Mainframe i z/OS, część 3 z 9: OS/360 i mityczny osobomiesiąc

Posłuchaj strony

Posłuchaj strony

OS/360 i mityczny osobomiesiąc

Sprzęt był triumfem. Oprogramowanie było katastrofą — i to katastrofą tak pouczającą, że jej opis stał się najważniejszą książką w historii inżynierii oprogramowania.

System, który miał zrobić wszystko

OS/360 miał obsłużyć całą rodzinę: od maszyn z 32 kilobajtami pamięci po największe modele, od zastosowań naukowych po transakcyjne, od jednego zadania na raz po wieloprogramowość. Dostarczono go w trzech wariantach z tej samej bazy kodu:

- **PCP** (Primary Control Program) — jedno zadanie naraz, dla najmniejszych konfiguracji;
- **MFT** (Multiprogramming with a Fixed number of Tasks) — stałe partycje pamięci, początkowo cztery, później do 52;
- **MVT** (Multiprogramming with a Variable number of Tasks) — dynamiczny przydział pamięci i, w konsekwencji, fragmentacja.

Cel pamięciowy 32 kilobajtów nie został osiągnięty — realne minimum produkcyjne wynosiło 44 KB. System liczył około **1,5 miliona instrukcji**. Już w 1963 roku projekt opisywano wewnętrznie jako będący „w rozsypce”; IBM dorzucił tysiąc dodatkowych programistów, a koszty samego oprogramowania przekroczyły w jednym roku budżet zaplanowany na *cały* program System/360.

Brooks i smołowy dół

Kierownikiem tego przedsięwzięcia był **Frederick P. Brooks Jr.** W 1975 roku wydał ***The Mythical Man-Month*** — zbiór esejów o tym, czego się nauczył. Pięćdziesiąt lat później to wciąż najczęściej cytowana książka o zarządzaniu projektami programistycznymi, a jej tezy zestarzały się boleśnie dobrze.

Obraz otwierający książkę to **doły smołowe**: duże systemy programistyczne jak prehistoryczne zwierzęta grzęznące w smole — im mocniej się szamocą, tym głębiej toną. Żadna pojedyncza trudność nie jest śmiertelna; śmiertelne jest to, że jest ich sto naraz.

Najsłynniejsze zdanie — **prawo Brooksa** — brzmi:

> *„Adding manpower to a late software project makes it later.”*
> Dodawanie ludzi do opóźnionego projektu opóźnia go jeszcze bardziej.

Brooks pisał to z pozycji człowieka, który sam to zrobił, dorzucając programistów do OS/360 i patrząc, jak termin ucieka. Wyjaśnienie jest banalnie proste i dlatego bezlitosne: nowy człowiek musi zostać wdrożony przez kogoś, kto już pracuje, a liczba kanałów komunikacyjnych w zespole rośnie kwadratowo.

Drugi kluczowy esej to **„efekt drugiego systemu”** (rozdział 5). Pierwszy system architekta bywa oszczędny i czysty, bo projektant działa ostrożnie i nie wie, co mu wolno. Drugi system jest *najniebezpieczniejszym, jaki kiedykolwiek zaprojektuje* — bo nabiera pewności siebie i upycha w nim wszystkie odłożone wcześniej ulepszenia, opcjonalne funkcje i uogólnienia. Brooks odnosił to wprost do przejścia IBM od prostych monitorów serii 700/7000 do przeładowanego ambicjami OS/360.

Trzeci tekst, **„No Silver Bullet”**, dopisano dopiero w 1986 roku (a do książki włączono w wydaniu jubileuszowym z 1995). Jego teza:

> *„There is no single development, in either technology or management technique, which by itself promises even one order of magnitude improvement within a decade in productivity, in reliability, in simplicity.”*

Brooks dzieli złożoność na **istotową** (wynikającą z natury samego problemu — nieusuwalną) i **przypadkową** (którą inżynierowie sami wytworzyli i mogą usunąć: asembler, długie czasy kompilacji, brak narzędzi). Narzędzia atakują tylko tę drugą. Dlatego każda kolejna „rewolucja produktywności” — programowanie obiektowe, metodyki zwinne, low-code, a dziś generatywna sztuczna inteligencja — daje mniej, niż obiecuje: walczy z przypadkowością, gdy problemem jest istota.

> **Ciekawostka.** To jeden z nielicznych przypadków w historii techniki, gdy porażka okazała się cenniejsza od sukcesu. Sprzęt System/360 zarobił miliardy i dawno zniknął z serwerowni. OS/360 spóźnił się, przekroczył budżet i był krytykowany — ale wyprodukował *książkę*, która czterdzieści lat później kształtuje sposób, w jaki myśli o swojej pracy każdy kierownik projektu w Dolinie Krzemowej. Brooks zmarł w 2022 roku jako laureat Nagrody Turinga.

Wirtualizacja: co wymyślono w Cambridge w 1964 roku

To jest klejnot koronny całej tej opowieści i moment, w którym mainframe wyprzedził resztę branży najbardziej — o **niemal czterdzieści lat**.

Porażka, która uruchomiła projekt

Historia zaczyna się od przegranej. W 1964 roku MIT wybierał komputer pod projekt **Multics** — i wybrał maszynę General Electric, nie IBM. Powód był techniczny: ogłoszony w kwietniu 1964 System/360 **nie miał sprzętowej relokacji adresów**, więc nie nadawał się do poważnego systemu z podziałem czasu. W MIT odebrano to jako zlekceważenie; w IBM jako policzek.

Norm Rasmussen, szef **Cambridge Scientific Center** — placówki IBM ulokowanej dosłownie obok MIT — postanowił odpowiedzieć. Zebrał ludzi z biura łącznikowego przy MIT: inżynierów **Lesa Comeau** i Johna Harmona oraz programistę Rona Brennana.

Robert Creasy wpadł na kluczowy pomysł, jak sam wspominał, *w autobusie MTA z Arlington do Cambridge* pod koniec grudnia 1964. On i Comeau spędzili ostatni tydzień roku „radośnie burząc mózgi".

Pomysł: dajmy każdemu całą maszynę

Idea była tak prosta, że aż radykalna. Zamiast projektować abstrakcyjny interfejs systemu z podziałem czasu, wzięli **podręcznik *System/360 Principles of Operation* jako specyfikację interfejsu użytkownika**. Każdy użytkownik dostaje *kompletną, wierną maszynę wirtualną System/360* — z wszystkimi instrukcjami, wszystkimi urządzeniami, całą semantyką sprzętu.

Konsekwencje tej decyzji są wciąż odczuwalne:

- W wirtualnej maszynie można uruchomić *dowolny* system, który działałby na prawdziwym sprzęcie — łącznie z samym programem sterującym. **CP-67 potrafił działać pod CP-67.** Rekurencyjna wirtualizacja, zademonstrowana w latach 60.

- Izolacja użytkowników jest absolutna, bo granica przebiega tam, gdzie granica sprzętu.
- Nie trzeba modyfikować gościa. Nie ma parawirtualizacji, nie ma sterowników gościa, nie ma kompromisów.

Sprzęt trzeba było dobudować. Zespół zaprojektował **Cambridge Address Translator** — tablicę asocjacyjną do stronicowania — i wspólnie z inżynierami z Poughkeepsie wszczepił ją do zmodyfikowanego **System/360 Model 40**. Jak opisywał Comeau: *„Virtual memory on the 360/40 was achieved by placing a 64-word associative array between the CPU address generation circuits and the memory addressing logic."* Pierwszy **CP-40** obsługiwał 14 maszyn wirtualnych po 256 kilobajtów.

Wewnątrz maszyny wirtualnej działał **CMS** (Cambridge Monitor System) — jednoużytkownikowy system operacyjny autorstwa Johna Harmona, czerpiący z doświadczeń MIT-owskiego CTSS. Podział obowiązków był elegancki: CP zarządza sprzętem i izolacją, CMS daje wygodę pracy jednemu człowiekowi. *Separation of concerns* w praktyce, dekady przed tym, jak to nazwano.

> **Ciekawostka — emulator emulatora.** CP-40 początkowo działał jako *symulacja sprzętu Modelu 67 uruchomiona na Modelu 40* — zanim prawdziwy Model 67 w ogóle istniał. Czyli: maszyny wirtualne emulowane na maszynie emulującej procesor, który jeszcze nie powstał.

Moment, w którym zawodowcy się przekonali

CP-67 (1966–1967) uogólnił projekt na zmienną liczbę maszyn i większe pamięci. Kluczowa scena rozegrała się w MIT Lincoln Laboratory: szef tamtejszego centrum obliczeniowego **Jack Arnow** zobaczył, jak Dick Bayles uruchamia CP i *wszystkie konsole są gotowe w niecałą minutę* — podczas gdy konkurencyjny, oficjalny TSS potrzebował na to około dziesięciu minut. Arnow zażądał CP-67 dla Lincoln. Decyzja, jak pisze Melinda Varian w swojej klasycznej historii VM, „wstrząsnęła całą firmą" — bo oficjalnym produktem IBM był TSS, a klient chciał projektu badawczego. Wymusiła jednak dalsze finansowanie i tym samym uratowała całą linię.

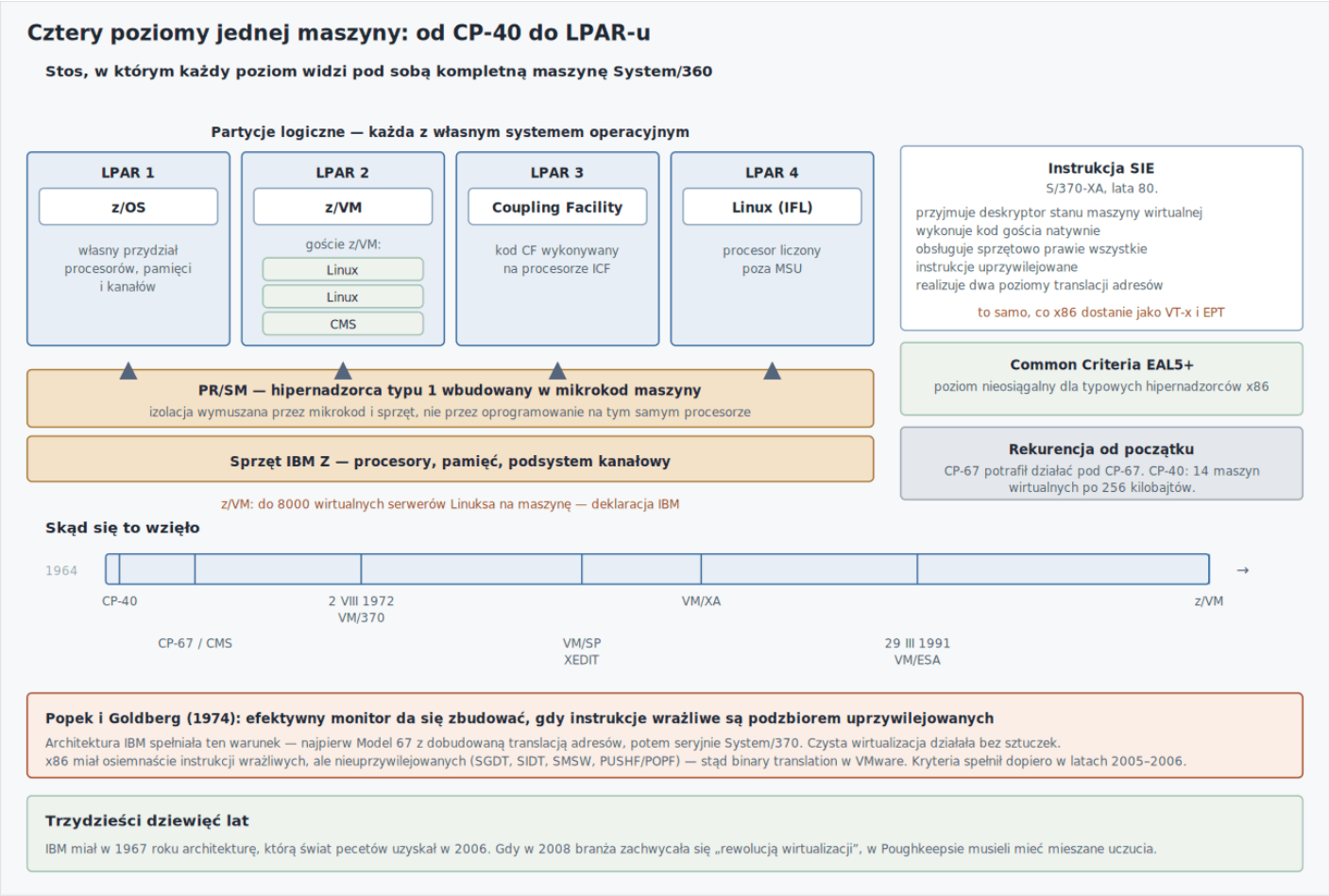
Praca w CMS wygląda zresztą do dziś tak samo, jak wyglądała w 1970 roku:

Trójczłonowa nazwa pliku — *filename filetype filemode* — pochodzi wprost z CTSS-a z MIT i z pomysłów Johna Harmona z 1965 roku. Litera na końcu to identyfikator dysku wirtualnego, a ten jest po prostu plikiem w systemie hosta.

Od CP-67 do z/VM

Po drodze, w latach 80., architektura S/370-XA dostała instrukcję `SIE` (Start Interpretive Execution) — *sprzętowe wsparcie wirtualizacji*. SIE przyjmuje deskryptor stanu maszyny wirtualnej, wykonuje

kod gościa natywnie, obsługuje sprzętowo prawie wszystkie instrukcje uprzywilejowane i realizuje **dwa poziomy translacji adresów** (gość → host). To jest dokładnie to, co x86 dostanie jako VT-x i EPT — dwie dekady później. SIE stała się też fundamentem **PR/SM**, czyli sprzętowych partycji LPAR.



Rysunek 2. *Ten sam mechanizm sprzętowy — SIE i dwa poziomy translacji adresów — obsługuje i partycje logiczne wypalone w mikrokodzie, i maszyny wirtualne z/VM, dlatego wirtualizacja na tej platformie nie jest warstwą doklejoną z zewnątrz, tylko właściwością architektury.*

Puenta: Popek, Goldberg i czterdzieści lat spóźnienia

W 1974 roku **Gerald Popek i Robert Goldberg** opublikowali w *Communications of the ACM* formalne kryteria wirtualizowalności architektury. Monitor maszyn wirtualnych musi spełniać trzy warunki: **równoważność** (program zachowuje się jak na gołym sprzęcie), **kontrolę zasobów** i **wydajność** (dominująca większość instrukcji wykonuje się bezpośrednio, bez interwencji monitora).

Ich **Twierdzenie 1** mówi, że efektywny monitor da się zbudować wtedy, gdy zbiór instrukcji *wrażliwych* jest podzbiorem instrukcji *uprzywilejowanych* — czyli gdy każda instrukcja mogąca

zaburzyć izolację zawsze pułapkuje w trybie użytkownika. Wtedy działa klasyczny *trap-and-emulate*.

I teraz zestawmy fakty:

- **Architektura IBM spełniała ten warunek** — najpierw Model 67 z dobudowaną translacją adresów, a potem seryjnie System/370. Czysta wirtualizacja działała bez sztuczek.
- **x86 go nie spełniał.** Istniało siedemnaście instrukcji wrażliwych, ale nieuprzywilejowanych (Robin i Irvine, USENIX 2000: `SGDT`, `SIDT`, `SMSW`, `PUSHF`/`POPF` w pewnych trybach i inne), które w trybie użytkownika po prostu wykonywały się, zwracając nieprawdziwe dane. Dlatego VMware połączył bezpośrednie wykonanie kodu gościa z dynamicznym translatorem binarnym (*binary translation*), który przepisywał kod jądra gościa w locie, i z segmentacją x86 chroniącą pamięć monitora — szczegóły na stronie **VMware ESX i ESXi**.
- **Kiedy x86 spełnił kryteria Popka i Goldberga — źródła się różnią.** Klasyczną metodą „pułapkuj i emuluj” dało się go wirtualizować dopiero ze sprzętowymi rozszerzeniami: Intel VT-x (14 XI 2005) i AMD-V (2006). Inżynierowie VMware Keith Adams i Ole Agesen piszą jednak w 2006 roku, że x86 „is not classically virtualizable, but it is virtualizable by Popek and Goldberg’s criteria” — dzięki translacji binarnej[1].

Czyli: IBM miał w **1967** roku architekturę, którą świat pecetów dostał w krzemie w latach **2005–2006**. Prawie czterdzieści lat. Gdy w 2008 roku branża zachwycała się „rewolucją wirtualizacji”, w Poughkeepsie musieli mieć mieszane uczucia.

> **Ciekawostka — skąd się wziął „hipernadzorca”.** Słowo **hypervisor** powstało około 1970 roku jako wariant „supervisora” — tradycyjnej nazwy jądra — z przedrostkiem „hyper-” w roli wzmocnienia „super-”. Czyli dosłownie: *nadzorca nadzorców*. Ciekawe, że IBM ukuł je pierwotnie **nie dla CP-67**, tylko dla oprogramowania zarządzającego równoczesnym wykonywaniem OS/360 i emulatora 7090 na maszynie 360/65. Do świata maszyn wirtualnych termin przywędrował później. Podział na hipernadzorcę **typu 1 i typu 2** pochodzi z rozprawy doktorskiej Roberta Goldberga z 1973 roku — a więc kategoria, której używamy dziś do opisu VMware i VirtualBoksa, została zdefiniowana na potrzeby opisu systemów IBM z lat 60.

Jak to wygląda dziś? **z/VM** na pojedynczej maszynie potrafi obsłużyć, według IBM, **do 8000 wirtualnych serwerów Linuksa** albo tysiące kontenerów. Warto tę liczbę traktować jako deklarację producenta — nie podano, co te maszyny miałyby robić — ale rząd wielkości jest realny i jest to jeden z niewielu przypadków, gdzie konsolidacja na mainframie ma sens ekonomiczny.

Rodowody: MVS, z/OS i reszta rodziny

Z OS/360 wyrosły dwie główne linie i kilka bocznych. Warto je znać, bo nazwy krążą do dziś i bywają mylone.

Linia główna: OS/360 → z/OS

Nazwa MVS zasługuje na chwilę uwagi, bo opisuje przełom. Wcześniejsze systemy dawały *jedną* przestrzeń adresową dzieloną przez wszystkie zadania — czyli błąd jednego programu mógł uszkodzić pamięć drugiego. MVS dał **każdemu zadaniu własne 16 megabajtów** widziane od adresu zero, z fragmentem wspólnym dla systemu. To jest to, co dziś jest banalną oczywistością w każdym systemie operacyjnym — a w 1974 roku było nazwą produktu.

W tych samych latach, od 1971 do lutego 1975 roku, IBM prowadził projekt Future System, który miał zastąpić System/370 nową architekturą z pamięcią jednopoziomową; po jego anulowaniu rozwijano dalej System/370, a jego idee przeżyły w Systemie/38 z Rochester — opisuje to strona **IBM i: System/38 i AS/400**.

Warto też odnotować **UNIX System Services**: z/OS ma pełnoprawne, certyfikowane środowisko POSIX-owe, z hierarchicznym systemem plików, powłoką, forkiem i gniazdami. Można na z/OS napisać program w C, skompilować i uruchomić dokładnie tak, jak na Linuksie — a obok, w tej samej maszynie, działa czterdziestoletni COBOL pod CICS-em.

Linia druga: DOS/360 → z/VSE → VSEn

Równolegle istniał system prostszy, dla mniejszych instalacji: **DOS/360** (1965) → DOS/VS → DOS/VSE (1979) → VSE/SP → VSE/ESA → **z/VSE** (2005).

Ta linia ma dziś zaskakujący finał. **z/VSE 6.2 było ostatnim wydaniem IBM, a wsparcie skończyło się 30 września 2023.** IBM nie zaproponował następcy — ale **wylicencjonował kod źródłowy firmie 21st Century Software**, która od 2021 roku rozwija pochodny produkt **VSEn** (wersja 6.4 wydana 15 października 2025, jako jedyna wspierająca sprzęt z17). Czyli: IBM porzucił swój własny system operacyjny, a trzecia firma utrzymuje go przy życiu jako komercyjny fork — sytuacja uderzająco podobna do tego, co VMS Software zrobiło z OpenVMS-em.

Linia trzecia: TPF, czyli ekstremum

TPF (Transaction Processing Facility) to system operacyjny zaprojektowany do jednego zadania: obsługi absurdalnie dużej liczby krótkich transakcji. Rodowód prowadzi od systemu rezerwacji **SABRE** American Airlines, przez **PARS** i **ACP** (Airline Control Program, IBM, prace od 1965, wydanie 1968), do przemianowania na TPF w grudniu 1979, gdy zastosowania wyszły poza lotnictwo.

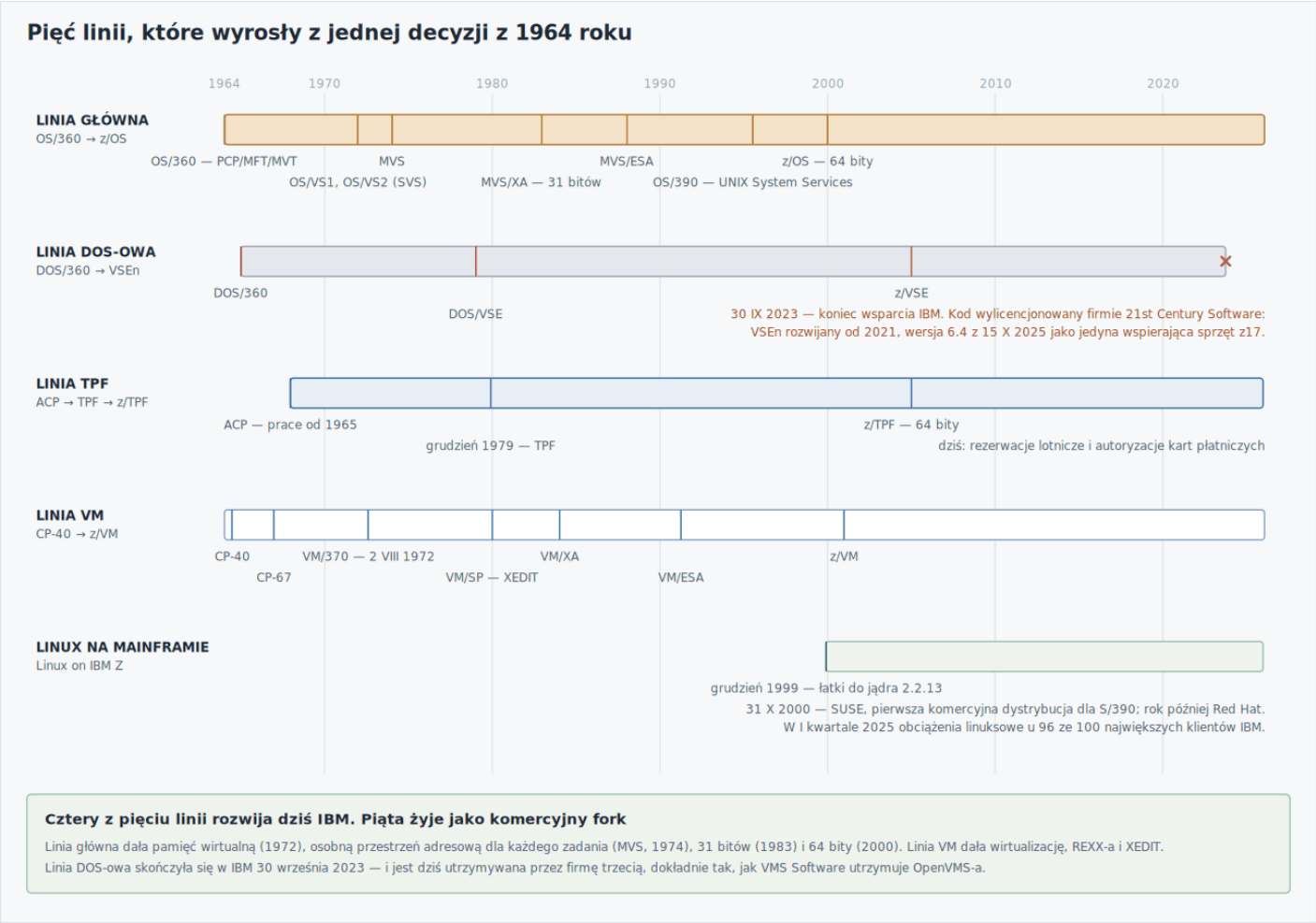
Dziś **z/TPF** (64-bitowy od 2005) obsługuje rezerwacje lotnicze (Sabre, Deltamatic, Japan Airlines) i autoryzacje kart płatniczych — między innymi w **Visa**. Deklarowane parametry robią wrażenie: dziesiątki tysięcy transakcji na sekundę, w niektórych wdrożeniach powyżej 25 000 tps, czasy odpowiedzi rzędu pięciu milisekund, restart po awarii w 30 sekund. *(Te liczby krążą w materiałach branżowych i trudno je niezależnie zweryfikować — traktujmy je jako rząd wielkości, nie jako pomiar.)* Współczesny z/TPF integruje się z REST, MQ, Kafką i MongoDB, co samo w sobie jest ładnym obrazem tego, jak ta platforma się zmienia.

Linia czwarta: pingwin na mainframie

W grudniu 1999 IBM opublikował łatki do jądra Linuksa 2.2.13 pozwalające uruchomić go pod z/VM. W maju 2000 ogłoszono pełne wsparcie produkcyjne, a **31 października 2000** SUSE wydało pierwszą komercyjną dystrybucję dla S/390. Red Hat dołączył rok później.

Dziś **Linux on IBM Z** to poważny kawałek biznesu: według IBM w pierwszym kwartale 2025 roku **96 ze 100 największych klientów mainframe'owych uruchamiało obciążenia linuksowe**, a już w 2020 ponad 35% zainstalowanej mocy procesorowej IBM Z wykonywało Linuksa. Do tego doszły **z/OS Container Extensions** (kontenery Dockera jako część przestrzeni adresowej z/OS) i **Red Hat OpenShift** na Z.

> **Ciekawostka — graffiti za grzywnę.** W 2001 roku IBM promował Linuksa kampanią **„Peace, Love and Linux”**, malując na chodnikach Chicago i San Francisco symbole pokoju, serduszka i pingwina Tuksa. Miasta uznały to za zwykłe graffiti i nałożyły na IBM kary. Korporacja, która trzydzieści lat wcześniej wymagała od pracowników garnituru i białej koszuli, płaciła mandaty za nielegalne malowanie po chodnikach — w imię promocji darmowego systemu operacyjnego na maszynie za kilka milionów dolarów.



Rysunek 3. *Cztery z pięciu linii są dziś rozwijane przez IBM, a piąta — ta po DOS/360 — żyje jako komercyjny fork utrzymywany przez firmę trzecią, co jest dokładnie tym samym losem, który spotkał OpenVMS-a.*

Revision #2
Created 2026-11-03 00:54:56 UTC by Tomasz Konieczny
Updated 2026-11-03 04:16:50 UTC by lektor