O niepotrzebnej wiedzy i dwóch zegarach

Paweł Gburzyński

> *„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

Niektórzy z czytelników pamiętają zapewne 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 bezmyślnego reagowania na jego irytujące werbalne komunikaty i popiskiwania obejmując dolewanie 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.

Przez drugą połowę lat 60-tych, w czasach licealnych, naprawiałem znajomym radioodbiorniki i telewizory. Była to działalność ciekawa i dla młodego człowieka niezwykle pouczająca. Moja wiedza na temat elektroniki, której później potrzebowałem w ciekawszych poczynaniach naukowo-przemysłowych, pochodzi niemal w całości z tamtych lat. Okazała się niezbędna i (co ważniejsze) zasadniczo wystarczająca. Formalnie kształciłem się bowiem jako matematyk indoktrynowany w pogardzie dla „drutów”.

Naprawa telewizora w tamtych latach była zajęciem, które z dzisiejszej perspektywy trzeba uznać za niezwykle egzotyczne. Dziś popsuty aparat odsyła się w ramach gwarancji lub wyrzuca na śmietnik. Czasem można przywrócić go do używalności przez wymianę modułu, co się rzadko opłaca, gdyż statystycznie sprzęt starzeje się szybciej niż nadąży się psuć. Opanowanie elektroniki jest przy tym absolutnie zbyteczne, przydaje się natomiast znajomość

przeróżnych nazw, skrótów, typów kabelków, wtyczek, itd., krótko mówiąc wiedza kojarzona z klasyfikacją motyli. Wówczas, uzbrojony w lutownicę, zestaw lamp, oporników oraz kondensatorów (szczególnie ważny był komplet z układu odchylania poziomego, gdzie kondensatory notorycznie ulegały przebiciu), potrafiłem usunąć zasadniczo każde uszkodzenie, które w życiu odbiornika typu „Szmaragd" przytrafiało się co najmniej dwa razy w roku.

Zastanawiam się nad źródłem fascynacji młodego człowieka tamtą dłubaniną. Pojedynczy przypadek mógłby być bez znaczenia, ale pamiętam, że było nas więcej. Może nie wszyscy interesowali się podstawami techniki w takim samym stopniu, ale z pewnością było to w modzie, a już na pewno nie konotowało negatywnych reakcji młodzieżowego, z definicji postępowego, środowiska. Dla mnie była to chyba uchwytna możliwość osiągnięcia nirwany zrozumienia praktycznie wszystkich elementów składających się na funkcjonowanie ówczesnych technologii. Widziałem przewody, którymi płynął prąd, żarzące się lampy, słyszałem brzęczenie transformatora. Nawet jeśli czegoś nie ogarniałem od razu, to dostrzegałem ścieżkę, która doprowadzi mnie do celu, jeśli tylko wykonam kilka kroków ewidentnie znajdujących się w moim zasięgu. Mój pierwszy kontakt z liczbami zespolonymi dokonał się w taki właśnie sposób – chciałem zrozumieć, jak działa filtr.

Zafascynowani techniką młodzi ludzie mojego pokolenia wychowywali się w szacunku dla podejścia „od podstaw" lub „bottom up", jak wolą niektórzy. Świat im tego nie utrudniał, gdyż wszystkie technologiczne obiekty ich zainteresowań razem wzięte były nieporównanie mniej skomplikowane niż współczesna komórka. Łatwo w ten sposób wytłumaczyć, dlaczego dzisiejszy informatyk nie musi w pełni rozumieć funkcjonowania sprzętu czy systemu operacyjnego. Nie jest po prostu w stanie ogarnąć wszystkiego. Tak chyba musi być.

Takiego usprawiedliwienia łatwo jednak nadużyć uczyniwszy z niego pretekst do systemicznej rezygnacji z zainteresowania treścią nabywanych umiejętności i wykonywanych zadań. Poza koniecznością coraz węższej specjalizacji, operuje gdzieś mechanizm wypierający znaczenie z zawartości specjalistycznej wiedzy. Komplikacja technologii jest tu chyba niewinna. Dlaczego wszystkie poradniki pomagające młodemu człowiekowi sprokurować życiorys w celu ubiegania się o pracę zajmują się wyłącznie formą nie zająkując się na temat treści? Żadnemu z nich nie przychodzi do głowy namówić delikwenta do napisania gołej prawdy, jakkolwiek atrakcyjna by ona nie była bez pozamerytorycznych dekoracji. Czemu strategie optymalizowania formy okazują się najbardziej skuteczne już na pierwszym kroku demonstrowania światu kompetencji młodego i rzekomo wykształconego człowieka?

Kilka tygodni temu pojawił się u mnie, na zdalne konsultacje, magistrant, któremu próbowałem pomóc rozwiązać banalny problem związany z telekomunikacją (prowadziłem wówczas specjalistyczny wykład na ten temat). Student cierpiał na nerwowy tik polegający na tym, że kiedykolwiek napisałem na wirtualnej tablicy dwie liczby (dane do zadania), on natychmiast przestawał mnie słuchać, otwierał arkusz Excela, wprowadzał tam moje liczby, po czym klikał myszą na lewo i prawo produkując lawinę diagramów, wykresów i wszelkich excelowych niedorzeczności zupełnie niezwiązanych z problemem. Zrozumiałem po chwili, że tak funkcjonował jego mechanizm obronny przed niewiedzą wykształcony w ramach podejścia „od góry", któremu student ewidentnie hołdował i którego konsekwentne wdrażanie w procesie jego kształcenia doprowadziło do kompletnego zaniku troski o treść. Formą celu była konfiguracja excelowych pól i okienek. Treść przestała istnieć jako element rozwiązania. Obserwowałem granicę współczesnej filozofii przyjaznego nauczania.

Problem sprowadzał się do policzenia jaki procent pierwszej liczby stanowi liczba druga. Gdy w ramach dyskretnej wskazówki objaśniłem studentowi ten fakt i zasugerowałem działanie arytmetyczne, w którym występują obie liczby wejściowe, liczba 100, takoż znaki mnożenia oraz dzielenia, student (nadal operując w Excelu) zaczął generować wszelkie możliwe permutacje tych elementów, niczym ślepiec próbujący ułożyć kostkę Rubika. Dzięki wydatnie mniejszej liczbie możliwości, udało mu się w końcu trafić na rozwiązanie. Na zakończenie naszych konsultacji przeprosił mnie tłumacząc, że telekomunikacja nie jest jego silną stroną, gdyż specjalizuje się w c y b e r b e z p i e c z e ń s t w i e .

Przypadki ekstremalne mają to do siebie, że łatwo je zepchnąć do sfery anegdot, jakich kolekcję posiadł każdy edukator. Nie mam jednak pewności czy opisany przypadek jest faktycznie jaskrawy, czy tylko mnie się takim wydaje, gdyż – przyznaję – tracę wyczucie w tej materii. Można to przecież uznać za nowoczesną metodę obliczania procentów z wykorzystaniem sztucznej inteligencji, czy może … algorytmu genetycznego? Nie jestem pewien, czy żartuję.

Odejście od podstaw w nauczaniu informatyki (i nie tylko) oglądałem osobiście (z Kanady) jako wyraźny przełom w nauczaniu, który być może nie zaznaczył się ostrą kreską w kalendarzu, lecz wystąpił nie mniej klarownie niż wynalazek języka Java. Presja, by powiększyć nabór na studia pierwszego stopnia pojawiła się około roku 1990 i zaowocowała galopującą redukcją wymagań względem poziomu programu nauczania, czego rzecz jasna nie nazywano po imieniu. Zarówno akademia jak i przemysł odetchnęły z ulgą, że nie trzeba już będzie rozumieć wskaźników (pointerów) i uważać na wycieki pamięci, co w tajemniczy

sposób miało przybliżyć arkana skutecznego programowania masom, podnosząc jednocześnie jakość nowych produktów. Preteksty dla przemycenia elementów architektury, systemów operacyjnych i wszelkiej wiedzy na temat tego, co się dzieje naprawdę, gdy komputer wykonuje program zostały skutecznie wyeliminowane czyniąc miejsce na mniej twarde i bardziej przyjazne tematy. Nie było przy tym jasne, dlaczego ktoś, kto nie potrafi zrozumieć, że pamięć komputera nie jest nieskończona i że obiekty w niej przechowywane posiadają adresy, ma zacząć produkować wspaniałe programy, gdy tylko owe niewygodne fakty zostaną przed nim zatajone.

Przemysł się ucieszył, gdyż potrzebował ludzi i chętnie dostosował się do nowych warunków, w których ilość dominowała nad jakością. Odbyło się to przez adopcję nowych technologii tworzenia oprogramowania opartych o „bezpieczne" środowiska, tak by średnio inteligentny programista był w stanie wspomagać pracę zespołu bez pokaleczenia rąk sobie i innym. Spowodowana tym inflacja rozmiarów oprogramowania stymulowała rozwój sprzętu, co jest przecież dobre dla biznesu. Nauczanie od góry okazało się przyjazne i skuteczne, szczególnie dla płacących klientów (czyli studentów pierwszego stopnia), a jak wiadomo klient ma zawsze rację.

To niby naturalne, że zapotrzebowanie na prawdziwych ekspertów (rozumiejących podstawy) nie rozszerza się na wszystkich absolwentów kierunków nominalnie informatycznych, być może włączając nawet tak zwane cyberbezpieczeństwo (w co trudno mi jednak uwierzyć). Nie wszyscy specjaliści od motoryzacji muszą się znać na termodynamice czy mechanice. Niepokojąca jest jednak erozja poziomu, od którego niekompetencja się rozpoczyna, wynikająca zapewne z tych samych mechanizmów, które wykładał Kopernik przywdziawszy czapkę ekonomisty. Nie chodzi tu nawet o fakt, że absolwenci informatyki są niekompetentni w swojej masie. Ich kompetencję ocenia rynek pracy, gdzie, póki co, wszystko wydaje się być w porządku. Współczesne progresywne filozofie każą nam kwestionować pojęcie prawdy, nic więc dziwnego, że definicja kompetencji w informatyce także podlega ewolucji. Taka sytuacja ma jednak prawo budzić niepokój (nie tylko u dinozaurów jak ja), gdyż trend posiada granicę.

Po dziesięcioleciach programowania w przeróżnych (konserwatywnych i archaicznych) środowiskach, postanowiłem niedawno skorzystać z Pythona, któremu dotąd przypatrywałem się z dystansu, z tą samą pobłażliwą niechęcią, z jaką ustępujące pokolenia spoglądają na nowe obyczaje młodzieży. Nie imponowała mi ani składnia, ani koncepcja (na marginesie: moim ulubionym językiem skryptowania jest Tcl/Tk, który pokoleniu Pythona prawdopodobnie kojarzy się z asemblerem). Cóż, czasem człowiek ma ochotę sprawdzić, czy jest jeszcze

dostatecznie młody, by nauczyć się czegoś nowego – jak bardzo poprawi mu się samopoczucie po delikatnym pokolorowaniu posiwiałej brody.

Szybko odkryłem, że podstawową czynnością przy programowaniu w Pythonie (zakładając, że chce się to robić zgodnie z zasadami sztuki) jest wyszukiwanie modułów dostarczających gotowych klas i funkcji na praktycznie dowolne okazje. Każda możliwa operacja, którą ktoś chciałby wykonać na najprostszym obiekcie (napisie, ciągu bajtów, sekwencji bitów), posiada gotowca, który należy wyszukać i zastosować, choć jej zaprogramowanie od podstaw zajęłoby tak zacofanemu programiście jak ja kilka minut. I choć ręka czasem świerzbi, żeby coś prostego zrobić jednak od początku i po swojemu, to ani składnia tego nie ułatwia, ani filozofia tego nie wspiera. Jej celem jest bowiem zastępowanie tradycyjnej koncepcji programowania sklejaniem programów z coraz kompletniejszych gotowców.

Jak bohater „Powrotu z gwiazd" Lema, przyłapuję się na tym, że znaczenie pewnych słów pojmuję inaczej niż otoczenie. Tak więc „programowanie" nie oznacza już pisania programu instrukcja po instrukcji, tylko jego tworzenie z przeróżnych klocków przy pomocy rozmaitych zaawansowanych narzędzi. Podsłuchałem niedawno rozmowę dwóch młodych ludzi w kawiarni (w sąsiedztwie korpo z mojego osiedla), z których jeden twierdził, że programuje, ale nie koduje. Nie jestem pewien, czy przez kodowanie rozumiał to samo co ja.

Gdy pewnego razu poleciłem grupce studentów (także w ramach zajęć z telekomunikacji) napisać w ich ulubionym języku program generujący macierz Hadamarda (elementarne 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. W 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 macierz 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 nowsza 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 wpadająca w zakres systemów operacyjnych to umiejętność zainstalowania systemu, skonfigurowania go przy pomocy 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, większość w owym nieszczęsnym cyberbezpieczeństwie, które ostatnio stało się podejrzanie popularne. Wygląda na to, że dano im do zrozumienia, iż znajomość systemów operacyjnych w cyberbezpieczeństwie to zbyteczny balast: nie przydaje się ani do programowania, ani nawet do kodowania. Może kiedyś było inaczej, ale czasy się zmieniły. Opowiedziałem im wtedy następującą historię.

Pod koniec lata roku 2000 wróciłem na mój kanadyjski uniwersytet z przeciągniętego półtorarocznego (oficjalnie naukowego) urlopu, który spędziłem w Kalifornii próbując z przyjacielem rozkręcić startup. Wróciłem spłukany, gdyż intryga nie powiodła się. W tym podsumowaniu kryje się sporo materiału, no ale ma być o czymś innym. 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 nurtu. 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 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 otrzymałem telefon i umówiłem się na spotkanie. 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 problem w krótkich słowach obiecując 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-ch zagregowanych połączeń telefonicznych (tzw. DS0) obsługiwanych jednocześnie przez serwer. Szczegóły nie są istotne, ale prosta arytmetyka (poniżej poziomu obliczania procentów) 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 wyodrębniony i odizolowany od systemu bankowego, z oczywistych powodów.

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 serwer faktycznie co jakiś czas alarmował administratora i sygnalizował poważne przeciążenie, podczas gdy 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) bez widocznego powodu 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 (koderzy) 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 i dotknąłem klamki, 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.

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 liczba 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!

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-tym 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 kilkunastu minutach, gdy znalazłem się w biurze, 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.