Przedmowa

Można śmiało powiedzieć, że na komputerach używanych w latach 70. i 80. ubiegłego wieku każdy dostępny fragment pamięci oraz cykl procesora był na wagę złota. W czasach, gdy masa megabajtu przestrzeni dyskowej była wyrażana w kilogramach, a programiści dysponowali niezwykle znikomą mocą obliczeniową, wyciskano siódme poty ze wszystkich instrukcji procesora, składających się na program, a o każdym bajcie było wiadomo, skąd się wziął i jaką rolę dokładnie odgrywa. Fakt ten był szczególnie widoczny na przykładzie starych gier wideo, które pomimo niewyobrażalnych dziś ograniczeń potrafiły zaskoczyć dopracowaniem oraz grywalnością na maszynach pokroju konsoli Atari 2600. Choć operowała ona na zaledwie 128 bajtach pamięci operacyjnej, to słynne produkcje, takie jak *River Raid*, *Keystone Kapers* czy *Pitfall!* na długo pozostały we wspomnieniach ówczesnych graczy. Z perspektywy lat trudno nie docenić kunsztu tych majstersztyków oraz umiejętności ich autorów.

Na skutek błyskawicznego rozwoju branży sprzętowej oraz powstawania coraz wygodniejszych systemów operacyjnych i środowisk programistycznych, szybszych i efektywniejszych kompilatorów czy w końcu samych języków programowania, lepiej skrojonych pod konkretne zastosowania i niezależnych od docelowej platformy, programowanie z biegiem czasu stało się w pewnym sensie łatwiejsze. Asembler został zamieniony na C, ten na C++, by następnie ustąpić w wielu przypadkach interpretowanym językom wysokiego poziomu, takim jak PHP, Java czy Python. Z drugiej strony postępujące oddalenie programisty od krzemu wykonującego tworzony kod utrudnia dostrzeżenie i zrozumienie pełnego obrazu tego, co dzieje się „pod maską" programu i całego środowiska uruchomieniowego. Na przykład wykonanie prostego skryptu „Hello, World!" napisanego w języku Python na systemie Windows składa się – w dużym uproszczeniu – z uruchomienia interpretera, przetłumaczenia kodu wysokiego poziomu na tzw. kod bajtowy (*bytecode*), obsłużenia każdej instrukcji tego kodu, przejścia do trybu jądra poprzez odpowiednie wywołanie systemowe, rasteryzacji tekstu do bitmapy i przesłania jej do sterownika graficznego, a w końcu na ekran. Ilu współczesnych programistów mogłoby opisać szczegółowo przebieg większości wymienionych etapów wykonania? Prawdopodobnie tylko nieznaczny procent – i choć nie

jest to oczywiście problem sam w sobie, przykład ten ilustruje, jak długa jest lista zależności naszych aplikacji od elementów środowiska wykonawczego, z których istnienia możemy nawet nie zdawać sobie sprawy.

Mimo że zrozumienie wewnętrznych mechanizmów związanych z oprogramowaniem z całą pewnością przydaje się w pracy programisty, muszę przyznać, że główną motywacją wielu badań, które wspólnie z autorem tej książki przeprowadziliśmy podczas naszej 11-letniej znajomości, była zwyczajna ciekawość i rozrywka. Zagłębianie się w coraz to niższe partie środowiska wykonania aplikacji i skryptów, systemów operacyjnych czy w końcu samego procesora z czasem stało się swego rodzaju uzależnieniem pozwalającym na zaspokojenie głodu wiedzy. Szczególnie miło wspominam wczesne lata tego zauroczenia, gdy nierzadko w aktywnej jeszcze sieci IRC wystarczyło, by ktoś rzucił hasło: „Zróbmy konkurs na najmniejszy kompilator ezoterycznego języka Brainfuck!", i już grupa ludzi rozmyślała przez tydzień nad ekstremalnymi optymalizacjami w asemblerze. Wkrótce potem zamiłowanie do niskopoziomowych aspektów informatyki i inżynierii wstecznej w naturalny sposób skierowała Gynvaela i mnie na tor bezpieczeństwa oprogramowania. Dalsze prace pozwoliły na odebranie prestiżowych nagród *Pwnie Award* (w tym wspólnej w 2013 r. w kategorii „Najbardziej Innowacyjne Badania Naukowe"), a stworzenie polskiej drużyny *Dragon Sector* na konkurowanie z najlepszymi zespołami *Security CTF* na świecie oraz zajęcie pierwszego miejsca w globalnym rankingu CTFtime.org w roku 2014.

Książka, którą trzymasz w ręku, jest wynikiem pasji, dociekliwości i wyjątkowego zacięcia dydaktycznego autora, które dotychczas ujawniało się w rozmaitych formach – od licznych, długich na kilka stron ekranowych postów na forach powiązanych z programowaniem i bezpieczeństwem, poprzez inicjatywę IRCowych wykładów (*wyklady.net*), posty na prywatnym blogu, wystąpienia na konferencjach i spotkaniach branżowych, aż po techniczne podcasty publikowane w serwisie YouTube. Dzieło to odbieram osobiście jako swoiste ukoronowanie edukacyjnej działalności autora. O prawdziwej wyjątkowości książki świadczy jednak przede wszystkim fakt, że podchodzi do tematyki inaczej niż podobne opracowania tego typu. Nie znajdziemy tutaj opisu składni lub wstępu do żadnego języka programowania, metodyk tworzenia oprogramowania czy wzorców architektonicznych – tematy te zostały już gruntownie opisane w innych publikacjach, a ich znalezienie nie stanowi problemu. W zamian autor, opierając się na swoim ponad 20-letnim doświadczeniu, skupił się na równie istotnych kwestiach, które bywają z reguły pomijane w innych źródłach, co skutkowało tym, że początkujący programiści byli zmuszeni dochodzić do nich sami na podstawie prób i błędów, tracąc przy tym energię i nabierając niekoniecznie prawidłowych nawyków lub przekonań.

Większość opisanych tutaj mechanizmów, technik i zachowań ma charakter ogólny i posiada zastosowanie niezależnie od użytego języka programowania, dotykając problemu tworzenia poprawnych i przemyślanych programów, a nie wyłącznie kompilującego się kodu. Część I porusza zagadnienia związane z codziennymi czynnościami programistycznymi, ze szczególnym uwzględnieniem konsoli i okna poleceń. Aspekt ten jest ważny głównie dla niedoświadczonych koderów, stawiających swoje pierwsze kroki w środowiskach konsolowych, choć często ignorowany lub traktowany po macoszemu. Część II opisuje najbardziej

fundamentalne koncepty, kierujące współczesnym programowaniem, takie jak podstawy architektury komputerów, binarne kodowanie i operacje na typach liczb naturalnych, całkowitych i pseudorzeczywistych czy reprezentacja znaków i ich ciągów. Rozdziały te są jednymi z moich ulubionych, gdyż bardzo wyraźnie akcentują, jak pewne niskopoziomowe elementy środowiska wykonania przenikają i wpływają na kod wysokiego poziomu – za przykład mogą posłużyć niedokładne (wbrew intuicji) typy zmiennoprzecinkowe w językach takich jak Java czy Python, czy niekompatybilność reprezentacji ciągów tekstowych używanych wewnętrznie i przekazywanych do funkcji biblioteki standardowej, umożliwiająca atak typu *poison null byte* w starszych wersjach PHP. Części III, IV i V omawiają najważniejsze komponenty środowiska uruchomieniowego aplikacji, wprowadzając Czytelnika w świat procesów i wątków (jak również związanych z nimi problemów), interakcji z systemem plików oraz samymi plikami, a także komunikacji pomiędzy programami. Wszystkie rozdziały są okraszone olbrzymią dawką technicznych smaczków, studiów przypadku i przykładowych listingów kodu, tworząc lekturę wypełnioną po brzegi treścią.

Z pełnym przekonaniem polecam ten tytuł każdemu początkującemu i średnio zaawansowanemu programiście, a przede wszystkim tym, którzy zamiast pobieżnych wyjaśnień dostępnych w wielu opracowaniach (np. „zmienna typu short może przechowywać wartości z zakresu od –32768 do 32767") wolą poznać pełną historię. Uważam, że jest to pozycja obowiązkowa na półce każdego zainteresowanego tą dziedziną informatyki, tuż obok książki do nauki konkretnego języka programowania oraz podręcznika do algorytmów i struktur danych; idealnie wypełni ona lukę dotyczącą podstaw środowiska wykonawczego oraz kluczowych mechanizmów używanych w oprogramowaniu w praktycznych zastosowaniach. Życzę Ci, Czytelniku, dobrej zabawy na kolejnych kartach tego tomu, gdyż to właśnie zabawa i fascynacja tematem pomagają z łatwością *Zrozumieć programowanie*.

Mateusz Jurczyk
Senior Software Engineer, Google

Kraków, wrzesień 2015 r.