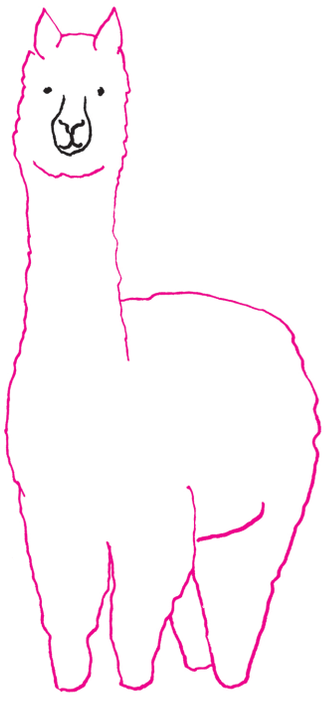
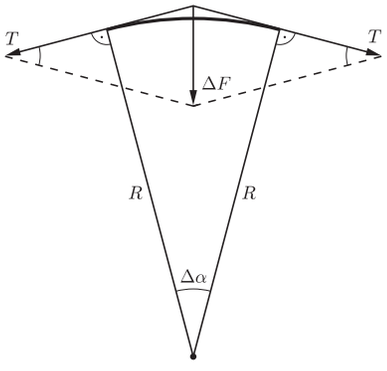
Abstrakcyjne modele w informatyce

*Szymon GUT**



Rozwiązanie zadania F 861.
Każdy mały element kółka o promieniu R zrobionego ze sprężyny przy prędkości kątowej ω porusza się z przyspieszeniem dośrodkowym $a = \omega^2 R$. Jeśli przez $\Delta\alpha$ oznaczyć kąt pomiędzy promieniami łączącymi końce tego elementu z położeniem osi obrotu, to jego masę można zapisać jako $m\Delta\alpha/2\pi$. Siła dośrodkowa działająca na ten element wynosi $\Delta F = T\Delta\alpha$, pochodzi od naciągu sprężyny $T = k(2\pi R - l)$.



Stąd równanie ruchu dla elementu sprężyny ma postać

$$m\Delta\alpha/2\pi\omega^2 R = k(2\pi R - l)\Delta\alpha,$$

a stąd $R = l/(2\pi - m\omega^2/2\pi k)$. Gdy ω rośnie, R także rośnie, dążąc do wartości maksymalnej, którą osiąga dla krytycznej wartości prędkości kątowej

$$\omega_0 = 2\pi\sqrt{k/m}.$$

Sprężyna osiąga wówczas granicę sprężystości (rozprostowuje się), a przy odpowiednio dużej sile rozciągającej ulega zerwaniu.

*Katedra Informatyki,
Akademia Górniczo-Hutnicza

Jesteśmy świadkami szybkiego postępu w informatyce. Pojawiają się nowe języki programowania, biblioteki i platformy programistyczne. Korzystamy z obliczeń wielkiej skali, a swoje dane i aplikacje przenosimy do tzw. chmury. Jak jest możliwy tak szybki rozwój, pomimo dużego skomplikowania systemów komputerowych?

Odpowiedzią jest modne ostatnio pojęcie *wirtualizacji*, które oznacza korzystanie z abstrakcyjnych modeli. Ukrywamy złożoność pewnego skomplikowanego elementu w czarnej skrzynce, która z zewnątrz wygląda całkiem przystępnie. Abstrakcja występuje jednocześnie na wielu poziomach – czarne skrzynki mogą zawierać w sobie kolejne czarne skrzynki, zupełnie jak rosyjskie matrioszki. Przyjrzymy się kolejnym poziomom abstrakcji, zaczynając od najwyższych.

Programista dostaje do dyspozycji wysokopoziomowy język programowania. Może używać pojedynczych operacji, instrukcji warunkowych i pętli, z których buduje procedury. Procedury wraz z danymi, na których operują, tworzą klasy będące definicjami obiektów. Część procedur klasy jest widoczna z zewnątrz i tworzy jej interfejs, a pozostałe są pomocnicze i stanowią jej wewnętrzną implementację. Obiekty mogą operować na innych obiektach, i tak cały program można zbudować jak z klocków.

Złożone aplikacje często tworzy się w oparciu o *architekturę wielowarstwową*. Składają się one wtedy z osobnych warstw, które mają jasno określone zadanie i mogą być niezależnie rozwijane. Typowe warstwy są następujące: warstwa prezentacji stanowiąca interfejs użytkownika, warstwa logiki, w której zawarte są algorytmy i reguły przetwarzania poleceń, oraz warstwa danych, która realizuje dostęp do bazy danych. Każda warstwa korzysta wyłącznie z warstwy znajdującej się bezpośrednio pod nią.

Kolejny poziom abstrakcji jest zapewniany przez system operacyjny. Jego rolą jest udostępnianie programom zwirtualizowanych zasobów komputera: procesora, pamięci operacyjnej oraz urządzeń wejścia/wyjścia.

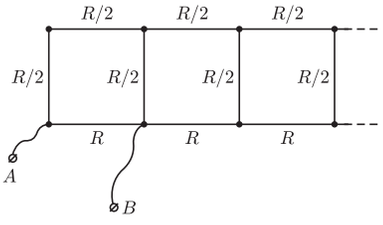
Proces (czyli wykonujący się program) nie jest świadomy obecności pozostałych procesów. Wydaje mu się, że działa na procesorze w sposób ciągły, od uruchomienia do zakończenia. Tymczasem, gdy uruchomionych jest wiele procesów, system operacyjny dzieli dostępny czas procesora, nieustannie przełączając wykonywanie między nimi. Jest to właśnie *wirtualizacja procesora*.

Natomiast *wirtualizacja pamięci operacyjnej* daje procesom wrażenie pracy w ciągłym obszarze pamięci, rozciągającym się przez większość dostępnych adresów. Adresy pamięci widoczne w programie nazywamy wirtualnymi lub logicznymi. Nie są one bezpośrednio używane do adresowania komórek pamięci operacyjnej. System operacyjny za pomocą układów sprzętowych zajmuje się tłumaczeniem adresów logicznych na fizyczne. Dzięki temu może w sposób niezauważalny dla procesów wykonywać wiele pożytecznych funkcji.

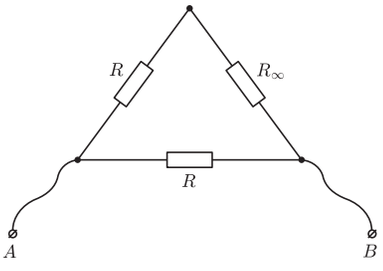
Po pierwsze, różne procesy mają swoje adresy wirtualne odwzorowane na różne adresy fizyczne. Dzięki temu nie mogą uzyskać dostępu do pamięci innych procesów i zakłócać ich pracy. Nazywamy to ochroną pamięci. Jednak procesy mogą również współdzielić pamięć fizyczną. Przykładowo, w części pamięci procesu zapisany jest wykonywany kod programu. Znajduje się w sekcji pamięci tylko do odczytu, więc mamy pewność, że w każdym procesie ten fragment będzie wyglądał tak samo. Nic nie stoi zatem na przeszkodzie, żeby wszystkie te adresy wirtualne odwzorować do tego samego miejsca w fizycznej pamięci operacyjnej, oszczędzając w ten sposób pamięć.

Kolejna technika jest wykorzystywana, gdy zabraknie pamięci operacyjnej. Część zawartości pamięci może być wtedy przeniesiona na dysk bez świadomości procesów. Dane zostaną przywrócone dopiero po ponownym odwołaniu do nich. To ten mechanizm jest odpowiedzialny za spowalnianie komputera, gdy brakuje pamięci operacyjnej. Umożliwia jednak uruchomienie programów zajmujących łącznie więcej pamięci niż dostępne w kościach RAM-u.

Rozwiązanie zadania F 862.
Z symetrii obwodu względem linii A–B wynika, że potencjały punktów symetrycznych względem tej linii są jednakowe. Na tej podstawie możemy rozpatrywany obwód zastąpić obwodem



który z kolei można zwinąć do postaci



i znaleźć oporność pomiędzy punktami A i B:

$$R_{AB} = (R_\infty + R)R/(R_\infty + 2R).$$

Aby znaleźć R_∞, korzystamy z faktu, że oporność nieskończonego obwodu jest taka sama z pierwszym jego ogniwem i bez niego. Stąd

$$\frac{1}{R_\infty} = \frac{1}{R/2} + \frac{1}{R/2 + R_\infty + R},$$

więc

$$R_\infty = \frac{R/2(R_\infty + 3R/2)}{R/2(3R/2 + R/2)},$$

skąd $R_\infty = (\sqrt{21} - 3)R/4$. Podstawiając tę wartość do wzoru na R_{AB} znajdujemy

$$R_{AB} = \frac{\sqrt{21} + 1}{\sqrt{21} + 5} R \approx 0{,}58R.$$

Procesy nie mają również bezpośredniego dostępu do pozostałych zasobów komputera, takich jak dysk. Zamiast tego system operacyjny udostępnia procesom *wirtualny system plików*. Operacje dostępu (otwarcie pliku, odczyt zawartości itp.) implementowane są za pomocą ściśle zdefiniowanych wywołań systemowych, które stanowią interfejs pomiędzy procesami użytkownika a systemem operacyjnym. Rozwiązanie takie ma szereg zalet. System operacyjny utrzymuje kontrolę nad tym, co się dzieje w komputerze, dostarcza spójny model plików niezależny od faktycznie użytego na dysku systemu plików, ma możliwość buforowania danych w pamięci podręcznej oraz zapewnia kompatybilność różnych programów.

Kolejną warstwą abstrakcji jest *model programowy procesora*. Zarówno programy, jak i kod systemu operacyjnego w procesie kompilacji tłumaczone są z języka źródłowego na kod maszynowy. Mają wtedy postać wykonywalną i mogą zostać bezpośrednio uruchomione na procesorze. Program w kodzie maszynowym zapisany jest jako ciąg instrukcji zrozumiałych dla procesora. Struktura tych instrukcji wyspecyfikowana jest właśnie przez model programowy procesora (ang. *Instruction Set Architecture*), który jest abstrakcyjnym opisem procesora, wirtualną maszyną przedstawioną programistom. Określa, jak programy mogą przeprowadzać obliczenia: definiuje typy danych, dostępne instrukcje i rejestry (nieliczne komórki pamięci wbudowane w procesor, do których ładuje się dane z pamięci operacyjnej w celu wykonania obliczeń). Procesory z tym samym modelem programowym mogą wykonywać te same programy. Dzięki temu jeden program może działać na wielu procesorach.

Sprzętowa realizacja modelu programowego przez procesor nazywana jest *mikroarchitekturą*. Procesor nie musi wcale implementować bezpośrednio abstrakcyjnej maszyny zdefiniowanej w modelu programowym. Faktycznie wykonywane instrukcje nie muszą być tymi zapisanymi w programie, a fizyczne rejestry, do których trafiają dane, nie muszą mieć wzajemnego odwzorowania na wirtualne rejestry, na których operuje program. Ważne jest tylko to, żeby efekt działania programu był taki, jakby działała właśnie wirtualna maszyna z modelu programowego.

I tak instrukcje zwykle nie są wykonywane sekwencyjnie, jedna po drugiej. Procesory nazywane superskalarnymi dysponują zwielokrotnionymi jednostkami wykonawczymi, dzięki czemu instrukcje mogą się wykonywać równolegle. W przypadku procesorów firmy Intel już model Pentium był wyposażony w dwie jednostki wykonawcze. Oczywiście, procesor musi wykryć, czy instrukcje przeznaczone do wykonania równoległego są niezależne. Ciąg wzajemnie zależnych instrukcji musi być wykonany sekwencyjnie. W takiej sytuacji na pozostałych jednostkach wykonawczych uruchamia się sąsiednie instrukcje, występujące kawałek dalej w kodzie. Ten mechanizm nazywany jest wykonywaniem poza kolejnością.

Również w obrębie samej jednostki wykonawczej może mieć miejsce równoległość. Cykl wykonania pojedynczej instrukcji dzielony jest na fazy nazywane potokiem. Każda faza wykonuje się na innej specjalizowanej części procesora, dzięki czemu możliwe jest wykonywanie równocześnie różnych faz kolejnych instrukcji. Więcej szczegółów o przetwarzaniu potokowym Czytelnik znajdzie w *Delcie* 8/2012. Problemy pojawiają się, gdy zależności między instrukcjami nie pozwalają na utrzymanie potoku zapełnionego. Przeciwdziała się im, korzystając z szeregu technik, takich jak przewidywanie wyniku instrukcji warunkowych czy przemianowywanie rejestrów.

W dzisiejszych systemach komputerowych abstrakcja występuje na wielu poziomach. Na każdym jednak ma podobne zalety. Umożliwia korzystanie z danego elementu bez znajomości jego wewnętrznych zasad działania. Jednocześnie „pod spodem” mogą w niezauważalny sposób mieć miejsce złożone mechanizmy. Dodatkowo, jasno zdefiniowane przez abstrakcję interfejsy zapewniają kompatybilność różnych implementacji. Dowolny element można zastąpić innym, jeśli tylko komunikuje się z pozostałymi w taki sam sposób. Naturalnie dzieli to system na moduły, z których każdy jest koncepcyjnie prostszy od całości i może być rozwijany niezależnie.