapper volající další REXX skripty, čímž lze modularizovat funkce.
- **Spouštění aplikací v jiných jazycích** – z REXXu lze spouštět i programy v assembleru, COBOLu atd. Tím REXX řídí a koordinuje běh jiných komponent. Například může připravit data, zavolat COBOL program na zpracování a pak zpracovat výsledek.
- **Tvorba ISPF aplikací** – REXX je hojně využíván k vytváření interaktivních nástrojů v ISPF (Interactive System Productivity Facility). Umožňuje definovat panely (obrazovky), menu, číst vstupy od uživatele, zobrazovat tabulky dat, a reagovat na uživatelské volby.
- **Jednorázové rychlé skripty** – REXX je ideální pro ad hoc řešení problému. Pokud potřebujete například procházet logy jobů a najít v nich určité údaje, nebo převést výstup z jednoho formátu do jiného, REXX lze napsat a spustit velmi rychle. Mnoho mainframe specialistů má sbírku drobných REXXů pro všemožné úlohy (hledání datasetů, porovnávání členů, generování JCL z šablony apod.).

V praxi se REXX stal neocenitelným pomocníkem systémových programátorů a operátorů na mainframe. Při monitorování systému lze REXXem naprogramovat reakce na události (v prostředí z/OS existuje i subsystém REXX Batch, a některé produkty jako IBM Tivoli System Automation umožňují psát automatizační skripty v REXXu). Vývojáři používají REXX, aby si usnadnili práci (např. napsání REXXu, který provede sekvenci testovacích jobů a zkontroluje jejich output). Program REXX zkompilovaný pod z/OS lze spustit pod z/VM. Podobně může program REXX zkompilovaný pod z/VM běžet pod z/OS. Program REXX zkompilovaný pod z/OS nebo z/VM může běžet pod z/VSE, pokud je nainstalován REXX/VSE. Existuje i varianta podporující OOP – Object REXX. Spousta programátorů může místo REXXu pro potřeby skriptování využít podporovaný Python v rámci „moderního“ přístupu [28, 79].

4.4 Python

Python nevznikl v mainframe prostředí, jako jiné zmíněné jazyky. Autorem Pythonu je Guido van Rossum, který jej poprvé vydal v roce 1991 jako následníka jazyka ABC. Python se od počátku zaměřoval na srozumitelnost kódu a jednoduchost syntaxe. Python vzrostl na popularitě při vydání verze 2.0 v roce 2000 v širokém spektru oblastí (webové aplikace, vědecké výpočty, automatizace, skriptování). Následovaly další verze jako Python 3.0 (2008), Python 3.12.0 (2023) a nejnovější verze 3.13 (2024), verze 3.14 je nyní ve vývoji. Dnes je Python jedním z nejpopulárnějších jazyků vůbec a de facto standardem pro data science, umělou inteligenci a automatizační skripty.

Python na IBM Z

Na mainframech IBM Z však Python nemá dlouhou tradici – historicky se aplikační vývoj na mainframech zpravidla odehrával v COBOLu a pro skriptování sloužil REXX. Dlouho také nebylo snadné Python na z/OS vůbec spustit. IBM od počátku roku 2000 podporuje běh Linuxových distribucí na již zmíněném Linux on Z, takže zákazníci mohli provozovat Python na mainframe hardwaru. Výsledkem iniciativy IBM poskytnout plnou podporu pro Python na IBM Z je, že dnes existuje oficiální distribuce Pythonu pro z/OS od IBM, nazvaná IBM Open Enterprise SDK for Python. Součástí SDK je interpret a kompilátor Pythonu, který podporuje provoz z/OS aplikací napsaných v Pythonu. Nejnovější IBM Open Enterprise SDK for Python podporuje verzi 3.13.

Na z/OS je jedním z odlišných aspektů zpracování znakové sady – mainframe tradičně používá EBCDIC[1] [20], zatímco většina moderního softwaru, včetně Pythonu, používá UTF-8 kódování. IBM však tuto problematiku vyřešila pomocí kodeků, které jsou součástí Python SDK a podporují převody mezi EBCDIC, ASCII nebo UTF-8, takže Python skripty mohou pracovat s texty v EBCDIC (datasety z/OS). Pro programátora to znamená, že při otevření souboru může použít správné kódování a Python se postará o zbytek. Další specialitou jsou knihovny pro práci se z/OS prostředím, jinak ale samotný jazyk Python funguje na mainframu shodně jako na jiných platformách.

Syntaxe a sémantika Pythonu

Python je interpretovaný, interaktivní objektově orientovaný jazyk s dynamickým typováním. Je známý svou syntaktickou vlastností – používá odsazení (indentaci) bloků kódu místo složených závorek nebo klíčových slov pro ukončení bloků, doporučené odsazení jsou 4 mezery. Tento design podporuje čitelnost kódu a donucuje programátory psát přehledně strukturovaný kód. Říká se, že „číst Python je jako číst angličtinu“. Python obsahuje bohatou sadu datových typů jako čísla různých přesností, řetězce, seznamy, slovníky, množiny, objekty atd. a disponuje rozsáhlou standardní knihovnou modulů, která pokrývá všechno od práce se soubory, přes síťovou komunikaci, až po webové služby nebo podporu trénování umělé inteligence. Existují standardní výrazy pro podmínky `IF...ELSE...ELIF`, cykly `FOR` nebo `WHILE`, `TRY...EXPECT` bloky pro práci s výjimkami. Funkce se definují klíčovým slovem `DEF`, příkazy pro přidání potřebných knihoven a modulů jsou `FROM` a následně `IMPORT`. Díky tomu programátoři mohou hodně funkcionality využít bez psaní vlastního kódu. Sémanticky je Python dynamicky typovaný (typy hodnot se určují až za běhu, proměnná může měnit typ), spravuje paměť automaticky (garbage collector), podporuje vícenásobnou dědičnost v OOP nebo funkcionální prvky (lambda, map/filter).

```
def pozdrav(text):
    print(f"Ahoj, {text}")

pozdrav("světe")
```

[1] EBCDIC (Extended Binary Coded Decimal Interchange Code) je osmibitová znaková sada vyvinutá firmou IBM pro sálové počítače, na základě kódu používaného pro děrné štítky. Na rozdíl od běžně používané ASCII nebo UTF-8 má jiné číselné kódování znaků, což může způsobit problémy při přenosu nebo zpracování textu mezi různými platformami. EBCDIC je nadále používán v systému z/OS, zejména v tradičních datasetech a výstupech aplikací.

Tato ukázka je jasná i neprogramátorovi na první pohled. Definuje se funkce, která při zavolání vypíše pozdrav.

Na z/OS jsou dostupné speciální moduly a knihovny, které usnadňují práci s mainframe prostředím, především moduly z knihovny ZOAU (Z Open Automation Utilities).

Příklad pro práci s datasety:

```
from zoautil_py import datasets

# Kontrola existence datasetu
if datasets.exists("IBMUSER.TEST.DATA"):
    print("Dataset existuje.")
else:
    print("Dataset neexistuje.")

# Vytvoření datasetu typu PDS
datasets.create("IBMUSER.TEST.PDS", dataset_type="PDS")

# Zápis do memberu datasetu
datasets.write("IBMUSER.TEST.PDS(MEMBER1)",
               content="První řádek v PDS memberu.")

# Čtení obsahu memberu datasetu
obsah = datasets.read("IBMUSER.TEST.PDS(MEMBER1)")
print(obsah)
```

Podobně lze na mainframu manipulovat s joby prostřednictvím JES (Job Entry Subsystem):

```
from zoautil_py import jobs

# Odeslání JCL jobu
job = jobs.submit("/u/ibmuser/jobs/testjob.jcl", is_unix=True)
print(f"Job odeslán s ID: {job.job_id}")

# Zjištění stavu jobu
stav_jobu = jobs.fetch(job.job_id)
print(f"Stav jobu: {stav_jobu.status}")

# Výpis výstupu jobu
output_jobu = jobs.read_output(job.job_id)
print(output_jobu)
```

Důležitá je také možnost zachytit chyby, které vzniknou během práce s mainframem pomocí specifických výjimek ZOAU:

```
from zoautil_py import jobs
from zoautil_py.exceptions import JobSubmitException

try:
```

```
    job_id = jobs.submit_return_job_id("IBMUSER.TEST.JOB", is_unix=False)
    print(f"Job úspěšně odeslán s ID: {job_id}")
except JobSubmitException as e:
    print(f"Chyba při odesílání jobu: {e.response.stderr_response}")
```

Tyto ukázky ilustrují, jak Python na mainframe kombinuje přehlednou syntaxi a moderní programovací praktiky s možnostmi prostředí z/OS.

Využití Pythonu na IBM Z

- **Automatizace a skriptování operací** – Python poskytuje způsob, jak skriptovat mainframe operace, podobně jako REXX, ale s výhodou obrovského ekosystému knihoven a hlavně znalostí Pythonu širokým spektrem programátorů. Pomocí knihovny zoautil_py lze v Python skriptu např. otevřít dataset, číst z něj záznamy, zapisovat do něj, spouštět JCL joby, dotazovat se na RACF (bezpečnostní systém) atd. To vše dovoluje administrátorům a devops inženýrům psát skripty v Pythonu namísto REXXu, a spravovat z/OS podobně jako by spravovali Linux.
- **Modernizace a integrace aplikací** – Python lze použít k rychlému vytvoření API rozhraní či wrapperu kolem stávajících mainframe aplikací. IBM to propaguje jako způsob, jak rozšiřovat business-critical aplikace pro rychlejší dodání nových funkcionalit. Python se tak může stát „lepidlem“ mezi tradičním mainframem a moderními aplikacemi.
- **Datová analytika a AI přímo na mainframu** – S nárůstem významu dat se objevuje potřeba analyzovat data tam, kde leží. Mnoho kritických dat leží na mainframech. Místo pracného přesunu dat na externí servery mohou organizace využít velký výkon IBM Z a Python přímo na z/OS pro datovou analýzu. IBM nabízí Python AI Toolkit for z/OS, který přináší populární knihovny na mainframe. To umožňuje provozovat strojové učení a AI algoritmy přímo v blízkosti mainframe dat, bez nutnosti data exportovat.
- **Vývoj nových funkcionalit** – Ačkoli kritické transakční jádro zůstává často v COBOLu, Python lze využít pro tvorbu doplňkových aplikací na mainframu. Python je dostatečně univerzální pro mnoho typů úloh a pokud nepotřebujeme extrémní optimalizaci, vývoj v Pythonu je rychlejší a levnější. IBM ostatně uvádí, že Python na z/OS umožňuje tvořit nové aplikace rychleji s menším množstvím kódu a využít přitom bohatý ekosystém balíčků.

Závěrem, Python zcela nenahrazuje například COBOL v jeho doméně (transakční zpracování), ale rozšiřuje možnosti mainframe platformy směrem k současným trendům (cloud, AI, DevOps). IBM tuto konvergenci podporuje – její produkt IBM Open Enterprise SDK for Python je zdarma pro držitele z/OS. Kombinace spolehlivosti mainframe a univerzálností Pythonu tak umožňuje podnikům urychlit digitální transformaci i v prostředí s legacy systémy [21] [78].

4.5 Vývojové nástroje

Tradiční způsob vývoje na mainframu probíhal a stále probíhá přímo na terminálu (3270) v editoru ISPF. Terminály řady 3270 byly představeny firmou IBM již v roce 1971 jako

součást interaktivního přístupu k sálovým počítačům. Šlo o tzv. *block-oriented* zařízení, která posílají a přijímají data po blocích místo po znacích, čímž výrazně snižují zátěž procesoru a komunikační linky. I když dnes už fyzické terminály téměř vymizely, emulátory 3270 zůstávají základním nástrojem pro přístup k mainframu.

ISPF a terminálový přístup (3270)

V z/OS umožňuje facility známá jako Time Sharing Option/Extensions neboli TSO přihlášení více uživatelů a interaktivní sdílení prostředků mainframu. TSO také poskytuje uživatelům omezenou sadu základních příkazů. Používání této sady se někdy nazývá používání TSO v nativním režimu. Prostředí ISPF se spouští z nativního prostředí TSO. V rozsahu povoleném různými bezpečnostními kontrolami má uživatel ISPF plný přístup k většině funkcí systému z/OS.

Interaktivní prostředí ISPF (Interactive System Productivity Facility) představuje historicky nejrozšířenější rozhraní pro vývoj a správu systémů na mainframe platformě. ISPF se používá prostřednictvím terminálů typu 3270, které jsou charakteristické svým jednoduchým, textovým zobrazením, běžně označovaným jako „zelená obrazovka“.

V prostředí ISPF mají vývojáři přístup k operačnímu systému z/OS, mohou přímo editovat members se zdrojovými kódy, spravovat datasety a spouštět úlohy pomocí JCL.

ISPF editor disponuje základními editačními funkcemi, přičemž uživatelé interagují s textem přímo na obrazovce pomocí klávesnice, jednoduchých navigačních příkazů a klávesových zkratek. Existují dnes i ISPF s uživatelským grafickým rozhraním, která umožňují použití myši při navigaci.

Přestože ISPF může působit zastarale ve srovnání s moderními grafickými či webovými rozhraními, stále představuje efektivní a často používaný nástroj v praxi, zejména v oblastech, kde dosud nedošlo k plné modernizaci infrastruktury. Pro zkušené vývojáře a systémové administrátory je ISPF ceněno především díky své rychlosti, přehlednosti a přímé interakci s mainframovým systémem [28].

Tlak na produktivitu a integraci s moderními procesy však vedl k vzniku celé řady nástrojů, které mainframe vývoj výrazně zlepšují a přibližují standardům známým z vývoje na jiných platformách, zvláště pro současnou generaci programátorů. Níže jsou popsány klíčové z nich:

Zowe

Zowe je otevřený framework pro z/OS, který vznikl jako projekt Open Mainframe Project. Klade si za cíl poskytnout moderní rozhraní pro práci s mainframem a umožňuje vývojářům a administrátorům interagovat se systémem z/OS způsoby, které jsou běžné mimo mainframe – přes webové aplikace, REST API a příkazovou řádku – aniž by museli používat tradiční 3270 terminál. Zowe je vlastně soubor několika projektů, kdy se každý soustředí na jiný aspekt, takže si uživatel může vybrat, co mu nejvíce sedí a jaké funkce potřebuje. Zde je seznam a popis nejpoužívanějších:

- **Zowe Application Framework (Web UI)** – Jde o webové rozhraní, které poskytuje webovou virtuální plochu s aplikacemi pro přístup k z/OS funkcionalitám. V základní instalaci Zowe jsou webové aplikace například: 3270 Terminal (emulátor terminálu v prohlížeči), VTAM Terminal, také editor a prohlížeč pro práci s JES, prohlížení

datasetů a USS terminál[2]. Uživatel se přes prohlížeč přihlásí do Zowe a má grafické prostředí, odkud může provádět běžné úlohy správy a vývoje. Tyto funkcionality jsou zajištěny díky REST API. Rozhraní je navíc rozšiřitelné – komunita či společnosti mohou vyvíjet další pluginy nebo aplikace do Zowe plochy pro specifické účely. To je velký krok kupředu – vývojář nebo správce už nepotřebuje zvláštní terminálový emulátor, stačí mu webový prohlížeč.

- **Zowe API Mediation Layer (API brána)** – Tato vrstva poskytuje jednotný přístup k různým REST API službám na z/OS. Funguje jako API Gateway – umožňuje registrovat různé z/OS služby (např. API pro JES, data sety) a zpřístupnit je bezpečně přes jednotné endpointy. Zowe v základu nabízí některé API (pro dataset operace, joby, systémové informace). Díky API katalogu pak vývojáři mohou objevovat a volat mainframe API snadno díky dodané dokumentaci. API Mediation Layer řeší autentizaci, routing, verze API atd. V praxi to znamená, že firma může svým interním aplikacím poskytnout REST API pro odeslání jobu do JES, a to skrz Zowe. Tato komponenta výrazně otevírá mainframe k integraci – umožňuje psát nové front-end aplikace, které volají z/OS funkcionalitu přes standardní protokoly.

- **Zowe CLI (Command Line Interface)** – Jde o nástroj příkazové řádky, který lze nainstalovat na běžný počítač a který umožňuje ovládat mainframe z terminálu pomocí jednoduchých příkazů. CLI komunikuje s mainframem právě přes zmíněná API. Zowe CLI příkazy pokrývají akce jako: `zowe zos-files` (práce s datasety – zobrazení, editace, vytvoření atd.), `zowe zos-jobs` (odeslání JCL jobu, sledování stavu, stažení výstupu, atd.), umožňuje také interakci s TSO pomocí `zowe zos-tso`. Všechny dostupné příkazy jsou přehledně zdokumentované. Zowe CLI tímto způsobem umožňuje zařadit mainframe do skriptů a automatizace velmi jednoduše. Např. v CI pipeline může být krok, který pomocí Zowe CLI nahraje nové zdrojové kódy do knihovny na z/OS nebo spustí testovací job a stáhne výsledky. Z pohledu vývojáře, který nechce používat 3270, je Zowe CLI skvělý nástroj, kde může využít své znalosti skriptování a volat mainframe služby.

- **Zowe Explorer** – rozšíření do Visual Studio Code. Tento projekt bude popsán detailně dále v textu v souvislosti s IBM Z Open Editorem

Podstatné je, že Zowe je open-source a zdarma, s širokou podporou komunity. Zowe je koncipováno jako platforma, kde vývojáři mohou přispívat k rozšíření jak pro CLI, web UI, tak pro API layer. Z hlediska přínosu Zowe zmenšuje závislost na specifických znalostech, jako je práce s terminálem a ISPF – nová generace vývojářů může využít REST API a CLI, což jsou technologie, které již znají. Problém nastává, pokud je potřeba pracovat na distribuci mainframu, kde není podpora Zowe [38].

Visual Studio Code a rozšíření

Visual Studio Code (VS Code) je populární editor zdrojových kódů od Microsoftu, hojně využívaný napříč vývojářskými komunitami. Sám o sobě je to modulární editor, který lze rozšířit o podporu různých jazyků a funkcí pomocí rozšíření (extensions). Pro práci na

[2]USS (UNIX System Services) je komponenta operačního systému z/OS, která poskytuje prostředí kompatibilní s POSIX a umožňuje provoz unixových aplikací na mainframu. Uživatelé mají přístup k shellu, souborovému systému a standardním nástrojům známým z UNIX/Linux prostředí. USS tvoří základ pro běh moderních nástrojů, jako je Zowe nebo jazyk Python.

mainframe byla komunitou i IBM vytvořena sada rozšíření, která z VS Code dělá použitelné vývojové prostředí pro práci s mainframe kódem. Nejvýznamnější z nich je IBM Z Open Editor.

IBM Z Open Editor

IBM Z Open Editor je bezplatné rozšíření pro VS Code poskytující podporu pro jazyky z/OS – COBOL, PL/I, HLASM, JCL a REXX. Vývojář může v prostředí VS Code na svém PC otevírat mainframe zdrojové kódy. Z Open Editor nabízí například zvýraznění syntaxe, kontrolu správného formátu ve sloupcích (pro COBOL), automatické doplňování klíčových slov, napovídání názvů proměnných, refaktorizaci nebo formátování dokumentů. Podobně pro JCL zvýrazní klíčová slova (//JOB, //EXEC, //DD) a upozorní na chyby v parametrech. Tyto funkce jsou umožněny díky Language Server Protocol, který provádí syntaktickou a sémantickou analýzu.

Z Open Editor je tak určen vývojářům, kteří preferují moderní vývojové prostředí a chtějí psát mainframe kód ve stejném editoru, v jakém třeba píšou i kód v JavaScriptu či Pythonu. Je to i lákadlo pro novou generaci – studenti a mladí programátoři často znají VS Code, takže je pro ně menší bariéra začít pracovat s COBOLem, pokud jej mohou psát ve známém editoru místo ISPF terminálu. Pomocí tohoto rozšíření tedy můžete editovat kódy, ale na připojení a práci s mainframe komponenty slouží další rozšíření [29].

Zowe Explorer

Zowe Explorer je další rozšíření VS Code, které využívá Zowe CLI k propojení s mainframem a používá se právě se zmíněným editorem. V bočním panelu VS Code vám zobrazí připojení k z/OS a umožní procházet datasety, otevírat je, upravovat a ukládat přímo zpět na mainframe prostřednictvím Zowe API. Umožňuje odeslání JCL jobů a následné zobrazení jejich seznamu na spoolu a jejich výstupy. Tím se VS Code stává v podstatě GUI náhradou ISPF – pro mnoho běžných operací netřeba opustit VS Code. Umožňuje také přístup k USS souborům a přes Zowe CLI následně spouštět třeba Python skripty [38].

IBM Db2 for z/OS Developer Extension

Toto rozšíření umožňuje připojení k Db2 databázi na z/OS. Vývojářům poskytne přehled tabulek v databázi, kontrolu syntaxe pro SQL jazyk, jejich automatické základní koncepty a následnou podporu pro spouštění SQL dotazů a jejich zobrazení přímo na z/OS. Toto rozšíření také umožňuje zobrazit detailní proces dotazování na databázi i v případě neúspěšných dotazů pro následné ladění [32].

IBM z/OS Management Facility (z/OSMF)

Webové rozhraní od IBM pro správu z/OS, které obsahuje mj. Workflow Editor, možnost prohlížet joby, editovat PARMLIBy, datasety atd. z/OSMF také poskytuje REST API, jež Zowe částečně využívá. Pro vývoj je relevantní tím, že umožňuje orchestraci nasazování balíčků, konfiguraci subsystémů pomocí web UI [24].

IBM Z Xplore platforma

IBM Z Xplore je vzdělávací platforma navržená společností IBM. Umožňuje zájemcům se seznámit s technologií IBM Z mainframů interaktivní a hlavně praktickou formou. Uži-

vatel postupně prochází jednotlivými výukovými úrovněmi, které kombinují instruktážní videa, přehlednou dokumentaci, praktické úlohy a kvízy. Většina úloh je přímo interaktivní a probíhá na skutečném mainframovém systému IBM Z přes vzdálený přístup. Uživatel se postupně seznámí s dříve popsanými jazyky, naučí se používat VS Code s popsanými rozšířeními a v neposlední řadě i práci s ISPF terminálem. Platforma je rozdělena do tří úrovní (tzv. tiers), přičemž za úspěšné splnění všech úkolů v každé úrovni získá uživatel digitální odznak (badge) přímo od IBM, který lze uplatnit na profesní síti LinkedIn. Osobní dosažení potřebné úrovně jsem již zmínil v úvodu práce. Po dokončení všech úloh má tedy uživatel obecný a celkový přehled o správě z/OS a vývoji aplikací na IBM Z mainframech. Velkým přínosem ovšem také je, že poskytnutý přístup může uživatel využít k vlastnímu vývoji a spouštění aplikací, v rámci smluvních pravidel, na IBM Z bez nutnosti speciálního oprávnění, kdy v dnešní době není přístup k mainframe prostředí obvyklý.

IBM Developer for z/OS (IDz)

IBM Developer for z/OS, zkráceně IDz, je integrované vývojové prostředí (IDE) poskytované IBM speciálně pro vývoj aplikací na z/OS. Jde o robustní sadu nástrojů pro editaci, kompilaci, ladění a analýzu mainframe kódu, která odpovídá moderním principům vývoje softwaru DevOps. IBM nabízí dvě edice. Základní IDz nabízí rozsáhlé nástroje pro vývoj aplikací v jazycích COBOL, PL/I, High Level Assembler, REXX, C/C++, JCL a Java. Enterprise verze IDzEE navíc poskytuje rozšíření do VS Code spolu s velice využívaným debugovacím nástrojem, nástroji pro průběžné sestavení a nasazení aplikací a jiné nástroje. Benefity používání IDz jsou tedy rychlejší a odladěné dodání aplikací. Případové studie ukazují, že Swedbank nebo Danske Bank při používání IDz zefektivnily vývoj služeb pro potřeby zákazníků.

IDz má několik zásadních vlastností. Funkce pro editor zahrnují lepší navigaci, kontrolu syntaxe v reálném čase. Ladící komponenta IBM z/OS Debugger poskytuje podporu pro ladění aplikací v již uvedených jazycích. Dále IDz obsahuje nástroje pro analýzu struktury a kvality aplikací spolu s IBM Dependency Based Build, který umožňuje standardizovat procesy vývoje (DevOps) s integrací Gitu a Jenkins.

Celkově IDz výrazně snižuje čas potřebný k provedení změn v mainframe aplikacích a zvyšuje kvalitu díky nástrojům jako statická analýza kódu nebo debugger. IDz tedy představuje de facto standardní IDE pro mainframe dnes. IDz nabízí nejucelenější řešení s podporou IBM [19].

Celkově lze konstatovat, že tradiční mainframe technologie a moderní vývojářské praktiky se dnes sbíhají. Znalost COBOLu a JCL lze nyní doplnit dovedností v Pythonu a práci s Git/Jenkins, čímž se profil mainframe specialisty rozšiřuje. Pro firmy to znamená možnost dále využívat robustní mainframe systémy a současně se zapojit do trendů jako DevOps, hybrid cloud a AI – aniž by musely vše přepisovat do jiných technologií. Příkladem takového propojení je právě kombinace jazyků a nástrojů popsaných v této kapitole.

Kapitola 5

Obecný popis administračních demonstračních úloh

Tato kapitola se zaměří na obecný popis implementovaných úloh v rámci demonstrace základní práce na z/OS. Cílem úloh je ukázat základní administrátorské úkony jak v rámci klasického přístupu – použití JCL, COBOLu a REXX – tak moderního – Python skriptu. Python umožňuje implementaci úloh v rámci jednoho skriptu díky knihovně `zoautil_py`, která poskytuje různá API pro práci s prostředím z/OS. Tedy místo učení, jak funguje JCL a COBOL, může začínající vývojář pracovat s moderním a přehlednějším Pythonem a díky funkcím z knihovny pracovat přímo s joby nebo datasety.

Popis v této kapitole bude spíše obecný, s popisem filozofie jednotlivých úloh. Úkázky budou pouze ze zajímavých částí, zjednodušené skoro do formy pseudokódu, proto doporučuji i zároveň procházet úlohy odevzdané v příloze A pro hlubší pochopení.

5.1 Zálohování

V každém systému, tak i v prostředí z/OS, je běžnou administrační úlohou vytvoření zálohy. Tato úloha tedy funguje velice jednoduše. Výsledný PDS `SYSUID..TRXREC.BACKUP` se zálohou je vytvořen podle člena z PDS `SYSUID..INPUT(TRXREC)`. V této úloze je demonstrován tradiční přístup pomocí utility **`IEBGENER`** v rámci JCL jobu a moderní alternativa v jazyce Python, využívající knihovnu `zoautil_py`. Nyní bude popsán obsah složky `TRXREPORT` → `BACKUP` → `REGULAR`.

Tradiční přístup

Job `TRXBCKP` se skládá ze dvou kroků. Nejprve pomocí utility `IDCAMS` odstraní případný předchozí záložní dataset a následně pomocí `IEBGENER` zkopíruje člena `TRXREC` z datasetu `Z60930.INPUT` do nového sekvenčního datasetu:

```
//BACKUP EXEC PGM=IEBGENER
//SYSUT1 DD DSN=\&SYSUID..INPUT(TRXREC),DISP=SHR
//SYSUT2 DD DSN=\&SYSUID..TRXREC.BACKUP,DISP=(NEW,CATLG),...
```

Tento přístup vyžaduje hlubší znalost JCL syntaxe, datových parametrů a standardních utilit systému z/OS. Výhodou je plná kontrola nad způsobem zpracování a vlastnostmi výsledného datasetu.

Moderní přístup

Python skript `trxbackup.py` provádí stejnou operaci výrazně jednodušeji. Člen `TRXREC` je zkopírován do datasetu s názvem ve formátu `HLQ.TRXREC.BACKUP.BCyyyymm`, kde `yyyymm` značí aktuální rok a měsíc.

```
from zoautil_py import datasets
today = datetime.today().strftime('%Y%m')
source = "{HLQ}.INPUT(TRXREC)"
target = "{HLQ}.TRXREC.BACKUP.BC{today}"
datasets.copy(source, target)
```

Znatelným zjednodušením pro vývojáře je zde použití knihovny `datetime` v Pythonu, díky které můžeme aktuální datum připojit jako název vytvářeného datasetu. V JCL toto udělat přímo nemůžeme a je na to potřeba speciální REXX skript navíc.

Zálohování pomocí Generation Data Group

Jedním ze způsobů, jak do JCL přidat logiku, díky které budou zálohované datasety odlišeny jménem podobně jako u zmíněného Python skriptu, je použití **Generation Data Group**. GDG představuje systémovou strukturu, která sdružuje jednotlivé verze datasetů (tzv. generace) pod jednu bázi. Každá nová generace zálohovaného souboru je označena indexem jako `(+1)`, `(0)`, `(-1)` atd., přičemž systém automaticky spravuje jejich uchování. Nyní bude popsán obsah složky `TRXREPORT` → `BACKUP` → `GDG VERSION`.

Klasický přístup

GDG báze je definována pomocí utility `IDCAMS` v JCL skriptu. V následujícím příkladu je vytvořena báze `Z60930.TRXREC.BCKPGDG`, která uchovává maximálně 10 aktivních generací.

```
//SYSIN    DD *
  DEFINE GDG(NAME(Z60930.TRXREC.BCKPGDG) -
             LIMIT(10) -
             NOEMPTY -
             SCRATCH)
/*
```

Vysvětlení parametrů:

- `LIMIT(10)`: Uchovává maximálně 10 generací.
- `NOEMPTY`: Při překročení limitu se smaže pouze nejstarší generace.
- `SCRATCH`: Staré generace jsou fyzicky odstraněny (na rozdíl od `NOSCRATCH`).

Pro kontrolu aktuálního stavu GDG báze se používá příkaz `LISTC` utility `IDCAMS`. Skript `LISTBCKP.jcl` vypíše všechny katalogizované generace báze, včetně data vytvoření, stavu katalogizace a dalších metadat.

```
//SYSIN    DD *
  LISTC LEVEL(Z60930.TRXREC.BCKPGDG)
/*
```

Pro vytvoření nové zálohy se nejčastěji používá program `IEBGENER`, který umožňuje jednoduché kopírování dat. Skript `TRXBCGDG.jcl` zajistí zápis obsahu členu `TRXREC` ze vstupního datasetu do nové generace `(+1)` do GDG báze.

```
//BACKUP   EXEC PGM=IEBGENER
//SYSUT1   DD DSN=&SYSUID..INPUT(TRXREC),DISP=SHR
//SYSUT2   DD DSN=Z60930.TRXREC.BCKPGDG(+1),
//            DISP=(NEW,CATLG,DELETE),
//            UNIT=SYSDA,
//            SPACE=(TRK,(1,1),RLSE),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=0)
```

Moderní přístup

Pomocí modulu `gdgs` se dají vytvářet a pracovat s GDG jako v JCL. Modul poskytuje funkce pro vytvoření `base` nebo zobrazení generací. Pro vytvoření nové generace se používá copy funkce z modulu `datasets`. Zkrácená ukázka využití těchto funkcí demonstruje.

```
gdgs.create(name=gdg_base, limit=10, empty=False, scratch=True)
generations = gdgs.GenerationDataGroupView(gdg_base).generations()
for gen in generations:
        print(f" - {gen.name}")
datasets.copy(
        source=input_pds_member,
        target=new_generation
    )
```

Využití GDG na z/OS je robustním způsobem správy verzovaných záloh. Systém sám udržuje historii a automaticky odstraňuje přebytečné verze dle definice báze. Takový přístup je vhodný zejména při opakovaném zálohování, kde je důležitá spolehlivost, přehlednost a automatizace správy verzí.

5.2 Zpracování dat

Zpracování dat je také základní administrátorskou úlohou na systémech. Tyto úlohy demonstrují zpracování vstupního textu – dat – a následný formátovaný výstup.

Zpracování transakčních dat

Tato úloha zpracovává pevně strukturované transakční záznamy z členu `TRXREC` a jejím cílem je vytvořit výstupní zprávu obsahující původní záznamy, seznam identifikátorů, celkovou částku a počet zpracovaných transakcí. Jde o základní úlohu, bez validací vstupů a složitého formátování. Nyní bude popsán obsah složky `TRXREPORT` → `TRXREPORTNORMAL`.

Klasický přístup

Job `TRXREP` se skládá ze tří kroků: smazání předchozího výstupu, kompilace a linkování programu `TRXPROC` a samotného spuštění programu. Vstup je připojen pod DDNAME `INPUT`, výstup pod `OUTPUT`. Poprvé se vyskytuje v úlohách kompilace a linkování COBOL kódů pomocí utility `IGYWCL`.

```
//TRXCOMP       EXEC IGYWCL
//COBOL.SYSIN   DD DSN=&SYSUID..CBL(TRXPROC),DISP=SHR
//LKED.SYSLMOD  DD DSN=&SYSUID..LOAD(TRXPROC),DISP=SHR
//LKED.SYSPRINT DD DUMMY
```

COBOL program `TRXPROC.cbl` zpracovává každý záznam podobně jako v této ukázce (zjednodušeně):

```
READ INPUT-FILE INTO INPUT-RECORD AT END SET END-OF-FILE TO TRUE
MOVE IN-REC TO OUT-REC
ADD TRX-AMOUNT TO TOTAL-AMOUNT
MOVE "Trans ID: " TO WS-TRANS-LINE(1:11)
MOVE TRX-ID-TEXT(WS-INDEX) TO WS-TRANS-LINE(12:10)
WRITE OUT-REC FROM TOTAL-AMOUNT
```

COBOL program čte pevně formátované záznamy, počítá celkovou částku a formátuje výstupní řádky pro výstup. Záznamy i formátování výstupu jsou řízeny přesně definovanými strukturami ve `WORKING-STORAGE SECTION` a `FILE SECTION`.

Moderní přístup

Moderní alternativa v jazyce Python využívá knihovnu `zoautil_py` a je výrazně čitelnější a snáze laditelná, díky jednoduchému zpracování textu a realizuje zpracování vstupních dat v několika krocích:

```
lines = datasets.read("{HLQ}.INPUT(TRXREC)").splitlines()
for line in lines:\ extract\ ID\ and\ amount
total = sum(amounts)
datasets.write("{HLQ}.OUTPUT.PYTHON.TRXREP", content=report)
```

Generování reportu s řazením záznamů

V této části je demonstrován rozdílný přístup k vytváření reportu nad transakčními záznamy pomocí klasického JCL jobu a moderního skriptu v jazyce Python. Obě řešení pracují se stejným vstupem – pevně strukturovanými záznamy obsahujícími ID transakce a částku jako u předešlé úlohy, ale čísla ID jsou náhodně prohozena. Cílem je vytvořit výstupní report se seřazenými záznamy dle ID transakce. Zpracování záznamů v COBOLu je stejné, jelikož vstup je už seřazený v JCL. Nyní bude popsán obsah složky `TRXREPORT` → `TRXREPORTWITHSORT`.

Klasický přístup

V ukázce ze skriptu `TRXREPWS.jcl` je použit klasický přístup využívající JCL job a systémový nástroj DFSORT k seřazení dat uložených v memberu `TRXRECNS`. SORT utility jsou na platformě z/OS běžně používány k zajištění řazení a filtrování dat bez nutnosti programování v aplikačním jazyce. Klíčovou částí jobu je použití parametrů `SORT FIELDS`:

```
//SORTSTEP EXEC PGM=SORT
//SYSIN    DD  *
    SORT FIELDS=(1,10,CH,A)
/*
```

Tato konfigurace specifikuje, že data mají být seřazena podle prvních deseti znaků (tedy ID transakce) jako znakové (character – CH) a vzestupně (ascending – A).

Výhodou tohoto přístupu je plná integrace do mainframe prostředí a jednoduchá syntaxe `SORT FIELDS`. Nevýhodou je však nižší flexibilita a čitelnost při složitějších transformačních operacích nad daty.

Moderní přístup

Alternativní řešení je realizováno pomocí skriptu `trxreportwithsort.py`, který běží v prostředí UNIX System Services (USS) na z/OS. Skript využívá knihovnu `zoautil_py` pro přístup k datasetům a k řazení slouží vestavěná funkce `sorted`. Ukázka řazení v Python skriptu je velmi stručná a přehledná:

```
# Each transaction is a tuple: (trx_id, amount, original_line)
# The lambda function selects the first element (trx_id) for sorting.
def sort_transactions(transactions):
    return sorted(transactions, key=lambda x: x[0])
```

Tento zápis jasně ilustruje výhodu moderního přístupu – jednoduchost, čitelnost a plnou kontrolu nad daty v rámci jednoho programovacího prostředí. Navíc je možné skript dále rozšiřovat o další analytické funkce bez potřeby přepisování JCL jobů nebo použití externích utilit. Oba přístupy dosahují funkčně shodného výsledku – seřazeného výstupního reportu transakcí. Zatímco JCL řešení reprezentuje tradiční a optimalizovaný přístup typický pro mainframe provoz, Python skript nabízí přístupnost, přenositelnost a moderní vývojářský komfort.

Zpracování prodejních dat a generování reportu

Tato úloha má za cíl vytvořit report z pevně strukturovaného vstupu obsahujícího informace o prodaných produktech. Každý záznam obsahuje kód produktu, jeho název, počet prodaných kusů a jednotkovou cenu. Výsledný report sumarizuje prodeje jednotlivých položek včetně celkové hodnoty všech uskutečněných prodejů a počtu zpracovaných záznamů. Nyní bude popsán obsah složky `SALESREPORT`.

Klasický přístup

Tradiční řešení využívá kombinaci JCL, programovacího jazyka COBOL, který zpracovává vstup, a skriptu napsaného v jazyce REXX. COBOL program načte záznamy ze vstupního datasetu `HLQ.INPUT(SALESDT)`, zpracuje jednotlivé položky a vytvoří výstupní report s detailními řádky a celkovým součtem.

Klíčová část zpracování jednotlivých položek prodeje v COBOLu je oddělena do samostatné procedury `PROCESS-SALES` a zahrnuje výpočet mezisoučtu a sestavení formátovaného výstupního řádku. Zjednodušená ukázka odpovídající implementace `main` funkce:

```
MOVE HEADER TO OUT-REC
READ SALES-IN UNTIL END-OF-FILE
PERFORM PROCESS-SALES
    MOVE PROD-ID    TO OUT-REC
    MOVE PROD-NAME  TO OUT-REC
    MOVE QTY-SOLD   TO OUT-REC
    MOVE UNIT-PRICE-DISPLAY TO OUT-REC
    MOVE PROD-TOTAL-DISPLAY TO OUT-REC
    MOVE NUMBER-OF-TRANS TO OUT-REC
END-PERFORM
```

REXX skript v tomto případě slouží jako plánovací nebo spouštěcí mezivrstva – je zodpovědný za opakované „submitování“ JCL. Podle nastavení parametrů času lze zprávy o prodejích provádět každý den, nebo jednou za měsíc atd. Takový skript může znovu zjednodušeně vypadat následovně:

```
/* REXX */
target_day = "Monday"
target_time = "02:00"
if current_day = target_day then do
"SUBMIT 'Z60930.JCL(SALESREP)'"
```

Automatickou alternativou k plánovanému spouštění JCL je použití integrovaných nástrojů dostupných na z/OS, jako **Tivoli Workload Scheduler for z/OS (OPC) component**.

Moderní přístup

Alternativní řešení je realizováno pomocí skriptu `salesrep.py`, který běží v prostředí UNIX System Services (USS) na z/OS. Skript využívá knihovnu `zoautil_py` pro přístup k datasetům a jednoduchou práci s listem.

V ukázce skriptu `salesrep.py` je demonstrován zjednodušený způsob, jak jednoduše načíst pevně formátované záznamy, extrahovat požadované informace a zapsat vše do výstupního datasetu:

```
report_lines = []
report_lines.append(header_line)
for rec in sales_records
    report_lines.append()
datasets.write(output_ds, content=report_content)
```

Díky přístupu k systémovým datasetům prostřednictvím `zoautil_py` lze zcela obejít potřebu JCL nebo REXX při opakovaném spouštění – úlohy lze plánovat například pomocí `crontab` přímo v USS.

Zpracování osobních dat zaměstatnanců

Tato úloha má za cíl zpracovat osobní data a zároveň k tomu přidat i data o výplatách jednotlivých zaměstnanců. Každý zaměstnanec má svůj vlastní identifikátor, podle kterého se zpracovávají záznamy. Toto je poslední úloha, která zpracovává jednoduché vstupní záznamy do výstupu. Nyní bude popsán obsah složky `EMPLOYEESREPORT`.

Klasický přístup

JCL kód se v této úloze liší od předešlých. Jde o ukázku, jak se skrze JCL dá poskytnout více vstupních členů pro zpracování COBOLem. Zde je ukázka JCL kódu s definicí jednotlivých vstupních členů pomocí DD – definicí datasetu:

```
//RUN       EXEC PGM=EMPREP
//STEPLIB   DD DSN=&SYSUID..LOAD(EMPREP),DISP=SHR
//EMPPER    DD DSN=&SYSUID..INPUT(EMPPER),DISP=SHR
//EMPDEPT   DD DSN=&SYSUID..INPUT(EMPDEPT),DISP=SHR
//EMPSAL    DD DSN=&SYSUID..INPUT(EMPSAL),DISP=SHR
//REPORT    DD DSN=&SYSUID..OUTPUT.EMPREP,DISP=(NEW,...
```

COBOL program následně čte a zpracovává data všech vstupních členů. Očekává se, že ID zaměstnanců jsou seřazena ve všech členech, aby nedošlo k chybě. Zjednodušená ukázka poskytuje přehled, jak se v COBOLu dá pracovat s více vstupními členy:

```
FILE-CONTROL.
           SELECT EMP-PERSONAL ASSIGN TO 'EMPPER'
               ORGANIZATION IS SEQUENTIAL.
           SELECT EMP-DEPT     ASSIGN TO 'EMPDEPT'
               ORGANIZATION IS SEQUENTIAL.
           SELECT EMP-SALARY   ASSIGN TO 'EMPSAL'
               ORGANIZATION IS SEQUENTIAL.
...
FD EMP-PERSONAL.
01 EMP-PERSONAL-REC.
FD EMP-DEPT.
01 EMP-DEPT-REC.
FD EMP-SALARY.
01 EMP-SALARY-REC.
...
MOVE WS-EMP-ID     TO WS-REPORT-LINE.
MOVE WS-FIRST-NAME TO WS-REPORT-LINE.
...
WRITE EMP-OUT-REC FROM WS-REPORT-LINE.
```

Moderní přístup

Řešení v Pythonu se v této úloze liší oproti předešlým úlohám. Jde o ukázku, jak se tato úloha dá plně převést do moderní podoby, a to tím, že vstup se načítá ze dvou `JSON` souborů uložených přímo v USS IBM Z. Nicméně výstup se stále ukládá do datasetu.

Python skript pracuje s knihovnou `json`, která se použije pro načtení vstupních souborů. Následně se data uloží do slovníku, který se zpracovává pro výsledný výstup. Zde je zjednodušená ukázka zpracování dat:

```
with open(input_personal_file, 'r') as f:
    personal_data = json.load(f)
...
salary_dept_map = {rec["EMP_ID"]: rec for rec in salary_dept_data}
...
for person in personal_data:
    emp_id = person["EMP_ID"]
    if emp_id in salary_dept_map:
...
merged_data.append(merged_record)
...
for rec in merged_data:
    line = (
        f"{rec['EMP_ID']:<6} "
        ...
    )
    report_lines.append(line)
```

Díky použití knihovny a následného slovníku můžeme jednoduše např. kontrolovat, jestli oba vstupy obsahují stejné ID zaměstnanců, kdy by tohle v COBOLu nebylo tak triviální. Zároveň to zpřehledňuje celou logiku skriptu, zvláště pro vývojáře, kteří umí zpracovávat JSON soubory i mimo prostředí mainframe.

5.3 Úlohy na převod formátu vstupních dat

Převod dat mezi formáty je důležitý na každém systému. Obecně jsou data na z/OS ve formátu „fixed-width“. Tento formát má data, kde každé pole má pevnou šířku. Při zpracovávání dat v rámci analýzy nebo administrace je ale dobré mít data uložená ve formátu, který je standardizovaný a lehce zpracovatelný i pro jiné podpůrné programy.

Převod do formátu CSV

Formát **CSV** (Comma-Separated Values) je jednoduchý textový formát pro reprezentaci tabulkových dat. Každý řádek odpovídá jednomu záznamu a jednotlivé hodnoty jsou odděleny čárkou. CSV se hojně používá při exportu a importu dat mezi systémy.

Tato úloha demonstruje převod pevných textových záznamů reprezentujících údaje o zaměstnancích do formátu CSV. Data jsou uložena jako záznamy s pevnou délkou 80 znaků. Nyní bude popsán obsah složky `CSVCONVERT`.

Klasický přístup

COBOL program `CONVCSV.cbl` čte vstupní soubor přes DD jméno `INPUT` a výstup zapisuje pod `OUTPUT`. Každý vstupní řádek je zpracován na základě pevné pozice jednotlivých polí a výstupní CSV řádek je dynamicky sestavován pomocí řetězcových operací. Kód ukazuje, jak COBOL zpracovává záznam:

```
MOVE IN-LINE(1:5) TO EMP-ID
MOVE IN-LINE(6:19) TO EMP-NAME
MOVE IN-LINE(20:35) TO EMP-DEPT
MOVE IN-LINE(36:43) TO EMP-SALARY
MOVE IN-LINE(44:51) TO EMP-DOB  #Date of birth

STRING EMP-ID DELIM ','
EMP-NAME DELIM ','
EMP-DEPT DELIM ','
EMP-SALARY-FORMATTED DELIM ','
EMP-DOB DELIM SIZE INTO OUT-LINE
WRITE OUT-RECORD FROM OUT-LINE
```

Program také implementuje konverzi platového údaje z celočíselného formátu s implicitními desetinnými místy (např. `00085000` jako `850.00`) a formátuje jej do výstupního řetězce. Výsledkem je řádek CSV odpovídající specifikaci. Využívá se JCL job `CONVCSV.jcl`, jehož krok spouští kompilovaný program, stejně jako u předchozích ukázek.

Moderní přístup

Moderní varianta ve formě skriptu `convertcsv.py` využívá knihovnu `zoautil_py` k přímému čtení a zápisu datasetů na z/OS. Program načte každý řádek, provede rozřezání textu dle pozic a výstup zapíše do nového sekvenčního datasetu. Příklad části zpracování:

```
emp_id = line[0:5].strip()
name = line[5:24].strip()
dept = line[24:39].strip()
salary_raw = line[39:47].strip()
dob = line[47:55].strip()

salary = f"{int(salary_raw) / 100:.2f}" return f"{emp_id},
                                        {name},{dept},{salary},{dob}"
```

Výhodou tohoto přístupu je čitelnější syntaxe, jednodušší manipulace s textem a lepší laditelnost. Z pohledu vývoje i údržby je tato moderní varianta rychlejší na implementaci a méně náchylná k chybám ve formátování. Výsledek odpovídá očekávanému CSV výstupu.

```
ID,Name,Department,Salary,DOB
00001,John Doe,SALES,750.00,19900101
...
```

Převod do formátu XML

Formát **XML** (eXtensible Markup Language) slouží k reprezentaci hierarchických a strukturovaných dat ve formátu čitelném pro člověka i stroj. Každý záznam je reprezentován jako uzel (např. `<Employee>`) a jednotlivé položky jako poduzly (např. `<Name>`). XML je často používán pro přenos dat mezi systémy, včetně webových služeb nebo rozhraní mezi staršími systémy a moderními aplikacemi.

Úkolem této úlohy je převést fixní vstupní záznamy do XML formátu. Implementace je opět ukázána jak v COBOLu s využitím JCL, tak v jazyce Python. Nyní bude popsán obsah složky `XMLCONVERT`.

Klasický přístup

Program `CONVXML.cbl` čte jednotlivé vstupní řádky a převádí je do podoby XML uzlů pomocí stringových operací. Každé pole je extrahováno na základě pevné pozice:

```
MOVE IN-LINE(1:5) TO EMP-ID
MOVE IN-LINE(6:19) TO EMP-NAME
MOVE IN-LINE(20:35) TO EMP-DEPT
MOVE IN-LINE(36:43) TO EMP-SALARY
MOVE IN-LINE(44:51) TO EMP-DOB
```

Následně se každý údaj obalí XML značkami pomocí příkazů `STRING`. Odsazení kvůli přehlednosti se provede pomocí mezer:

```
STRING "<Employee>
<ID>" DELIMITED BY SIZE EMP-ID DELIMITED BY SIZE "</ID>
<Name>" EMP-NAME "</Name>" ...  INTO XML-RECORD
WRITE OUT-RECORD FROM XML-RECORD
```

Výstupní soubor je doplněn o hlavičku a kořenový element `<Employees>`.

Moderní přístup

Moderní varianta implementace využívá knihovnu `xml.etree.ElementTree` v jazyce Python a zajišťuje nejen sestavení struktury, ale i její formátování pro čitelnost. Zpracování je rozděleno do dvou částí:

1. **Parsování vstupních řádků:**

```
def parse_employee_record(line):
return { "ID": line[0:5].strip(),
"Name": line[5:24].strip(), ... }
```

2. **Sestavení XML dokumentu:**

```
root = ET.Element("Employees")
for emp in employees:
    e = ET.SubElement(root, "Employee")
    for key, value in emp.items():
    child = ET.SubElement(e, key)
    child.text = value
```

Pro zarovnání a formátování dokumentu je využita knihovna `xml.dom.minidom`, která převádí XML do tzv. pretty-print formy s odsazením:

```
xml_dom = xml.dom.minidom.parseString(xml_str)
indent_xml = xml_dom.toprettyxml(indent=" ")
```

Tato implementace je výrazně modulárnější a snadněji rozšiřitelná. Díky práci se slovníky a nativní XML podpoře je jednodušší přidat další pole, změnit strukturu nebo výstup znovu použít v jiném kontextu. Výsledek obou přístupů je stejný:

```
<?xml version="1.0" encoding="UTF-8"?>
<Employees>
    <Employee>
        <ID>00001</ID>
        <Name>John Doe</Name>
        <Department>SALES</Department>
        <Salary>750.00</Salary>
        <DOB>19900101</DOB>
    </Employee> ...
</Employees>
```

5.4 Vizualizace výstupu

Vizualizace dat je klíčovým nástrojem při interpretaci výsledků a analýze trendů. Moderní jazyky jako Python disponují rozsáhlými knihovnami pro tvorbu grafů (např. `matplotlib`), zatímco tradiční systémy jako z/OS a jazyk COBOL tuto možnost standardně nenabízí. Tato část popisuje přístup, kdy jsou údaje o prodejnosti produktů interpretovány formou **ASCII grafů** a srovnány s výstupem vytvořeným v Pythonu pro lokální prezentaci.

Z důvodu architektury z/OS a omezeného přístupu k moderním vizualizačním knihovnám je tvorba grafů v grafickém formátu prakticky nemožná. V tomto kontextu představují ASCII grafy jedinou smysluplnou alternativu, a proto je tato úloha brána čistě jako demonstrace. Nyní bude popsán obsah složky `CHARTGENERATOR`.

Vstupní data: Vstupní člen obsahuje záznamy o produktech a jejich prodaném množství ve fixní šířce. Každý záznam má 80 znaků:

```
PROD001 CPU Fan 00000100
PROD002 AMD GPU 00000075 ...
```

Pole jsou strukturována následovně:

- 1–10: Kód produktu (ignorováno)
- 11–30: Název produktu
- 31–38: Prodané množství (číselná hodnota)

Klasický přístup

I přes absenci grafických knihoven lze v COBOLu vytvořit základní formu datové vizualizace pomocí ASCII grafu. Tento přístup plně respektuje omezení prostředí z/OS, kde se typicky generují dávkové výstupy určené k tiskům nebo kontrole operátorem.

COBOL program `CHARTGEN.cbl` čte jednotlivé záznamy produktů a prodaného množství, vypočítá relativní délku výstupního sloupce a následně sestaví graf pomocí znaků *. Program podporuje jak horizontální graf, tak vertikální.

Zjednodušená ukázka části programu, která generuje horizontální ASCII sloupec:

```
MOVE INPUT-QUANTITY TO QTY-NUMERIC
DIVIDE QTY-NUMERIC BY 10 GIVING STARS-NUM
MOVE 1 TO IDX PERFORM VARYING IDX FROM 1 BY 1
    UNTIL IDX > STARS-NUM MOVE '*' TO CHART-LINE(IDX:1)
END-PERFORM

WRITE OUTPUT-RECORD FROM OUTPUT-LINE
```

Tento kód ukazuje jednoduchou transformaci číselné hodnoty (např. `100`) na sloupec složený ze znaků *, přičemž každý znak reprezentuje definovaný počet jednotek (např. 10).

Job `CHARTGEN.jcl` spouští COBOL s parametrem `VERT` nebo `HORIZ`

```
//CHARTRUN  EXEC PGM=CHARTGEN,PARM='VERT'
```

Tento přístup je jednoduchý, robustní a nezávislý na externích nástrojích. Hodí se pro prostředí, kde není k dispozici žádné GUI a výstup je určen ke kontrole operátory nebo pro pozdější zpracování.

Moderní přístup

Skript `chartgen.py` je navržen pro běh přímo na z/OS a generuje výstupní graf ve formě ASCII textu – buď horizontální, nebo vertikální. Orientace grafu se určuje argumentem `HORIZ` nebo `VERT`. Ukázka horizontálního výstupu:

```
CPU Fan | ********** (100)
AMD GPU | ******* (75)
```

Základní logika výpočtu délky sloupce:

```
bar_len = qty // SCALE
bar = '*' * bar_len
```

Pro vertikální graf je generována jednoduchá osa Y a hvězdičky jsou umístěny ve sloupcích pod názvy produktů. Výhodou tohoto přístupu je nezávislost na externích knihovnách a možnost nasazení i v omezeném prostředí z/OS.

Lokální vizualizace pomocí matplotlib

Python SDK nabízí možnost si potřebné knihovny doinstalovat, bohužel přístup skrze platformu IBM Z Xplore je velice limitovaný a instalace knihoven je zakázána. Proto vznikl skript, který na lokálním PC simuluje běh skriptu s knihovnou `matplotlib`.

Skript `chartgenlocal.py` zpracovává vstupní data ze souboru `CHARTDT.txt`, který má stejný formát jako člen PDS na mainframu.

Výstupní graf je interaktivní a barevný, což zlepšuje čitelnost a prezentovatelnost, avšak není vhodný pro běh na mainframu.

5.5 Validace a čištění vstupních dat

Reálná data bývají často neúplná, chybně naformátovaná nebo obsahují duplicity. Před jakýmkoli dalším zpracováním je nezbytné provést jejich validaci a případné čištění. V této úloze jsou vstupní záznamy zaměstnanců s pevným formátem. Záznamy jsou kontrolovány na základě definovaných pravidel a pouze validní záznamy jsou přeneseny do výstupního datasetu. Nyní bude popsán obsah složky `DATACLEAN`.

Pravidla validace

Každý řádek má 80 znaků, strukturovaných následovně:

- 1–6: ID (číselné, jedinečné, pouze 6 znaků)
- 7–16: Křestní jméno (pouze písmena)
- 17–26: Příjmení (pouze písmena)
- 27–34: Datum narození (formát YYYYMMDD, validní datum včetně přestupných let, pouze 8 znaků)
- 35–48: Město (jsou povoleny jen náhodně vybraná města ze seznamu – pro účely demonstrace)

Záznamy, které nesplňují jakékoli z těchto pravidel, jsou vyřazeny a důvod vyřazení se nachází v outputu, v případě Python varianty na standardní výstup.

Klasický přístup

COBOL program `DATACLN.cbl` čte vstupní záznamy přes DD jméno `INPUT`, kontroluje jednotlivá pole pomocí podmínek a v případě úspěšné validace zapisuje výsledek do výstupu `OUTPUT`. Validace je implementována pomocí kombinace porovnání, dekódování řetězců a v případě data narození i jednoduché aritmetiky a kontroly měsíců. Města jsou následně kontrolována pomocí tabulky, která je převedena do „pole“ pro indexované hledání. Nevalidní záznamy jsou přeskočeny a zapisovány na výstup i s důvodem selhání validace. Zde je zjednodušená validace platného jména:

```
IF WS-VALID-FLAG = "Y"
               PERFORM VALIDATE-NAME ...
VALIDATE-NAME.
PERFORM VARYING WS-CHAR-IDX FROM 1 BY 1 UNTIL
               WS-CHAR-IDX > LENGTH OF IN-FIRST-NAME ...
                    IF NOT (WS-CURRENT-CHAR >= "A" AND
                       WS-CURRENT-CHAR <= "Z")
                       AND NOT (WS-CURRENT-CHAR >= "a"
                       AND WS-CURRENT-CHAR <= "z")
```

```
                    EXIT PERFORM
                END-IF
                ADD 1 TO WS-CHAR-COUNT
            END-PERFORM
...
```

Moderní přístup

Moderní varianta v jazyce Python (`datacleaner.py`) implementuje shodnou logiku validace a je navržena pro běh pod Unix System Services (USS) v prostředí z/OS. Skript využívá `zoautil_py` API pro čtení a zápis datasetů a definuje validační pravidla formou funkcí:

```
def is_alpha(text: str) -> bool: ...
def is_valid_date(date_str: str) -> bool: ...
def is_valid_city(city: str) -> bool: ...

def process_record(line: str) -> tuple[bool, str]:
...
if not id_num.strip().isdigit():
        return False, "Invalid ID format"
...
```

Každý řádek je rozparsován na jednotlivá pole a validován. Po úspěšné validaci je záznam přidán do seznamu výstupních řádků. Na konci skriptu proběhne rekapitulace výsledků:

```
print("RECORD STATISTICS")
print(f"  Total records   : {len(valid_records) + len(invalid_records)}")
print(f"  Valid records   : {len(valid_records)}")
print(f"  Invalid records : {len(invalid_records)}")
print("--------------------------------------------------")
for rec_id, reason in invalid_records:
    print(f"[REJECTED] ID {rec_id} - {reason}")
```

Skript, tak i COBOL uchovává důvody, proč byly jednotlivé záznamy odmítnuty, např.:

```
[REJECTED] ID 000003 - Invalid surname
[REJECTED] ID 000011 - Invalid date of birth
```

Díky modulárnímu návrhu lze jednoduše měnit validační pravidla, přidávat další kontroly nebo rozšířit výstupní formát o chybová hlášení. Zároveň se validace dat pomocí zabudovaných funkcí velice zjednodušuje oproti COBOLu.

5.6 Demonstrační ukázky knihoven pro Python

V rámci praktické části byly vytvořeny dva pomocné Python skripty, které slouží jako ukázka možností, jež nabízí knihovna `zoautil_py` z IBM Z Open Automation Utilities (ZOAU). Tyto nástroje demonstrují reálné operace, které by běžně prováděl systémový administrátor nebo programátor na mainframu – manipulaci s datasetem a práci s dávkovými úlohami. Nyní bude popsán obsah složky `PYTHONDEMOS`, kde se nachází oba skripty.

Manipulace s datasety

Skript `demo_datasets.py` simuluje reálný scénář správy datasetů na z/OS. Demonstruje použití různých funkcí a následné odchytávání výjimek pomocí modulů `datasets` a `zoautil_py.exceptions` pro následující operace:

- vytvoření PDS datasetů,
- zápis člena (`member`) s uživatelským obsahem,
- kopírování člena mezi datasety,
- porovnání obsahu dvou členů,
- přejmenování celého datasetu a člena,
- výpis existujících datasetů a jejich členů pod prefixem `HLQ.PYDEMO.*`,
- výpis a zobrazení obsahu člena,
- volitelné mazání nebo ponechání vytvořených dat.

Používá strukturovaný systém výpisu, volitelný verbose režim a měření délky jednotlivých operací pomocí argumentů `--verbose`, `--timing` a `--keep`. Umožňuje také pomocí argumentu `--content` zapsat libovolný text do člena. Ukázka části volání:

```
write_member(src_ds, member, ARGS.content)
copy_member(src_ds, tgt_ds, member)
compare_members(src_ds, tgt_ds, member)
rename_dataset(src_ds, renamed_ds)
rename_member(tgt_ds, member, "RENAMED1")
```

Skript je vhodný jako výukový nástroj, díky přehledným komentářům, ale i jako základ pro automatizační scénáře v DevOps prostředí mainframu.

Práce s joby

Skript `demo_jobs.py` demonstruje kompletní cyklus práce s dávkovou úlohou (JCL jobem) na mainframu prostřednictvím ZOAU API. Demonstruje použití různých funkcí a následné odchytávání výjimek pomocí modulů `jobs` a `zoautil_py.exceptions` pro následující operace:

- odeslat dávkovou úlohu (`submit`),
- počkat na její dokončení a načíst její stav,
- ověřit její existenci v systému,
- zobrazit všechny kroky a DD jména,
- načíst obsah konkrétního DD (např. `JESYSMSG`),
- uložit výstup do souboru na USS (`-savedd`),

- získat rozšířené informace o úloze (název, RC, čas spuštění, priorita atd.).

Používá strukturovaný systém výpisu, volitelný verbose režim, umožňuje zvolit job, který se má spustit, dále také jaké DD se má vypsat a v poslední řadě, jestli se má i obsah DD uložit do souboru. To vše pomocí argumentů `--verbose`, `--job-name`, `--dd` a `--savedd`. Zjednodušený příklad volání funkcí:

```
job = jobs.submit(dataset)
wait_for_job(job)
if check_job_exists(job.job_id):
        list_job_dds(job.job_id)
        read_job_output(job.job_id)
        fetch_and_display_extended_info(job.job_id)
```

Skript je užitečný nejen pro vývojáře, ale i pro operátory a správce, kteří chtějí automatizovat monitoring úloh nebo analyzovat data z JES výstupů. Díky argumentům tento skript nemusí sloužit jako demo ukázka, ale i plnohodnotný program.

5.7 Demonstrační ukázka utilit JCL

Tato demonstrační úloha v jazyce JCL byla navržena jako výukový příklad, který pokrývá praktické využití základních systémových utilit z/OS. Jejím cílem je prezentovat běžné operace nad datasety, manipulaci s členy, práci se sekvenčními a PDS datasety, a také spuštění dávkového REXX skriptu. Nyní bude popsán obsah složky `JCLDEMO`, kde se nachází skript.

Job `JCLDEMO.jcl` demonstruje dvanáct základních kroků – od alokace a mazání datasetů, přes generování, kopírování a porovnávání dat, až po spuštění TSO příkazů a REXX skriptu.

Použité utility a prostředky

Job využívá tyto standardní programy a utility z/OS:

- `IEFBR14` – alokace prázdných datasetů,
- `IDCAMS` – mazání datasetů,
- `IEBGENER` – generování nebo kopírování obsahu datasetů,
- `IEBCOMPR` – porovnání obsahu dvou sekvenčních datasetů,
- `SORT` – třídění obsahu pomocí DFSORT,
- `IEBUPDTE` – vytvoření člena v PDS,
- `IEBCOPY` – kopírování všech členů mezi PDS,
- `IKJEFT01` – spuštění TSO příkazu z JCL (např. `LISTC`),
- `IRXJCL` – spuštění REXX skriptu.

Popis kroků jobu

Job začíná krokem `CLEANUP`, který pomocí utility `IDCAMS` maže případné zbytky z předchozího běhu, aby bylo možné úlohu spustit opakovaně bez zásahu.

Následuje alokace sekvenčních a partitioned datasetů:

```
//STEP01 EXEC PGM=IEFBR14
//DEMO01 DD DSN=&SYSUID..DEMO.FILE1, ...
```

Dále jsou pomocí `IEBGENER` zapsány testovací řádky do datasetu, zkopírovány do druhého datasetu, a následně oba porovnány pomocí `IEBCOMPR`. Dalším krokem je třídění obsahu datasetu pomocí utility `SORT`:

```
SORT FIELDS=(1,80,CH,A)
```

Poté následuje vytvoření člena `MEMBER1` v PDS pomocí utility `IEBUPDTE` a jeho zkopírování do nového PDS pomocí `IEBCOPY`. V kroku `STEP08` je spuštěn TSO příkaz `LISTC LEVEL(...)` k výpisu katalogovaných datasetů daného uživatele:

```
//STEP08 EXEC PGM=IKJEFT01
//SYSTSIN DD * LISTC LEVEL(Z60930.DEMO)
```

Spuštění REXX skriptu

V závěrečném kroku `STEP11` je spuštěn dávkový REXX skript jménem `JCLDEMO`. Skript je spouštěn pomocí utility `IRXJCL`:

```
//STEP11 EXEC PGM=IRXJCL,PARM='JCLDEMO'
//SYSEXEC DD DSN=&SYSUID..INPUT,DISP=SHR
//SYSTSPRT DD SYSOUT=* //SYSTSIN DD DUMMY
```

To znamená, že skript s názvem JCLDEMO je hledán jako člen v datasetu `Z60930.INPUT`, připojeném přes `SYSEXEC`.

Obsah REXX skriptu

Skript `JCLDEMO.rexx` je jednoduchý demonstrační program, který vypisuje uvítací zprávu:

```
/* REXX */
say "HI FROM REXX SCRIPT"
exit
```

Skript je určen jako ilustrace toho, jak lze v rámci dávkové úlohy použít klasické JCL utility se skriptovací logikou v REXXu.

5.8 Měření výkonu a testovací prostředí

Testy výkonu byly provedeny na platformě IBM Z Xplore opětovným spouštěním, jak python skriptů v prostředí Unix System Services (USS), tak JCL jobů, které zpracovává JES2 subsystém. Platforma IBM Z Xplore běží na mainframu IBM z16 (model A01), ale nejde

zjistit s jakou mírou virtualizace. Pro každou zkoumanou variantu (JCL/COBOL vs. Python) bylo zvoleno definované množství vstupních dat a zkoumané úlohy byly `TRXREPORT`, `BACKUP`, `XML CONVERT`, `CHART GENERATOR`, `DATA CLEAN` a `PYTHONDEMOS - práce s datasety`. Každý test byl opakován minimálně dvakrát, čímž se minimalizoval vliv proměnných okolních podmínek a umožnilo se stanovit průměrné hodnoty. Demonstrativně jsem provedl i více měření, ale čas trvání JCL jobů se neměnil. Python skripty také vykazovaly stabilní časy běhu.

Výsledky testu `PYTHONDEMOS - práce s datasety`

Skript `demo_datasets.py` byl spuštěn třikrát s parametrem `time`. Naměřené celkové časy běhu byly:

- **1. běh**: 5 m 9,393 s
- **2. běh**: 5 m 4,730 s
- **3. běh**: 4 m 56,627 s

Průměrný čas běhu činil přibližně 5 m 3,58 s (rozptyl 4 m 56,627 s – 5 m 9,393 s). Následující tabulka 5.1 ukazuje průměrné, minimální a maximální doby vykonání jednotlivých funkcí.

Funkce	Min (s)	Max (s)	Průměr (s)
ensure_dataset_exists_and_create	6,462	6,472	6,467
write_member	1,665	1,667	1,666
copy_member	64,221	65,496	65,022
compare_members	92,101	107,850	96,482
rename_dataset	19,426	22,454	20,223
rename_member	25,931	29,202	26,803
show_dataset_info	23,796	28,579	25,058
read_and_print_member	1,488	1,834	1,590
cleanup	49,573	56,927	51,495

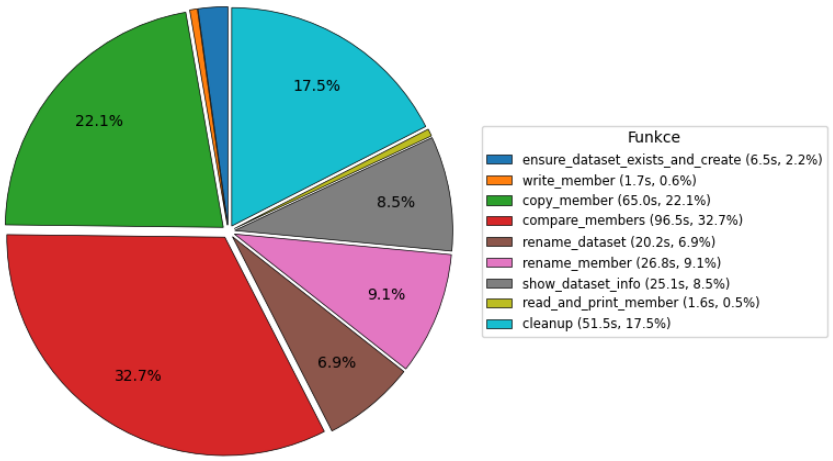
Tabulka 5.1: Doby vykonání funkcí skriptu `demo_datasets.py`

V následujícím výčtu je uvedeno, jaké metody z modulu `datasets` jsou využity v jednotlivých pomocných funkcích skriptu `demo_datasets.py`:

- **ensure_dataset_exists_and_create**: `datasets.exists()`, `datasets.create()`
- **write_member**: `datasets.write()`
- **copy_member**: `datasets.copy()`
- **compare_members**: `datasets.compare()`
- **rename_dataset**: `datasets.move()`
- **rename_member**: `datasets.move_member()`
- **show_dataset_info**: `datasets.list_datasets()`, `datasets.list_members()`, `datasets.find_member()`

- **read_and_print_member**: `datasets.read()`
- **cleanup**: `datasets.delete_members()`, `datasets.delete()`

Na obr. 5.1 je znázorněn kruhový graf relativních podílů průměrných dob vykonání funkcí ve skriptu `demo_datasets.py`.



Obrázek 5.1: Podíl průměrných dob vykonávání funkcí

Výsledky testu `TRXREPORT` (bez řazení)

Pro 100 vstupních dat byly naměřeny následující časy:

- **(COBOL/JCL)**: průměrně 0,12 min (7,2 s)
- **Python (se výpisem seznamu členů)**: 1 m 2,199 s a 1 m 1,098 s (průměr 61,65 s)
- **Python (bez výpisu seznamu)**: 37,870 s a 37,733 s (průměr 37,80 s)

Výsledky testu `TRXREPORT` (s řazením)

Pro 100 vstupních dat byly naměřeny následující časy:

- **(COBOL/JCL)**: průměrně 0,14 min (8,4 s)
- **Python (bez výpisu seznamu)**: 35,788 s, 37,372 s, 36,163 s (průměr 36,44 s)

Testování obou úloh proběhlo pouze na 100 datech, zpracování COBOLem, kdy se používala SORT utilita v JCL, trvalo cca o vteřinu déle. Dá se tedy očekávat výrazné zpomalení např. u 10 000 záznamů. Takové detailní měření nicméně není zadáním této práce, jde spíše o porovnání výkonnosti mezi klasickým a moderním přístupem.

Výsledky testu `BACKUP`

Pro 100 vstupních dat:

Regular

- **(COBOL/JCL)**: průměrně 0,02 min (1,2 s)
- **Python**: 1m 15,121 s a 1m 15,782 s (průměr 75,45 s)

GDG version

- **(COBOL/JCL)**: průměrně 0,01 min (0,6 s)
- **Python**: 1m 9,774 s a 1m 9,430 s (průměr 69,60 s)

Zde jde vidět poměrně velký rozdíl v rychlosti jednotlivých skriptů. Hlavním důvodem je použití `copy` funkce v Pythonu, která se používá ve funkci `copy_member` a z tabulky 5.1 jde vyčíst její časovou náročnost.

Výsledky testu `XML CONVERT`

Pro 5 vstupů byly naměřeny následující časy:

- **(COBOL/JCL)**: 0,12 min (7,2 s)
- **Python**: 35,961 s a 35,732 s (průměr 35,85 s)

Zde jde vidět klasický rozdíl ve výkonnosti, jelikož se zde v Pythonu nepoužívá `copy` funkce.

Výsledky testu `CHART GENERATOR`

Pro 5 vstupů:

Horizontální výstup

- **(COBOL/JCL)**: 0,12 min (7,2 s)
- **Python**: 35,729 s a 35,922 s (průměr 35,83 s)

Vertikální výstup

- **(COBOL/JCL)**: 0,12 min (7,2 s)
- **Python**: 36,052 s, 32,882 s a 32,306 s (průměr 33,75 s)

Zde jde také vidět klasický rozdíl ve výkonnosti. Zajímavostí je v tomto případě běh COBOL/JCL, kdy nedojde k žádnému zpoždění u vertikální varianty, u které je mnohem složitější logika vykreslování.

Výsledky testu `DATA CLEAN`

Pro 30 vstupů byly naměřeny následující časy:

- **(COBOL/JCL)**: 0,12 min (7,2 s)
- **Python**: 32,354 s a 32,341 s (průměr 32,35 s)

Zde jde také vidět klasický rozdíl ve výkonnosti. Dále by šlo zkoumat, jak by se lišily výsledky, kdyby se přidalo více záznamů na zpracování. Podobný případ jako u `TRXREPORT`.

Shrnutí

Ve všech testech skripty spuštěné v Pythonu prostřednictvím USS vykazují časy v řádu desítek sekund až minut, zatímco ekvivalentní COBOL/JCL implementace na mainframu dosahují řádově jednotek sekund. Je nutno také poznamenat, že při použití stejných funkcí z modulu `zoautil_py` vykazují Python skripty podobnou časovou náročnost. Dále použití funkcí z modulu `datasets` v jakémkoli skriptu odpovídá časům naměřeným v referenčním skriptu `demo_datasets.py`. Průměrné hodnoty uvedené v tabulce 5.1 lze využít pro teoretické odhady doby běhu Python skriptů při jejich návrhu. Největší relativní rozdíl byl zaznamenán u operací `BACKUP (GDG version)`, kde Python skript byl až 116× pomalejší než COBOL/JCL. Z pohledu návrhu efektivních řešení je proto nezbytné tyto rozdíly zohlednit již ve fázi analýzy a optimalizovat kritické funkce.

Porovnáním výkonu se zabýval i Aswin Venkat, který porovnával výkon různých typů řazení v Pythonu a COBOLu na vzorku převážně milionu dat. Nicméně způsob testování není úplně popsán, COBOL kód je nejspíše spouštěn na mainframu a Python na lokálním PC. Největší rozdíl nastal při testu DF Sort (vestavěná JCL metoda) vs. df.sort_values (metoda z knihovny Pandas). Nativní JCL DF Sort byl téměř 16× rychlejší oproti Pythonu [53].

Kapitola 6

Závěr

Tato bakalářská práce poskytla ucelený a hloubkový pohled na platformu mainframe. V první části práce byla zachycena evoluce mainframových systémů od prvních elektronkových strojů po nejnovější IBM z17, jehož detailní specifikace byla vydána pár týdnů před odevzdáním této práce. Detailní přehled klíčových milníků (System/360, System/370, z/Architecture) uvedl, jak postupné inovace formovaly dnešní podobu IBM Z mainframů.

Práce podrobně rozebrala moderní 64bitovou architekturu z/Architecture, kanálový subsystém, hardwarové akcelerátory (CPACF, Telum AIU, zIIP, IFL) a PR/SM hypervizor. Ukázala, jak tato modularita a redundance zajišťují provoz bez výpadků i při extrémních zátěžích.

Kapitola zaměřená na COBOL, JCL, Rexx a Python demonstrovala kontrast mezi klasickými dávkovými úlohami a flexibilitou moderních skriptů. Ukázalo se, že využitím Pythonu lze zpřístupnit vývoj aplikací, které poběží na mainframech i pro mladší generaci vývojářů, aniž by se ze začátku museli učit dnes už skoro historické programovací jazyky. Pythonem jde také zjednodušit správu dat, automatizaci a integraci s nástroji DevOps, ovšem výkonnostní rozdíl musí být více prostudován.

Součástí práce byla i implementace praktických úloh, které byly testovány na mainframe prostředí, díky přístupu skrze platformu IBM Z Xplore a správné fungování bylo ověřeno externím specialistou ze společnosti Kyndryl. Úlohy jsou zaměřeny na klasické administrátorské úkony a do určité míry zjednodušeny pro lepší čitelnost zájemcům, kteří s tímto prostředím a zejména jazyky jako COBOL a JCL zatím nepřišli do styku. Nicméně celkový soubor úloh poskytuje dobrý základ pro případné využití k tvorbě komplexnějších programů nebo navázáním např. podrobnějším výkonnostním benchmarkem tradičních vs. moderních úloh, integrací AI akcelerátoru, nebo ukázkou, jak by šel vývoj aplikací propojit s DevOps a navrhnout a implementovat CI/CD pipeline pro automatizované testování a nasazení. Z pohledu výkonnostního benchmarku by bylo i zajímavé zjistit, jak fungují API dotazy pro datasety z Pythonu a navrhnout další pomocné funkce a optimalizovat stávající, které byly rozebrány v předchozí kapitole a bylo zjištěno, že určité funkce jsou poměrně časově náročné. Nicméně pro tato rozšíření by bylo potřeba speciální přístup k IBM Z mainframu.

Ve srovnání s bakalářskou prací pana Vláčila [54], která se soustředí primárně na rozhraní pro vzdálené ovládání mainframu, tato práce poskytuje širší teoretický i praktický kontext včetně rozboru jednotlivých komponent IBM Z, subsystémů a používaných jazyků. Diplomová práce pana Vašáka obsahuje tutoriál [51], jenž je zaměřen čistě na výuku Assembleru a ISPF. Bakalářská práce pana Valáška [50] je strukturou nejvíce podobná mé práci a obsahuje podrobnější rozbor jazyka JCL, nicméně se nezaměřuje na další používané ja-

zyky na mainframech. Technická studie implementace paměti z/OS [86] poskytuje hluboký vhled do správy paměti, což tahle práce nepokrývá, ale jde o velice přehledný rozbor.

Díky kombinaci teoretického rozboru a praktických příkladů práce přispěla k hlubšímu pochopení síly mainframových platforem i jejich současného uplatnění v kritických informačních systémech a zvláště, proč tato platforma ještě zdaleka není pouze přežitek moderní doby. Z pohledu budoucnosti je zásadní, že se platforma neustále otevírá. Kombinace tradiční spolehlivosti a nových možností AI/ML na čipu (Telum II), kvantově bezpečné kryptografie atd. zajistí, že mainframe bude i v příštích desetiletích klíčovým pilířem enterprise architektury. Výzvou pro další generace vývojářů a administrátorů bude plně využít potenciál této platformy – nadále ji modernizovat, automatizovat a přizpůsobovat vznikajícím trendům v cloud computingu a datové analýze.

Literatura

[1] Computer History Museum. *Timeline of Computer History* [online]. Computer History Museum © 2025. Dostupné z: https://www.computerhistory.org/timeline/computers/. [cit. 2025-01-26].

[2] Česká správa sociálního zabezpečení. *Zpráva o činnosti* [online]. ČSSZ, 2015. Dostupné z: https://www.cssz.cz/documents/20143/99593/2015_ZoC.pdf/29e06ff2-1ac8-75dd-7e2b-52ea877b7acd. [cit. 2025-01-26].

[3] Das, T. Why Is Mainframe Still Relevant and Thriving in 2022? *Planet Mainframe* [online], Dec 20, 2022. Dostupné z: https://planetmainframe.com/2022/12/relevance-of-mainframe/. [cit. 2025-03-09].

[4] DiDio, L. IBM z16 and Power10 Deliver Highest Reliability Among Mainstream Servers for 15th Consecutive Year. *TechChannel* [online], July 5, 2023. Dostupné z: https://techchannel.com/backup-and-recovery/ibm-z16-and-power10-deliver-highest-reliability-among-mainstream-servers-for-15th-consecutive-year/. [cit. 2025-03-09].

[5] Digitalisation World. Unisys boosts power and flexibility for ClearPath clients. *Digitalisation World* [online], 2015. Dostupné z: https://digitalisationworld.com/news/34021/unisys-boosts-power-and-flexibility-for-clearpath-clients. [cit. 2025-03-10].

[6] Ebbers, M.; Kettner, J.; O'Brien, W.; Ogden, B. *Introduction to the New Mainframe: z/OS Basics* [online]. © Copyright International Business Machines Corporation, březen 2011. ISBN 0738435341. Dostupné z: https://www.redbooks.ibm.com/redbooks/pdfs/sg246366.pdf. [cit. 2025-01-27].

[7] Eviden. *BullSequana M* [online]. Copyright © Eviden SAS 2025. Dostupné z: https://eviden.com/solutions/advanced-computing/business-computing/bullsequana-m/. [cit. 2025-03-14].

[8] Fremery, R. de. The ENIAC Legacy: How a 1940s Invention Shaped Modern Computing. *Lifewire* [online], November 28, 2024. Dostupné z: https://www.lifewire.com/the-eniac-legacy-8749699. [cit. 2025-01-26].

[9] Fujitsu. *GlobalServer – MSP Operating System* [online]. Dostupné z: https://www.fujitsu.com/global/imagesgig5/msp.pdf. [cit. 2025-03-12].

[10] Fujitsu. *GlobalServer – OSIV/XSP* [online]. Dostupné z: https://www.fujitsu.com/global/imagesgig5/xsp.pdf. [cit. 2025-03-12].

[11] Fujitsu. *GS21 3400* [online]. Copyright © 1995 – 2021 FUJITSU. Dostupné z: https://www.fujitsu.com/global/products/computing/servers/mainframe/globalserver/lineup/gs21-3400/. [cit. 2025-03-11].

[12] Fujitsu. *Mainframes FUJITSU Server GS21* [online]. Copyright © 1995 – 2025 Fujitsu. Dostupné z: https://www.fujitsu.com/global/products/computing/servers/mainframe/globalserver/. [cit. 2025-03-11].

[13] Fujitsu. *Fujitsu Launches New GS21 Mainframe Models Supporting Next-Generation Mission-Critical Systems* [online]. Tokyo, April 17, 2018. Dostupné z: https://www.fujitsu.com/global/about/resources/news/press-releases/2018/0417-01.html. [cit. 2025-03-11].

[14] HG Insights. *HPE NonStop Systems* [online]. Copyright © 2025. Dostupné z: https://discovery.hgdata.com/product/hpe-nonstop-systems. [cit. 2025-03-10].

[15] HPE. *HPE NonStop family of systems* [online]. Copyright © 2025. Dostupné z: https://www.hpe.com/us/en/compute/nonstop-servers.html. [cit. 2025-03-10].

[16] HPE. *NonStop Systems* [online]. Copyright © 2025. Dostupné z: https://buy.hpe.com/in/en/compute/nonstop-systems/c/1013269759. [cit. 2025-03-10].

[17] IBM. *Enterprise COBOL for z/OS 6.4 Programming Guide* [online]. Copyright © International Business Machines Corporation 1991, 2024. Dostupné z: https://www.ibm.com/docs/en/SS6SG3_6.4.0/pdf/pgmvs.pdf. [cit. 2025-03-24].

[18] IBM. *The IBM 700 Series* [online]. Dostupné z: https://www.ibm.com/history/700. [cit. 2025-01-26].

[19] IBM. *IBM Developer for z/OS Enterprise Edition (IDzEE)* [online]. Dostupné z: https://www.ibm.com/products/developer-for-zos. [cit. 2025-03-31].

[20] IBM. *IBM Open Enterprise SDK for Python* [online]. Dostupné z: https://www.ibm.com/docs/en/python-zos/3.13. [cit. 2025-03-09].

[21] IBM. *IBM Open Enterprise SDK for Python* [online]. Dostupné z: https://www.ibm.com/products/open-enterprise-python-zos. [cit. 2025-03-27].

[22] IBM. *IBM PCIe Cryptographic Coprocessor* [online]. Dostupné z: https://www.ibm.com/products/pcie-cryptographic-coprocessor. [cit. 2025-03-10].

[23] IBM. *The IBM System/360* [online]. Dostupné z: https://www.ibm.com/history/system-360. [cit. 2025-01-26].

[24] IBM. *IBM z/OS Management Facility (z/OSMF)* [online]. Dostupné z: https://www.ibm.com/products/zos/management-facility. [cit. 2025-03-30].

[25] IBM. *Security in IBM Z Systems* [online]. Dostupné z: https://www.ibm.com/z/security. [cit. 2025-03-09].

[26] IBM. *Mainframe operating system: Linux for System z* [online]. Copyright © 1990, 2010. Dostupné z: https://www.ibm.com/docs/en/zos-basic-skills?topic=systems-mainframe-operating-system-linux-system-z. [cit. 2025-03-10].

[27] IBM. *What is JES?* [online]. Copyright © 1990, 2010. Dostupné z: https://www.ibm.com/docs/en/zos-basic-skills?topic=jobs-what-is-jes. [cit. 2025-03-10].

[28] IBM. *Z/OS Basic Skills* [online]. Copyright © 2001, 2010. Dostupné z: https://www.ibm.com/docs/en/zos-basic-skills. [cit. 2025-01-27].

[29] IBM. *IBM Z Open Editor* [online]. Copyright © 2021. Dostupné z: https://ibm.github.io/zopeneditor-about/. [cit. 2025-03-30].

[30] IBM. *Cryptographic hardware features* [online]. Copyright © 2023, 2024. Dostupné z: https://www.ibm.com/docs/en/zos/3.1.0?topic=features-cryptographic-hardware. [cit. 2025-03-10].

[31] IBM. *Z/OS 3.1.0 IBM Documentation* [online]. Copyright © 2023, 2024. Dostupné z: https://www.ibm.com/docs/en/zos/3.1.0. [cit. 2025-03-09].

[32] IBM. *Db2 for z/OS* [online]. Copyright © 2024. Dostupné z: https://www.ibm.com/docs/en/db2-for-zos/12?topic=zos-tools-ides-developing-db2-applications. [cit. 2025-03-30].

[33] Jacobi, C.; Tzortzatos, E. *Telum II* [online]. IBM, 26 August 2024. Dostupné z: https://www.ibm.com/new/announcements/telum-ii. [cit. 2025-03-09].

[34] Maintec. *MAINFRAMES IN HEALTHCARE* [online]. Maintec Technologies Inc., December 3, 2020. Dostupné z: https://maintec.com/mainframes-in-healthcare/. [cit. 2025-03-09].

[35] Maintec. *What is CICS in Mainframe?* [online]. Maintec Technologies Inc., July 15, 2024. Dostupné z: https://maintec.com/what-is-cics-in-mainframe/. [cit. 2025-03-09].

[36] Miller, S. SUSE and IBM: Bringing the Mainframe to the Masses. *SUSE* [online], April 29, 2024. Dostupné z: https://www.suse.com/c/suse-and-ibm-bringing-the-mainframe-to-the-masses/. [cit. 2025-03-10].

[37] Mulholland, S. WAL-MART TO SPEND MILLIONS ON NEW TECH. *Supermarket news* [online], October 29, 2001. Dostupné z: https://www.supermarketnews.com/independents-regional-grocers/wal-mart-to-spend-millions-on-new-tech. [cit. 2025-03-09].

[38] Open Mainframe. *Open Mainframe Project Zowe* [online]. Dostupné z: https://www.zowe.org/. [cit. 2025-03-30].

[39] Programmers.io Team. *IBM Z Operating Systems* [online]. Programmers.io, April 6, 2023. Dostupné z: https://programmers.io/blog/ibmz-operating-systems/. [cit. 2025-03-09].

[40] Repholz, K. *IBM Z15 September 12, 2019 Announcement* [online]. Mainline, September 13th, 2019. Dostupné z: https://mainline.com/ibm-z15-september-12-2019-announcement/. [cit. 2025-03-09].

[41] Robinson, D. Fujitsu confirms end date for mainframe and Unix systems. *The Register* [online], Fri 25 Feb 2022. Dostupné z: https://www.theregister.com/2022/02/25/fujitsu_signposts_the_end_for/. [cit. 2025-03-11].

[42] Rocket Software. *What is COBOL?* [online]. Rocket Software © 2025. Dostupné z: https://www.rocketsoftware.com/en-us/insights/what-is-cobol. [cit. 2025-04-18].

[43] Shah, A. Unisys replacing decades-old mainframe processor with x86 chips. *Computerworld* [online], Jun 17, 2014. Dostupné z: https://www.computerworld.com/article/1524927/unisys-replacing-decades-old-mainframe-processor-with-x86-chips.html. [cit. 2025-03-10].

[44] Susnjara, S.; Smalley, I. *What is a mainframe?* [online]. 8 April 2024. Dostupné z: https://www.ibm.com/think/topics/mainframe. [cit. 2025-01-27].

[45] Thompson, J. How Unisys Transitioned from Proprietary to Open Architecture. *AIwire* [online], July 28, 2015. Dostupné z: https://www.aiwire.net/2015/07/28/how-unisys-transitioned-from-proprietary-to-open-architecture/. [cit. 2025-03-10].

[46] Tišnovský, P. *Programovací jazyky používané na mainframech* [online]. Root, 8. prosince 2009. Dostupné z: https://www.root.cz/clanky/programovaci-jazyky-pouzivane-na-mainframech/. [cit. 2025-01-27].

[47] Tsai, M. IBM Launches Next-Generation Mainframe for Midsize Customers. *Marketwired* [online], 20 Octubre 2008. Dostupné z: https://mx.advfn.com/bolsa-de-valores/NYSE/IBM/noticias/28838552/ibm-launches-next-generation-mainframe-for-midsize. [cit. 2025-03-09].

[48] Unisys. *ClearPath® Solutions and Services* [online]. Unisys Corporation. Dostupné z: https://www.unisys.com/siteassets/collateral/ebook/eb-clearpath-foward-20240122.pdf. [cit. 2025-03-10].

[49] Urban Institute. *What technology does the IRS use?* [online]. Urban Institute, Brookings Institution, January 2024. Dostupné z: https://taxpolicycenter.org/briefing-book/what-technology-does-irs-use. [cit. 2025-03-09].

[50] Valášek, P. *Praktické využití systému mainframe* [online]. Praha, CZ, 2009. Bakalářská práce. Vysoká škola ekonomická v Praze. Dostupné z: https://insis.vse.cz/zp/15854/podrobnosti. [cit. 2025-05-04].

[51] Vašák, M. *Preparation of a hands-on tutorial teaching basics of Mainframe ISPF programming* [online]. Praha, CZ, 2011. Diplomová práce. Vysoká škola ekonomická v Praze. Dostupné z: https://insis.vse.cz/zp/24609/podrobnosti. [cit. 2025-05-04].

[52] Vellanki, S. A journey to tackle legacy code in online travel. *Sabre* [online], January 30, 2024. Dostupné z: https://www.sabre.com/insights/a-journey-to-tackle-legacy-code-in-online-travel/. [cit. 2025-03-09].

[53] Venkat, A. COBOL vs. Python – A sort battle. *Analytics Vidhya* [online], Jul 15, 2020. Dostupné z: https://medium.com/analytics-vidhya/cobol-vs-python-a-sort-battle-b23bc1d5fcf4. [cit. 2025-05-09].

[54] Vláčil, R. *Vývoj rozhraní pro vzdálené ovládání systému mainframe* [online]. Praha, CZ, 2007. Bakalářská práce. České vysoké učení technické v Praze. Dostupné z: https://kmlinux.fjfi.cvut.cz/~oberhtom/studenti/07-vlacil-radek-remote-control-of-mainframe.pdf. [cit. 2025-05-04].

[55] White, B.; Doll, J.; Lascu, O.; Minin, A.; Oughton, P. et al. *IBM z17 Technical Introduction* [online]. © Copyright International Business Machines Corporation, duben 2025. ISBN 0738462098. Dostupné z: https://www.redbooks.ibm.com/redbooks/pdfs/sg248580.pdf. [cit. 2025-04-18].

[56] Wikipedia contributors. *Atos* [online]. Wikipedia, The Free Encyclopedia. Dostupné z: https://en.wikipedia.org/wiki/Atos. [cit. 2025-03-14].

[57] Wikipedia contributors. *Burroughs Large Systems* [online]. Wikipedia, The Free Encyclopedia. Dostupné z: https://en.wikipedia.org/wiki/Burroughs_Large_Systems. [cit. 2025-03-15].

[58] Wikipedia contributors. *Burroughs MCP* [online]. Wikipedia, The Free Encyclopedia. Dostupné z: https://en.wikipedia.org/wiki/Burroughs_MCP. [cit. 2025-03-10].

[59] Wikipedia contributors. *COBOL* [online]. Wikipedia, The Free Encyclopedia. Dostupné z: https://en.wikipedia.org/wiki/COBOL. [cit. 2025-03-24].

[60] Wikipedia contributors. *General Comprehensive Operating System* [online]. Wikipedia, The Free Encyclopedia. Dostupné z: https://en.wikipedia.org/wiki/General_Comprehensive_Operating_System. [cit. 2025-03-14].

[61] Wikipedia contributors. *IBM 1401* [online]. Wikipedia, The Free Encyclopedia. Dostupné z: https://en.wikipedia.org/wiki/IBM_1401. [cit. 2025-03-15].

[62] Wikipedia contributors. *IBM 701* [online]. Wikipedia, The Free Encyclopedia. Dostupné z: https://en.wikipedia.org/wiki/IBM_701. [cit. 2025-03-15].

[63] Wikipedia contributors. *IBM 7030 Stretch* [online]. Wikipedia, The Free Encyclopedia. Dostupné z: https://en.wikipedia.org/wiki/IBM_7030_Stretch. [cit. 2025-03-15].

[64] Wikipedia contributors. *IBM 704* [online]. Wikipedia, The Free Encyclopedia. Dostupné z: https://en.wikipedia.org/wiki/IBM_704. [cit. 2025-03-15].

[65] Wikipedia contributors. *IBM Enterprise Systems Architecture* [online]. Wikipedia, The Free Encyclopedia. Dostupné z: https://en.wikipedia.org/wiki/IBM_Enterprise_Systems_Architecture. [cit. 2025-03-15].

[66] Wikipedia contributors. *IBM mainframe* [online]. Wikipedia, The Free Encyclopedia. Dostupné z: https://en.wikipedia.org/wiki/IBM_mainframe. [cit. 2025-03-15].

[67] Wikipedia contributors. *IBM System/360* [online]. Wikipedia, The Free Encyclopedia. Dostupné z: https://en.wikipedia.org/wiki/IBM_System/360. [cit. 2025-03-15].

[68] Wikipedia contributors. *IBM System/370* [online]. Wikipedia, The Free Encyclopedia. Dostupné z: https://en.wikipedia.org/wiki/IBM_System/370. [cit. 2025-03-15].

[69] Wikipedia contributors. *IBM System/390* [online]. Wikipedia, The Free Encyclopedia. Dostupné z: https://en.wikipedia.org/wiki/IBM_System/390. [cit. 2025-03-15].

[70] Wikipedia contributors. *IBM Telum* [online]. Wikipedia, The Free Encyclopedia. Dostupné z: https://en.wikipedia.org/wiki/IBM_Telum. [cit. 2025-03-09].

[71] Wikipedia contributors. *IBM Z* [online]. Wikipedia, The Free Encyclopedia. Dostupné z: https://en.wikipedia.org/wiki/IBM_Z. [cit. 2025-03-09].

[72] Wikipedia contributors. *Integrated Facility for Linux* [online]. Wikipedia, The Free Encyclopedia. Dostupné z: https://en.wikipedia.org/wiki/Integrated_Facility_for_Linux. [cit. 2025-03-10].

[73] Wikipedia contributors. *Job Control Language* [online]. Wikipedia, The Free Encyclopedia. Dostupné z: https://en.wikipedia.org/wiki/Job_Control_Language. [cit. 2025-03-25].

[74] Wikipedia contributors. *Linux on IBM Z* [online]. Wikipedia, The Free Encyclopedia. Dostupné z: https://en.wikipedia.org/wiki/Linux_on_IBM_Z. [cit. 2025-03-10].

[75] Wikipedia contributors. *Mainframe computer* [online]. Wikipedia, The Free Encyclopedia. Dostupné z: https://en.wikipedia.org/wiki/Mainframe_computer. [cit. 2025-03-09].

[76] Wikipedia contributors. *Memory segmentation* [online]. Wikipedia, The Free Encyclopedia. Dostupné z: https://en.wikipedia.org/wiki/Memory_segmentation. [cit. 2025-03-15].

[77] Wikipedia contributors. *NonStop (server computers)* [online]. Wikipedia, The Free Encyclopedia. Dostupné z: https://en.wikipedia.org/wiki/NonStop_(server_computers). [cit. 2025-03-10].

[78] Wikipedia contributors. *Python (programming language)* [online]. Wikipedia, The Free Encyclopedia. Dostupné z: https://en.wikipedia.org/wiki/Python_(programming_language). [cit. 2025-03-27].

[79] Wikipedia contributors. *Rexx* [online]. Wikipedia, The Free Encyclopedia. Dostupné z: https://en.wikipedia.org/wiki/Rexx. [cit. 2025-03-25].

[80] Wikipedia contributors. *Transaction Processing Facility* [online]. Wikipedia, The Free Encyclopedia. Dostupné z: https://en.wikipedia.org/wiki/Transaction_Processing_Facility. [cit. 2025-03-10].

[81] Wikipedia contributors. *Unisys 2200 Series system architecture* [online]. Wikipedia, The Free Encyclopedia. Dostupné z: https://en.wikipedia.org/wiki/Unisys_2200_Series_system_architecture. [cit. 2025-03-10].

[82] Wikipedia contributors. *UNIVAC 1100/2200 series* [online]. Wikipedia, The Free Encyclopedia. Dostupné z: https://en.wikipedia.org/wiki/UNIVAC_1100/2200_series. [cit. 2025-03-15].

[83] Wikipedia contributors. *Z/Architecture* [online]. Wikipedia, The Free Encyclopedia. Dostupné z: https://en.wikipedia.org/wiki/Z/Architecture. [cit. 2025-03-15].

[84] Wikipedia contributors. *ZIIP* [online]. Wikipedia, The Free Encyclopedia. Dostupné z: https://en.wikipedia.org/wiki/ZIIP. [cit. 2025-03-10].

[85] Williamson, L. Shake it Off: z13 Can Withstand 8.0 Magnitude Earthquake. *Share* [online], April 16, 2015. Dostupné z: https://blog.share.org/Article/TitleLink/shake-it-off-z13-can-withstand-8-magnitude-earthquake. [cit. 2025-03-09].

[86] Yalamanchili, C. M. Technical Insights into the Implementation of z/OS Memory and Address Spaces. *International Journal on Science and Technology* [online], December 4, 2020, sv. 11, s. 1–9. ISSN 2229-7677. Dostupné z: https://doi.org/10.5281/zenodo.14288200. [cit. 2025-05-04].

Příloha A

Obsah odevzdané souboru

```
/
├── text/ ..... Zdrojové soubory technické zprávy
├── implementace/ ..... Zdrojové kódy vytvořených programů
│   ├── CHARTGENERATOR/ ..... Generování grafického výstupu
│   ├── CSVCONVERT/ ..... Převod dat do CSV formátu
│   ├── DATACLEANER/ ..... Čištění a předzpracování dat
│   ├── EMPLOYEESREPORT/ ..... Generování reportů zaměstnanců
│   ├── JCLDEMO/ ..... JCL job pro demonstraci utilit
│   ├── PYTHONDEMOS/ ..... Demonstrační Python skripty
│   ├── SALESREPORT/ ..... Generování prodejních dat
│   ├── TRXREPORT/ ..... Generování transakčních dat
│   │   ├── BACKUP/ ..... Záloha původních vstupních dat
│   │   │   ├── GDG VERSION/ ..... Záloha pomocí generation data group
│   │   │   └── REGULAR/ ..... Zjednodušená záloha
│   │   ├── TRXREPORTNORMAL/ ..... Generování reportu transakcí
│   │   └── TRXREPORTWITHSORT/ ..... Generování reportu transakcí se seřazením
│   └── XMLCONVERT/ ..... Převod dat do XML formátu
└── xnajma03.pdf/ ..... PDF technické zprávy
```