O niepotrzebnej wiedzy i dwóch zegarach

Paweł Gburzyński

od 2014 r. profesor AFiB (Vistula), w latach 1984–2010 w Kanadzie: 1984–1985 University of Guelph, Computer and Information Science, 1985–2010 University of Alberta, Computing Science (Full Professor od 1991, Professor Emeritus od 2010). Przed wyjazdem z Polski, w latach 1976–1984 pracownik wydziału Matematyki Informatyki i Mechaniki UW (doktorat w 1982 r.). Współzałożyciel Olsonet Communications (Ottawa, Kanada), AppHome (Kalifornia, USA). Twórca i współtwórca oprogramowania, projektant sprzętu. W latach 1980–1984, członek projektu LOGLAN, współtworzył system operacyjny dla komputera MERA-400, brał udział w projektowaniu polskiego komputera (projekt SOLID) z Elżbietą Jezierską, Piotrem Findeisenem i Andrzejem Ziemkiewiczem.

Niektórym się wydaje, że astronauta musi posiadać odwagę supermana i emocje robota. Tymczasem, by zachować spokój w sytuacji stresu i zagrożenia, wystarczy odrobina wiedzy. Nie wyeliminuje ona stresu i podniecenia, lecz nie będziesz czuł się przerażony i bezsilny.

Chris Hadfield, kanadyjski astronauta

Być może niektórzy z czytelników pamiętają jeszcze czasy, kiedy uzyskanie prawa jazdy wymagało zdania egzaminu z tak zwanej budowy. Szmat drogi przebyliśmy od tamtych lat. Jako wczesny nastolatek wiedziałem absolutnie wszystko o naszej Syrence, potrafiłem wyczyścić świece, przedmuchać gaźnik, zdiagnozować większość problemów, a nawet, zakładając dostępność części, usunąć je samodzielnie. Teraz nie próbuję nawet zgadywać, co się dzieje pod maską mojego samochodu. Zakres moich kompetencji ogranicza się do reagowania na jego irytujące werbalne komunikaty i popiskiwania, dolewania płynu do spryskiwaczy, okazjonalne dopompowywanie kół i odstawianie pojazdu na przegląd. Niektórych sygnałów nie rozumiem, lecz podejrzewam, że są niezbyt istotne, gdyż samochód spokojnie jeździ dalej. Coś podobnego dzieje się dziś w praktycznej informatyce.

Gdy pewnego razu poleciłem grupce studentów (w ramach zajęć z telekomunikacji) napisać w ich ulubionym języku program generujący macierz Hadamarda (elemen-

tarne ćwiczenie w programowaniu rekurencyjnym), jeden z nich rozwiązał problem, ściągając z Internetu ponad cztery gigabajty (przeliczyłem) gotowca. Zważywszy, że zadanie da się rozwiązać w kilkunastu linijkach czystego kodu w najzwyklejszym C, narzut na modularność imponuje.

Filozofia rozwijania gotowców jest doskonale zgodna z predylekcją zdobywania wiedzy od góry. Na jej granicy leży całkowity zanik zapotrzebowania na ekspertyzę w zakresie tak rozumianego programowania. Tuż za zakrętem czai się bowiem ultymatywny gotowiec, matka wszystkich gotowców, linia demarkacyjna rozwoju tej „technologii", krótko mówiąc – sztuczna inteligencja. ChatGPT już dziś radośnie wyprodukuje kod do zbudowania macierzy Hadamarda w odpowiedzi na sformułowanie problemu w języku naturalnym, eliminując konieczność ręcznego przeszukiwania Internetu i ściągania niedoskonałych gotowców z archiwum przemijającej epoki. Zadajmy mu inne pytanie: czy eksperci od programowania przez nie-kodowanie będą komukolwiek do czegoś potrzebni za kilka miesięcy, gdy pojawi się kolejna wersja?

W ubiegłym semestrze prowadziłem wykład z systemów operacyjnych w mojej eklektycznej szkole, do której studenci przybywają z różnych stron świata w poszukiwaniu wiedzy nadającej europejską rangę ich rozległym doświadczeniom i talentom. Przed pierwszym wykładem podeszła do mnie grupka uczestników, próbując namówić mnie na zaliczenie im przedmiotu na podstawie rozmaitych kursów i certyfikatów z innych szkół. Według ich wyobrażeń, wiedza obejmująca zakres systemów operacyjnych to umiejętność zainstalowania systemu, skonfigurowania go za pomocą menu lub sekwencji komend odczytanych z podręcznika, uaktualnienia sterowników, wystartowania serwisów itp. Tego typu kwalifikacje mieli już w papierach. Gdy oświadczyłem im, że mój wykład będzie o czymś innym, uprzejmie zakwestionowali jego użyteczność i zażądali, bym wyjaśnił, do czego taka wiedza mogłaby im się przydać. Nie mają przecież zamiaru grzebać we wnętrznościach systemów operacyjnych, specjalizując się w odmiennych dziedzinach, jak nie przymierzając w c y b e r b e z p i e cz e ń s t w i e, które ostatnio stało się podejrzanie popularne, gdzie, jak wiadomo, znajomość systemów operacyjnych to zbyteczny balast. Nie przydaje się ona ani do programowania, ani nawet do kodowania. Może kiedyś było inaczej, ale czasy się zmieniły.

Opowiedziałem im wtedy następującą historię. Pod koniec lata 2000 r. wróciłem na mój kanadyjski uniwersytet z przedłużonego półtorarocznego (oficjalnie naukowego) urlopu, który spędziłem w Kalifornii, próbując z przyjacielem rozkręcić startup. Wróciłem spłukany, bo przedsięwzięcie się nie powiodło. Winę dało się zwalić na czynniki obiektywne, więc nie czyniłem sobie wyrzutów. Czułem się nawet emocjonalnie spełniony i pełen energii, tak że jedynym problemem był brak gotówki, który stawiał mnie w dziwnej sytuacji człowieka zaczynającego od nowa coś, co już kiedyś udało mu się skutecznie zacząć. Był to niby brak przejściowy (posiadałem solidną i bezpieczną posadę na przyzwoitym uniwersytecie), ale niewygodny, gdyż przed wyjazdem pozbyłem się domu i nie miałem teraz środków, by namówić bank na dogodny kredyt.

Przez kilka dni kręciłem się po kampusie bez celu, gdyż trwało jeszcze lato i nie wiedziałem za co się zabrać po długiej przerwie, która dość skutecznie wytrąciła mnie z akademickiego rytmu. Sporo czasu spędzałem w klubie, gdzie przy piwie opowiadałem koleżankom i kolegom o meandrach naszych kalifornijskich interesów. Pewnego razu przysiadł się do mnie znajomy z zaprzyjaźnionego wydziału inżynierii komputerowej i opowiedział, że pewna firma, z którą on współpracuje, ma problem z serwerem, że nie potrafią sobie z tym poradzić, więc może mógłbym ich wysłuchać i coś zasugerować. Następnego dnia czekało na mnie szefostwo i właścicielstwo (mąż i żona) oraz ich główny programista (w tamtych czasach znaczyło to koder). Firma była niewielka i produkowała serwery dla banków, których celem (serwerów, nie banków) było wykrywanie i interpretowanie sygnałów akustycznych (fachowa nazwa brzmi DTMF – Dual-Tone Multi-Frequency), którymi telefonujący klienci wybierali opcje (przyciskając guziki na telefonie). Takie to były czasy, pogaduszki z wirtualnym asystentem miały nadejść za kilkanaście lat.

Opisano mi krótko problem i obiecano dostarczyć sprzęt, dokumentację i kod, gdybym wyraził ochotę zagłębienia się w szczegóły. Serwer funkcjonował pod systemem Linux i był podłączony do szeregu linii telefonicznych typu T1 (taki standard), z których każda kodowała do 24 zagregowanych połączeń telefonicznych (tzw. DS0) obsługiwanych jednocześnie przez serwer. Szczegóły nie są istotne, ale prosta arytmetyka okaże się wkrótce pouczająca. Serwer wykonywał wielowątkową aplikację zaprogramowaną w C++, która czytała cyfrowy sygnał z linii wejściowych, rozbijała go na strumienie indywidualnych połączeń, wiązała je z sesjami klientów, wychwytywała z nich sygnały DTMF i tłumaczyła je na polecenia przekazywane systemowi bankowemu zarządzającemu sesjami i trzymającemu w garści wszystkie krytyczne i delikatne operacje. Problem rozpoznawania sygnałów DTMF był, z oczywistych powodów, wyodrębniony i odizolowany od systemu bankowego.

Serwer potrafił bez trudu obsłużyć wszystkie kanały jednocześnie z dużym zapasem mocy obliczeniowej na okoliczność natłoku klientów i wszelkich przewidywalnych czkawek w procesie interpretowania sygnału. Tyle wynikało z prostych testów i wyliczeń. Struktura aplikacji była klarowna, a jej dynamika dobrze określona. Dla pewności aplikacja monitorowała wydolność serwera, korzystając ze standardowej funkcji systemu Linux (sysinfo) zwracającej, wśród innych parametrów, numeryczną miarę obciążenia. System operacyjny obliczał ją jako średnią liczbę wątków czekających na procesor (chcących się liczyć) zaobserwowanych w ciągu ostatniej minuty. Gdy miara obciążenia przekraczała wartość progową (ustaloną ze sporym marginesem), serwer alarmował administratora.

Problem polegał na tym, że rzeczywiście serwer co jakiś czas alarmował administratora i sygnalizował poważne przeciążenie, ale nie dawało się bezpośrednio dostrzec żadnych jego oznak ani przesłanek. Tendencja nasilała się nieco przy większej liczbie sesji, lecz była z nią słabo skorelowana. Brak zrozumienia problemu utrudniał interpretację sygnalizowanego zagrożenia, gdyż nie dawało się stwierdzić autorytatywnie, czy serwer przypadkiem nie gubi kodów i nie frustruje klientów. Alarm ustępował po jakimś czasie (obciążenie wracało do normy), by powracać w sposób z grubsza cykliczny. Pojawiał się także przy minimalnym (kontrolowanym) obciążeniu w warunkach testowych, choć jakby rzadziej. Cykle bywały mniej lub bardziej regularne. Ich długość wyrażała się dziesiątkami minut. Żmudna analiza kodu i próby zlokalizowania problemu przez ustawianie w programie liczników, asercji itp. nie doprowadziły do żadnych wniosków. Lokalni programiści rozkładali ręce.

Wysłuchałem opowiadania do końca, oświadczyłem, że przyjrzę się problemowi z bliska, zapoznam się z aplikacją oraz z systemem, a potem, gdy rozeznam się w strukturze programu, przeprowadzę własne eksperymenty. Podpisałem umowę o zachowaniu poufności (serwer wykorzystywał unikalny algorytm opracowany przez firmę), poprosiłem o kod, dokumentację i czas do namysłu. Nie zapytano mnie o stawkę, co odebrałem jako dobry znak.

Gdy podszedłem do samochodu, coś mnie tknęło. Telekomunikacja to niby moja specjalność, lecz nie będąc ulepiony z inżynierskiej gliny, nie mam głowy do skrótów, standardów, parametrów numerycznych itd. No, ale przypomniano mi przed chwilą, że T1 to 24 kanały próbkowane 8000 razy (ramek) na sekundę. Interfejs, który to pompuje w system, z pewnością posługuje się buforem, prawdopodobnie definiowanym przez programistę jako całkowita liczba ramek. Powiedzmy, że bufor mieści 10 ramek. Zatem 800 razy na sekundę pojawia się nowy bufor, interfejs generuje przerwanie, sterownik interfejsu się budzi i wątki aplikacji dostają kopa, by przetworzyć nowy blok danych. Potem wątki mają chwilę spokoju do następnego bufora i kolejnego kopa, który nastąpi dokładnie za 0,00125 sekundy. Pracują impulsami. Śpią i budzą się regularnie, zgodnie z sygnałami zegara linii T1.

Mamy więc dwa niezależne zegary taktujące dwa rodzaje aktywności: akcje wątków aplikacji oraz pomiar obciążenia procesora. Nominalna częstotliwość zegara interfejsu T1 ma wspólne dzielniki z częstotliwością zegara systemowego. Jeśli przypadkiem bufor posiada pojemność 10 ramek, zegar systemowy synchronizuje się z co 16. taktem T1. Zegary są niezależne, więc dryfują. Są w miarę dokładne, więc dryfują powoli. Pełen cykl może trwać wiele minut, nawet godzinę, może się zmieniać zależnie od temperatury oraz innych czynników. Tak długo, jak długo takt zegara systemowego wypadać będzie krótko za taktem interfejsu T1, pomiar obciążenia trafi z dużym prawdopodobieństwem na (obiektywnie krótki) czas wysokiej aktywności wątków. Próbkowanie przestanie być miarodajne, podobnie jak sondaż opinii publicznej przeprowadzony wyłącznie wśród uczestników zakończonej przed chwilą antyrządowej demonstracji. Nie ma błędu i nie ma problemu. Jeśli chcemy znać prawdziwe obciążenie serwera, musimy je obliczać inaczej.

Po powrocie do biura zatelefonowałem do firmy. Oświadczyłem, że wiem, na czym polega ich problem i że nie muszę już niczego oglądać. Głupio mi było domagać się wynagrodzenia za tę w sumie banalną usługę (a raczej przysługę), więc poleciłem się jedynie na przyszłość. Firma poprosiła o zaproponowanie miarodajnej metody pomiaru obciążenia systemu, a ze swojej strony zaproponowała ciekawy kontrakt ze stałym miesięcznym uposażeniem (jak to określono, za priorytet w dostępie do mojego czasu) oraz indywidualną rekompensatą za każdy rozwiązany problem. Współpracowałem z nimi w ten sposób przez kilka lat. Następnego dnia ruszyłem rozglądać się za domem. Mała rzecz, a cieszy.

A teraz pora na malutki kawałek wiedzy całkowicie zbędnej współczesnemu programiście aplikacji. Cóż go bowiem może obchodzić, w jaki dokładnie sposób Linux oblicza obciążenie systemu? Zwięzły opis w podręczniku programisty mówi, że jest to średnia liczba wątków oczekujących na procesor pobrana z ostatniej minuty. Średnia, czyli wartość statystyczna. Koncepcyjnie prosta, lecz niemożliwa do policzenia dokładnie. System nie może bowiem poświęcić jej liczeniu zbyt wiele czasu (mając na głowie inne sprawy, jak na przykład wykonywanie aplikacji) i musi ją próbkować i aproksymować minimalnym wysiłkiem. Istnieją okoliczności, gdy jej zgodność z rzeczywistością jest mniej spektakularna niż wyniki politycznych sondaży.

Dokładnie 50 razy na sekundę, w takt standardowego systemowego zegara generującego specjalne przerwanie, system sprawdza, ile wątków chce się liczyć i dodaje ich liczbę do licznika. Pomijając nieistotne detale, przy obliczaniu miary obciążenia system dzieli wartość tego licznika przez 50 x 60 (czyli liczbę taktów zegara w minucie) i twierdzi, że taka jest średnia liczby niecierpliwych wątków z ostatniej minuty. Nie jest to oczywiście żadna obiektywna średnia liczona w ciągłym czasie, tylko średnia wartości obserwowanych w regularnych przedziałach odległych o 1/50 sekundy! Jeśli aktywność wątków aplikacji przypadkiem przejawia podobnie regularny (cykliczny) charakter, pomiary obciążenia mogą z nią korelować.