POLITECHNIKA CZĘSTOCHOWSKA

WYDZIAŁ INŻYNIERII MECHANICZNEJ I INFORMATYKI

PRACA DYPLOMOWA MAGISTERSKA

Analiza wydajności technik wirtualizacji, parawirtualizacji i konteneryzacji na podstawie implementacji wybranych algorytmów przetwarzania obrazów

Performance analysis of virtualization, paravirtualization and contenerization techniques based on selected image processing algorithms

Adam Raźniewski

Nr albumu: 129201

Kierunek: Informatyka

Studia: niestacjonarne

Poziom studiów: II

Promotor pracy: dr inż. Mariusz Ciesielski

Praca przyjęta dnia:

Podpis promotora:

Częstochowa 2019

Spis treści

ROZDZIAŁ 1

Wstęp

Wirtualizacja jest techniką znaną od lat. Jej historia sięga lat 60 ubiegłego wieku. Głównym problemem wirtualizacji jest wydzielanie zasobów sprzętowych wirtualizowanym bytom - tak zwanym maszynom wirtualnym. Maszyna wirtualna pozwala na uruchamianie systemu operacyjnego gościa równolegle z bieżącym systemem operacyjnym i przyjmowanie niezależnych od systemu głównego zadań. Cykl życia maszyny wirtualnej może być inny niż głównego systemu operacyjnego - maszyny wirtualne mogą być wyłączane, włączane, usuwane i tworzone.

Technik wirtualizacji jest kilka, wyróżniamy 3 główne obszary - wirtualizacja pełna i wirtualizacja wspomagana sprzętowo, parawirtualizacja, oraz wirtualizacja na poziomie systemu operacyjnego (zwana też konteneryzacją).

Podstawową różnicą pomiędzy tymi technikami jest inaczej realizowany dostęp do fizycznego sprzętu. O ile chociażby w technice wirtualizacji pełnej jest to realizowane na zasadzie translacji binarnej, to w wirtualizacji wspomaganej sprzętowo polegamy na tym, że obecny sprzęt, np. procesory zapewniają interfejsy wirtualizacyjne do wspomagania tego procesu. W takim wypadku maszyna wirtualna ma bezpośredni dostęp do fizycznego procesora pomijając pośrednika, który znacząco wpływa na wydajność wirtualizacji.

Jednym z ważniejszych zagadnień wirtualizacji jest ograniczanie zasobów - dzięki takiemu rozwiązaniu można na przykład z jednego serwera wydzielić N maszyn wirtualnych, gdzie każda będzie zachowywała się jak osobny serwer. Takie maszyny nazywamy wirtualnymi serwerami prywatnymi (ang. Virtual Private Server - VPS). Każda z technik również realizuje je inaczej.

1.1. Motywacja pracy

Motywacją do rozpoznania tego problemu jest poszerzenie wiedzy na temat zagadnień poruszanych w pracy. W internecie możemy przeczytać materiały głównie marketingowe, które mogą nie odzwierciedlać rzeczywistości. Wszystkie techniki wirtualizacji opisane w tej pracy są odpowiedzią na pewien popularny problem - mozliwość wydzielenia zasobów dużej jednostki tak, aby można było ten wydzielony zasób kontrolować, usuwać, oraz skalować kiedy to potrzebne.

Z wirtualizacją mamy do czynienia wszędzie- prawie każdy procesor nowej generacji ma sprzętową technologię wspomagania tego procesu (interfejsy procesora VT-x, AMD-V), a VPS'y są bardzo popularną usługą.

1.2. Cel i zakres pracy

Celem pracy jest przegląd i scharakteryzowanie aktualnie dostępnych systemów wirtualizacji podzielonych na trzy główne techniki wirtualizacji - wirtualizację, parawirtualizację i konteneryzację, oraz przeanalizowanie ich pod względem szybkości przetwarzania obrazów.

Celem pracy jest też przeprowadzenie badań (analiza czasu obliczeń aplikacji z zaimplementowanymi kilkoma algorytmami przetwarzania obrazów) z wykorzystaniem różnych systemów wirtualizacji.

Zakres prac obejmuje:

- analizę literatury dotyczącej wirtualizacji, parawirtualizacji i konteneryzacji,
- scharakteryzowanie systemów wirtualizacji,
- wybranie algorytmów przetwarzania obrazów wraz z ich omówieniem,
- implementację algorytmów w języku Go wraz z implementacją skryptów testowych powłoki,
- przygotowanie i konfiguracja środowisk obliczeniowych - systemów wirtualizowanych,
- przeprowadzenie obliczeń,
- zebranie pomiarowych wyników oraz ich prezentacja na wykresach,
- analizę wyników i opracowanie wniosków.

ROZDZIAŁ 2

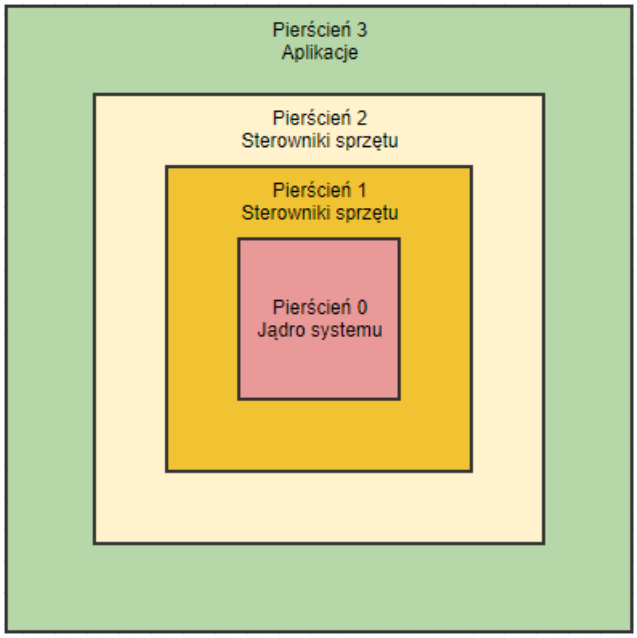
Wprowadzenie

2.1. Domeny ochrony

Każdy komputer to zbiór sprzętu i oprogramowania. Dostęp do każdego zasobu odbywa się za pomocą sensownych operacji [2]. Operacje te mogą wymagać dostępu do zasobów sprzętowych takich jak kamera czy drukarka, ale niebezpiecznym jest dawać ten dostęp dla aplikacji.

Pierścienie ochrony to mechanizm, który pozwala na ograniczenie dostępu aplikacji do tych operacji, które są względnie bezpieczne. Specjalne bramki dostępu pomiędzy pierścieniami pozwalają na to, aby aplikacja mogła skomunikować się ze sprzętem. Hierarchię pierścieni numeruje się od 0 (najbardziej uprzywilejowany) do N, gdzie N jest najmniej uprzywilejowanym pierścieniem.

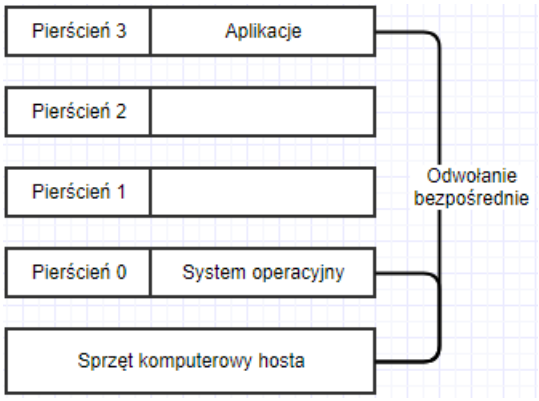
System operacyjny oferuje różne poziomy dostępu do zasobów. Najbardziej uprzywilejowane operacje są numerowane jako zero, najmniej - największym numerem pierścienia. Operacje w pierścieniu zero komunikują się praktycznie bezpośrednio ze sprzętem takim jak CPU czy pamięć. Na rysunku 2.1 ukazano schemat domen ochrony dla architektury x86.



Rysunek 2.1. Domeny ochrony architektury x86 [20]

Istnieją specjalne bramki, które pozwalają na komunikowanie się wyższego pierścienia z niższym. Przykładowo - aplikacje w pierścieniu 3 mogą poprosić o dostęp do sprzętu w domenie pierścienia 1.

Na rysunku 2.2 ukazano uproszczony schemat wykonywania żądań dla systemów bez wirtualizacji.



Rysunek 2.2. Schemat komunikacji w systemie operacyjnym. Bezpośrednie wykonanie żądań systemu operacyjnego względem sprzętu w pierścieniu 0. [20]

Domeny ochrony są ważne z punktu widzenia wirtualizacji, gdyż systemy wirtualizacji mogą zostać umieszczone w różnych pierścieniach, dzięki czemu mogą akcelerować, bądź ograniczać wydajność wirtalizowanego systemu.

2.2. Emulacja

Emulacja to technika pozwalająca na imitowanie innego sprzętu, czy oprogramowania. Kiedy sprzęt jest emulowany - oprogramowanie całkowicie imituje sprzęt. Dzięki temu oprogramowanie korzystające z tego emulowanego sprzętu zachowuje się dokładnie tak jak na sprzęcie, który jest emulowany [16].

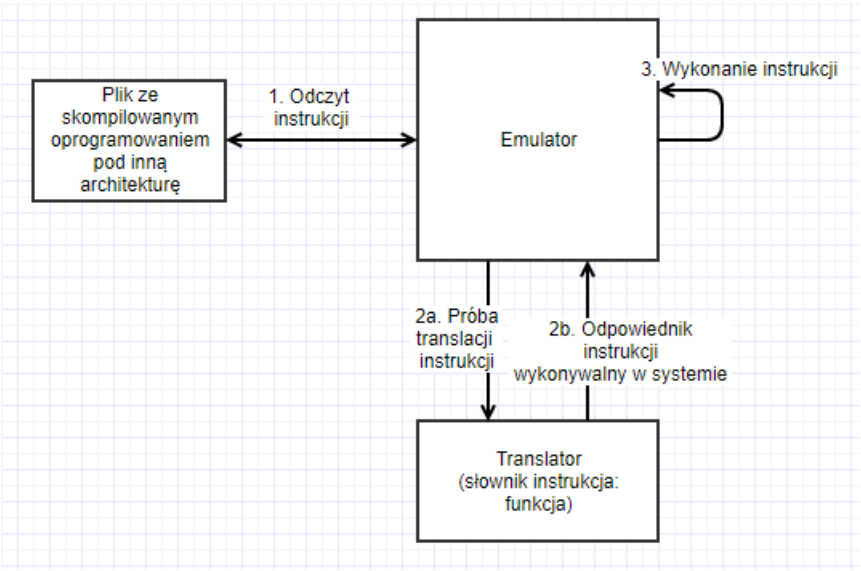
Pozwala to na przykład na emulacje innych architektur oprogramowania, jak ARMv7 na architekturze x86, czy emulowaniu chociażby konsoli GameBoy.

Jest to technika bardzo podobna do wirtualizacji, ale jej zadania są całkowicie różne. Wirtualizacja ma za zadanie wykorzystać nasz sprzęt optymalnie, dzięki takim zagadnieniom jak na przykład współdzielenie sprzętu (wirtualizowany sprzęt może mieć bezpośredni dostęp do procesora, jeżeli procesor ma taką samą architekturę).

Inaczej ma się to w przypadku emulacji - tutaj chodzi o dokładną imitację. Oczywiście ma to

duży narzut wydajnościowy - pełna emulacja to koszt translacji każdego żądania przez translator, oraz wykonanie odpowiedniej instrukcji.

Na przykładzie emulacji systemu CHIP-8 można w prosty sposób zobrazować jak to działa. System CHIP-8 ma 35 kodów operacji (fragment rozkazu przekazywanego do procesora). Tworząc emulator do tego systemu należy utworzyć oprogramowanie, które zawiera tablicę translacji, która ma zapisane działanie dla każdego z dostępnych kodów operacji, ale które da się wykonać na procesie emulatora. Emulator wczytuje oprogramowanie na system CHIP-8 w trybie bajtowym, wykonuje cykl operacji (odczyt instrukcji, translacja instrukcji, wykonanie instrukcji) i powtarza, dopóki nie dostanie kodu instrukcji zamknięcia. Zostało to zobrazowane na rysunku 2.3.



Rysunek 2.3. Schemat procesu emulacji oprogramowania.

2.3. Wirtualizacja pełna

Ta technika wirtualizacji polega na dwóch sposobach - translacji binarnej i bezpośrednim wykonaniu instrukcji. Żądanie wykonania instrukcji jeżeli jest to możliwe - jest wykonywane bezpośrednio na sprzęcie, ale po określeniu czy instrukcja nie narusza bezpieczeństwa [25]. Tak jak w przypadku emulacji - instrukcje są czytane binarnie i translatowane, ale na odpowiednie bezpośrednie instrukcje do sprzętu.

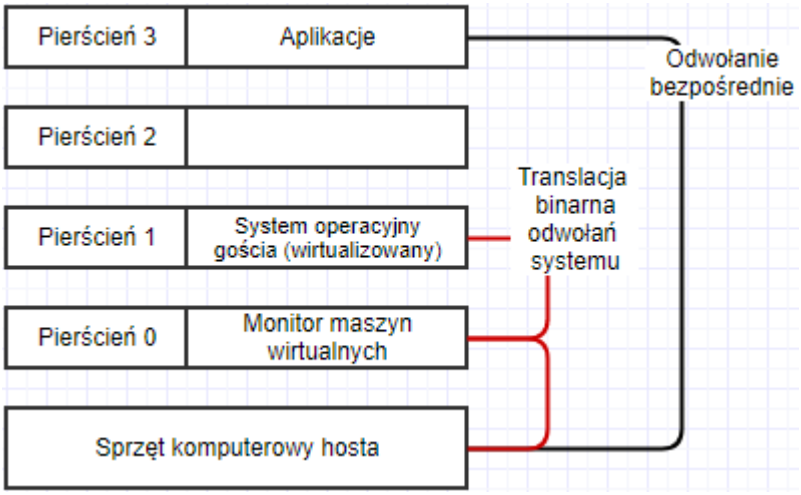
Translacja binarna to technika, która interpretuje instrukcje, które wywołuje obiekt wirtualizowany na docelowe instrukcje, które mogą być aktualnie wykonane na sprzęcie. Jest to pojęcie ważne

w przypadku emulacji i wirtualizacji. W przypadku wirtualizacji jest to zadanie uproszczone - architektura powinna być kompatybilna z architekturą bazową systemu.

System operacyjny gościa jest kompletnie odizolowany od systemu hosta. Nie jest w żaden sposób poinformowany o tym, że jest wirtualizowany i nie potrzebuje żadnej modyfikacji na poziomie jądra czy sprzętu. Wirtualne maszyny stworzone przy pomocy tej techniki mają całkowicie odseparowany system włączając w to urządzenia, pamięć, BIOS czy dyski twarde. Maszyny wirtualne nie są w stanie jednoznacznie określić, że są aktualnie w środowisku wirtualizowanym. Często da się sprawdzić przez określenie oprogramowania BIOS'u maszyny.

Technika pełnej wirtualizacji nie wymaga żadnego wspomagania sprzętowego. Instrukcje są przekazywane do sprzętu podczas działania systemu. a nie bezpośrednio na nim wykonywane.

Rysunek 2.4 ukazuje schemat działania systemu wraz z wirtualizatorem.



Rysunek 2.4. Schemat komunikacji w przypadku pełnej wirtualizacji. Maszyna wirtualna odwołuje się do wirtualizatora, a ten decyduje co z tą instrukcją zrobić.

Open Virtualization Format

Open Virtualization Format (OVF) jest standardem zapisu informacji dotyczącego wyeksportowanej maszyny wirtualnej, dzięki czemu wyeksportowane maszyny wirtualne z takim sposobem zapisu mogą zostać z powodzeniem zaimportowane na zupełnie innym systemie wirtualizacji.

Zawiera w sobie informacje o dodatkowych urządzeniach sieciowych, czy dyskach, nazwie maszyny, czy autorze maszyny wirtualnej. Możliwe do zapisania są również dokładne właściwości wirtualnych kart sieciowych włącznie z adresem MAC.

Format wirtualnego dysku maszyny

Virtual Machine Disk (VMDK) jest to format pliku, który opisuje wirtualny dysk maszyny wirtualnej. System operacyjny zwykle potrzebuje miejsca w którym może przechowywać dane. Główny system operacyjny traktuje dysk gościa po prostu jako plik.

VMDK pozwala na dynamiczne alokowanie dysku - dzięki czemu po przydzieleniu maszynie wirtualnej 10 gigabajtów początkowy rozmiar pliku będzie mniejszy niż rzeczywiście zaalokowany. Rodzi to problemy, kiedy system operacyjny hosta nie ma wolnego miejsca, a miejsce wolne widnieje jeszcze w maszynie wirtualnej - mogą wystąpić wtedy problemy z systemem plików. Jest też możliwość całkowitego zaalokowania dysku od początku, co zupełnie rozwiązuje ten problem.

2.4. Wirtualizacja wspomagana sprzętowo

Wirtualizacja wspomagana sprzętowo to typ wirtualizacji w której to sam sprzęt odpowiada za odpowiednio przetworzone żądania wirtualizowanego sprzętu. Rozwiązanie to polepsza wydajność wirtualizacji, która w tym przypadku może być porównywana do wydajności natywnej. Żądania również są przechwycone, ale w większości przefiltrowane na poziomie samego sprzętu. Po przebadaniu czy żądanie jest w tym kontekście prawidłowe - zostaje na nim wykonane.

Rysunek 2.5 ukazuje schemat działania systemu ze wspomaganą sprzętowo wirtualizacją. Żądania przez maszynę wirtualną zostają odpowiednio przechwycone na poziomie sprzętowym. To podejście nie wymaga binarnej translacji żądań.

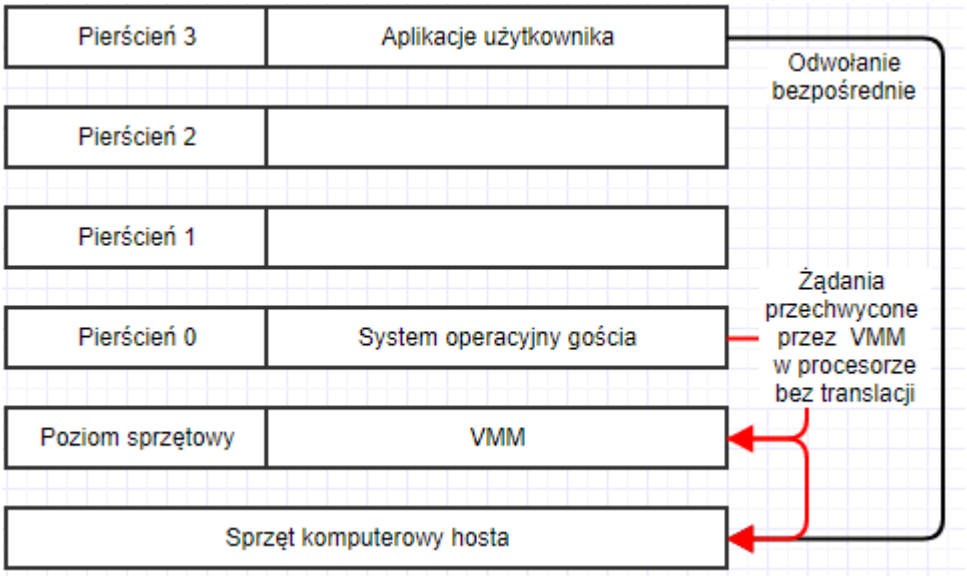
Aktualnie najpopularniejszy sposób wirtualizacji ze względu na możliwie wysoko osiąganą wydajność i popularność interfejsów wspomagania sprzętowego w ówczesnym sprzęcie.

Używanie tej techniki stwarza pewne zagrożenie, gdyż maszyna wirtualna uzyskuje dostęp do teoretycznie bezpiecznego realnego środowiska sprzętowego. W przypadku wykrycia jakiejś usterki, lub luki bezpieczeństwa w oferowanych przez sprzęt interfejsach - maszyna wirtualna mogłaby dostać się do realnego sprzętu i na przykład uzyskać poufne dane.

Dodatek wspomagania wirtualizacji procesora VT-x i AMD-V

VT-x i AMD-V to nazwy interfejsów wspomagania wirtualizacji stworzonych odpowiednio przez firmę Intel i AMD. Obydwa interfejsy służą do wspomagania wirtualizacji przez sam procesor w izolowanym środowisku.

Wspomaganie przez procesor pozwala na pominięcie translacji binarnej i bezpośrednie wykona-



Rysunek 2.5. Schemat komunikacji w przypadku sprzętowo wspomaganej wirtualizacji. Maszyna wirtualna odwołuje się bezpośrednio do sprzętu i tam przez odpowiedni interfejs żądanie zostaje przechwycone i skonsumowane.

nie instrukcji maszyny wirtualnej na procesorze. Jest to możliwe, dzięki temu, że procesor sam nie pozwoli na wykonanie instrukcji uprzywilejowanych przez maszynę. Daje to wielokrotnie szybsze działanie maszyn wirtualnych.

Interfejsy dodają nowe instrukcje wywołania. W przypadku VT-x są to VMPTRLD, VMPTRST, VMCLEAR, VMREAD, VMWRITE, VMCALL, VMLAUNCH, VMRESUME, VMXOFF i VMXON [3], a w przypadku AMD-V - VMRUN, VMCALL, VMSAVE, VMLOAD i VMEXIT [11].

Interfejs wspomagania wirtualizacji urządzeń PCI

Interfejs ten pozwala wirtualnym maszynom dostęp do urządzeń PCI przez IOMMU (jednostka zarządzania pamięcią wejścia wyjścia). Dzięki takiemu rozwiązaniu można przykładowo kartę sieciową wpiąć bezpośrednio do wirtualizowanej jednostki, a nie emulować jej działanie. Przekłada się to jednak na trudność przenoszenia takiej maszyny wirtualnej do innego systemu zarządzania. Przykładem takiego interfejsu jest VT-d opracowany przez firmę Intel i AMD-Vi opracowany przez AMD.

KVM

Kernel-based Virtual Machine jest to interfejs i hipernadzorca wbudowany w jądro systemu operacyjnego. Pozwala na zarządzanie maszynami wirtualnymi. Ujednolica jednocześnie dostęp do

interfejsów wspomagania sprzętowego, dzięki czemu oprogramowanie może być stworzone bez ich rozróżnienia [19]. KVM wymaga do działania jednego z interfejsów wspomagania wirtualizacji procesora (np. VT-x lub AMD-V) do poprawnego działania.

2.5. Systemy wirtualizacji

QEMU

QEMU jest systemem wirtualizacji pozwalającym na wirtualizowanie maszyny w takich technikach jak emulacja, pełna wirtualizacja, wirtualizacja wspomagana sprzętowo i parawirtualizacja. System ten został stworzony jako projekt otwartoźródłowy [5][27].

QEMU pozwala na dodawanie szeregu urządzeń do maszyny wirtualnej[6] - kontrolerów USB, kart sieciowych róznego typu, wirtualnego łącza wideo (np. VGA), dysków, czy też interfejsów audio. Ma też wbudowany interfejs VNC, więc można w każdej chwili podłączyć się do monitora maszyny. Umożliwia też przekierowywanie portu z maszyny wirtualnej do maszyny wirtualnej systemu.

Jedną z najciekawszych funkcji QEMU jest możliwość emulacji architektury ARM (od ARMv7 do ARMv5TEJ), więc można na nim z powodzeniem emulować system Raspbian i testować oprogramowanie stworzone dla Raspberry PI.

VirtualBox

VirtualBox został stworzony przez firmę Innotek GmbH. Od grudnia 2010 roku jest rozpowszechniany na licencji GPLv2.

System ten wspiera wirtualizację wspomaganą sprzętowo, oraz wirtualizację pełną. Jest oferowany z intuicyjnym graficznym interfejsem użytkownika i ma wsparcie dla zunifikowanych formatów wirtualizacji (OVF, VMDK)[17].

VirtualBox ma również wbudowane API, przez które można obsługiwać w popularnych językach programowania. Dzięki temu można stworzyć oprogramowanie, które będzie zarządzało maszynami wirtualnymi.

Wspiera też tak zwane migawki (ang. snapshot) - zapisuje stan maszyny w chwili stworzenia migawki. W dowolnym momencie można przywrócić stan maszyny do wybranej migawki [8].

VirtualBox oferuje też oprogramowanie na maszyny wirtualne, które łączy je z głównym systemem operacyjnym - "Dodatki Gościa"(Guest Additions). Po włączeniu dodatków na systemie gościa

jest możliwe dzielenie się schowkiem z maszyną, lepsze wsparcie dla video, automatyczne dostosowanie rozdzielczości, czy współdzielenie folderów.

VMware Workstation

VMware Workstation wspiera technikę wirtualizacji pełnej i wirtualizacji wspomaganej sprzętowo.

Producent oprogramowania oferuje również wiele innych rozwiązań, które są komplementarne. Na przykład można przeprowadzic integrację z vSphere i zarządzać maszynami wirtualnymi zdalnie [12].

Rozwiązanie to równiez oferuje "Dodatki Gościa", które zapewniają funkcjonalność podobną do VirtualBox'a.

2.6. Parawirtualizacja

W systemach parawirtualizacji systemy operacyjne gościa są specjalnie zmodyfikowane, aby działać dobrze z hipernadzorcą (hypervisor) - specjalnym systemie, który zarządza procesami wirtualizacji. Inną nazwą hipernadzorcy jest VMM (Virtual Machine Monitor). Hipernadzorca w tym wypadku jest swoistą bramą pomiędzy właściwym sprzętem a odwołaniem do niego. Rozdziela odpowiednio zasoby, np. pamięć.

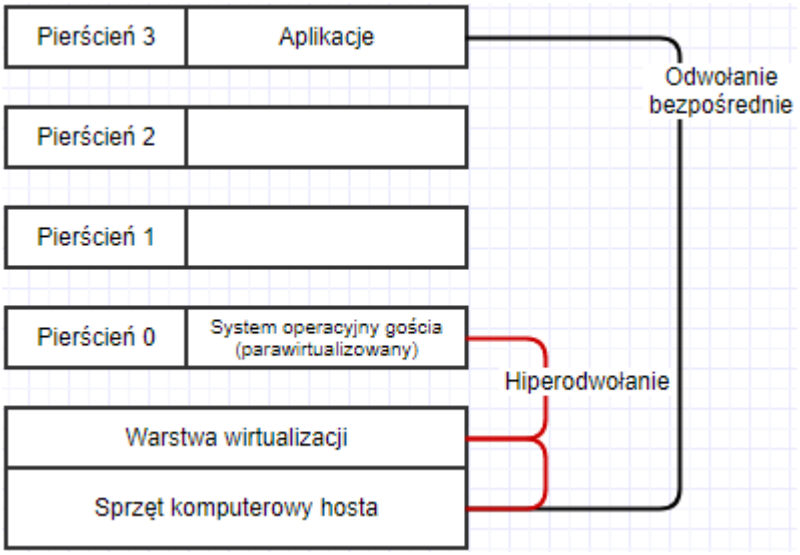
Każda uprzywilejowana operacja (w pireścieniu 0) jest zamieniana na tak zwane hiperodwołanie (hypercall) - odwołanie do hipernadzorcy. Hypervisor odwołuje się wtedy bezpośrednio do sprzętu[9]. Na systemach operacyjnych gości wymaga to odpowiednio zmodyfikowanego jądra - system ma zastąpione instrukcje bezpośrednich odwołań. Na rysunku 2.6 został pokazanych schemat komunikacji w systemie parawirtualizowanym.

Największą różnicą względem systemów wirtualizacji jest to, że system parawirtualizowany ma właściwe do tego celu sterowniki. Zamiast próbować bezpośrednio odwoływać się do sprzętu - odwołuje się od razu do hypervisora, przez co nie musi być to przechwycone przez odpowiedni proces.

2.7. Systemy parawirtualizacji

XenServer

Citrix XenServer jest rozwiązaniem opartym o hipernadzorcę typu 1, co umożliwia uruchamianie maszyn wirtualnych w trybie parawirtualizacji [35][34].



Rysunek 2.6. Na schemacie pokazano komunikację w systemie parawirtualizawanym.

XenServer oferuje wygodny graficzny interfejs użytkownika do obsługi maszyn wirtualnych, przez który odpowiednio możemy je tworzyć, usuwać i modyfikować istniejące. Interfejs ten pozwala też na podgląd maszyn wirtualnych w czasie rzeczywistym.

Jedną z funkcji tego systemu są migracje na żywo (live migrations), które umożliwiają skopiowanie maszyny wirtualnej z jednego systemu Xen na drugi bez konieczności pełnego zatrzymywania maszyny wirtualnej. Podczas tego procesu kopiowana jest też sama pamięć maszyny. XenServer pozwala też na podłączanie dysków sieciowych jako wolumenów, które można podłączyć do maszyn wirtualnych jako dyski. Dzięki temu maszyna wirtualna może być w pełni przechowywana w chmurze.

VMware ESXi

VMware ESXI to warstwa wirtualizacji w produktach VMware (na przykład vSphere Server) [4][24]. Jest hipernadzorcą typu 1, tak jak XenServer.

ESXi dostarcza własne jądro systemu operacyjnego, tak zwany "vmkernel", który zarządza pamięcią maszyn wirtualnych oraz dostępem do sprzętu [10]. Nie jest on oparty o jądro Linux'a.

Rozwiązanie to dostarcza interfejs REST do zarządzania maszynami wirtualnymi oraz wygodne CLI.

2.8. Konteneryzacja

Konteneryzacja jest zupełnie innym podejściem do wirtualizacji względem poprzednio omówionych technik. W technikach konteneryzacyjnych maszyna wirtualna jest zwana po prostu kontenerem.

Kontener jest tak naprawdę odpowiednio ograniczonym procesem uruchomionym w odizolowanym środowisku, jednak cały czas na systemie operacyjnym hosta. Kontenery cechuje szybkość, skalowalność, łatwa możliwość modyfikacji, orchestracji oraz organizacji [30].

Jedyn z głównych zagrożeń konteneryzacji jest to, że użytkownik w kontenerze jest tym samym użytkownikiem, który występuje na głównym systemie operacyjnym. Na przykład jeżeli kontener jest uruchomiony jako użytkownik o identyfikatorze 0 i grupie o identyfikatorze 0 - jest uruchomiony jako root. Jeżeli sprawdzimy jaki użytkownik naprawdę wykonuje ten proces w liście procesów - też będzie to root. Sprawia to pewien problem - jeżeli znajdzie się usterka, bądź potocznie zwana dziura w systemie izolacji kontenera - ten będzie mógł z łatwością dostać się do głównego komputera [29].

Orchestracja jest to pojęcie dotyczące zarządzania cyklem życia systemu. W przypadku wykorzystania do tego systemów konteneryzacji można szybko tworzyć nowe kontenery, łączyć je między sobą pomiędzy urządzeniami, przenosić kontenery z urządzenia na urządzenie, oraz stosować równoważenie obciążenia. Serwis sam reaguje na zwiększenie obciążenia poprzez powołanie do życia nowych kontenerów. Sam również usuwa niepotrzebne kontenery, jeżeli obciążenie jest mniejsze. Pozwala to na utrzymywanie serwerów tylko w sytuacji, w których są one aktualnie wykorzystywane. Przykładem systemów orchestracji są docker swarm lub Kubernetes.

Programiści mogą w bardzo szybki i prosty sposób testować oprogramowanie. Wszystkie zależności systemowe są w systemie plików kontenera [31].

Wdrażanie oprogramowania również jest tutaj zaletą - wystarczy odpowiedni system konteneryzacji, obraz i można stworzyć kontener. Dzięki czemu można dostarczać oprogramowanie w paczkach i uruchamiać je na środowisku produkcyjnym.

Kontener powinien być tworzony tak, aby mógł być w bardzo prosty sposób zastąpiony innym kontenerem, dzięki czemu taki kontener może w przypadku awarii, lub aktualizacji być szybko odtworzony. Żadne dane, które są w obrębie kontenera nie powinny być przechowywane w samym kontenerze, a być przechowywane w tak zwanym wolumenie [29].

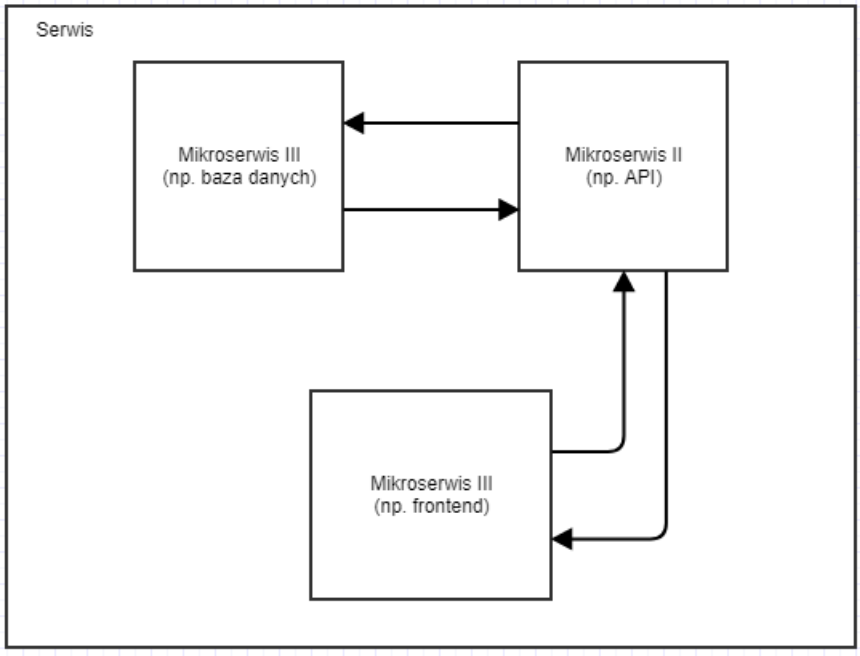
Jedną z kolejnych zalet konteneryzacji jest to, że skoro jest to tylko proces - można kontenery uruchamiać w środowiskach wirtualizowanych. W przypadku wirtualizacji, aby móc udostępnić zagnieżdżoną wirtualizację należałoby przyznać prawa do interfejsów wspomagania wirtualizacji maszynie wirtualnej, a to wiąże się z bardzo dużym ryzykiem naruszenia bezpieczeństwa. Maszyna

wirtualna bowiem ma wtedy możliwość uruchomienia uprzywilejowanej maszyny wirtualnej, która mogłaby - na przykład - przydzielić potomnej maszynie wszystkie zasoby serwera.

Kontenery teoretycznie są pozbawione dodatkowego nakładu w postaci hipernadzorcy, ale posiadają ograniczenia narzucone przez system realizowane poprzez wbudowane mechanizmy jądra systemu. To jednak pozwala na uruchamianie kontenerów w środowiskach wirtualizowanych, co pozwala na komplementarne używanie tych technik.

Architektura mikroserwisów

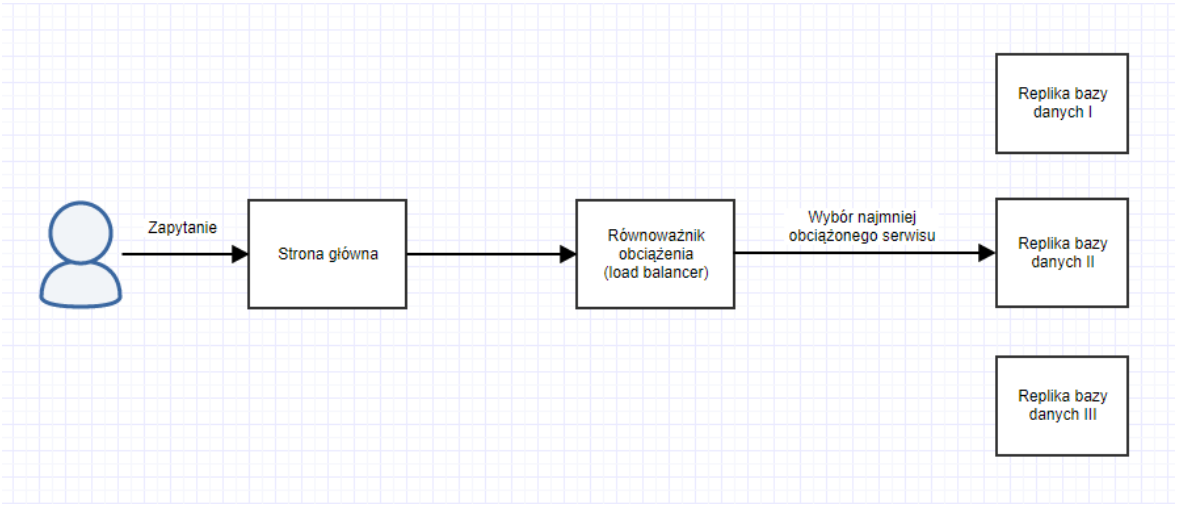
Architektura mikroserwisów to modularny system małych serwisów. Odbiega się od tworzenia wielkich monolitycznych konstruktów i tworzy się małe serwisy, które udostępniają określoną funkcjonalność w określonych (najczęściej prostych) kanałach komunikacji. Tą funkcjonalność może konsumować inny mikroserwis. Można sobie wyobrażać mikroserwisy jak puzzle, które łączymy w całość poprzez określonych konsumentów i producentów. Schemat tej architektury można zobaczyć na rysunku 2.7.



Rysunek 2.7. Architektura mikroserwisów. Pomiędzy komponentami w serwisie znajdują się połączenia.

Architektura mikroserwisów pozwala na lepszą pracę zespołową (zespoły mogą pracować nad mikroserwisami osobno), bardziej zorganizowane i modularne komponenty, łatwość modyfikacji poszczególnych elementów, oraz skalowalność samych serwisów.

Wystarczy wyobrazić sobie sytuację w której mamy 100 zapytań na godzinę - każdy serwer raczej utrzyma taką ilość. Co jednak w sytuacji, jeżeli mamy takich zapytań kilka milionów na godzinę? Bez skalowalności i równoważenia obciążenia takie serwisy by nie istniały. Schemat takiego rozwiązania można zobaczyć na rysunku 2.8.



Rysunek 2.8. Działanie równoważnika obciążenia dla przykładowego serwisu.

Jeżeli jednak wyabstrahujemy tutaj odpowiednio komponenty i dobrze dopasujemy połączenia między nimi, a następnie dobrze dobierzemy system orchestracji - jest to jak najbardziej możliwe.

Systemy konteneryzacji są jedną z głównych technik wykorzystywanych przy mikroserwisach. Kontener pełni wtedy rolę wyżej opisanego mikroserwisu, puzzla, którego można łączyć z innymi. Taki kontener może eksponować określone porty i mieć odpowiednio ustawione zależności z innymi kontenerami.

Open Container Initiative

Open Container Initiative (OCI) jest to jeden z projektów Linux Foundation, którego celem jest ujednolicenie formatu uruchamiania kontenerów i schematu obrazów w kontenerze. Systemów konteneryzacji jest coraz więcej i schematów działania również. To stworzyło potrzebę stworzenia ujednoliconego formatu. Całą specyfikację OCI można zobaczyć na głównej stronie projektu.

Dzięki ujednoliconemu schematowi można z powodzeniem budować systemy niezależne od systemu konteneryzacji, który w tym przypadku może być wymienny. Przykładem takiego systemu jest system orchestracji Kubernetes. Kubernetes zaimplementował interfejs CRI (interfejs urucho-

mieniowy kontenerów) dzięki któremu można z powodzeniem wymienić system konteneryzacji na taki, który realizuje schemat OCI.

Obraz i repozytoria obrazów

Obraz jest w systemach konteneryzacji podstawowym zainicjalizowanym systemem plików, z którego zostaje stworzony kontener [30]. Obrazy te są umieszczane w repozytoriach obrazów, skąd mogą być sciągane na inne komputery, czy tez upubliczniane przez dostawców, aby trafiły do szerszego grona.

Uzupełnieniem tego konceptu jest ujednolicona kolekcja warstw, która pozwala na tworzenie obrazu na podstawie innego obrazu. Każda operacja na bazowym systemie plików obrazu tworzy wtedy nową warstwę.

Najpopularniejszym repozytorium obrazów dla kontenerów spełniający standard OCI jest docker-hub. Istnieją otwartoźródłowe implementacje repozytorium obrazów takie jak Sonatype Nexus 3 lub Harbour, dzięki czemu można utworzyć prywatną usługę repozytorium.

Sieć

Obsługa sieci jest realizowana najczęściej poprzez mostkowaną wirtualną kartę sieciową (bridge). Do kontenera podczas uruchomienia uzyskuje własny interfejs podpięty do mostka, a ruch sieciowy przechodzi właśnie przez niego.

Istnieją różne sterowniki sieciowe kontenerów, ale są zależne od poszczególnej implementacji.

Wolumen

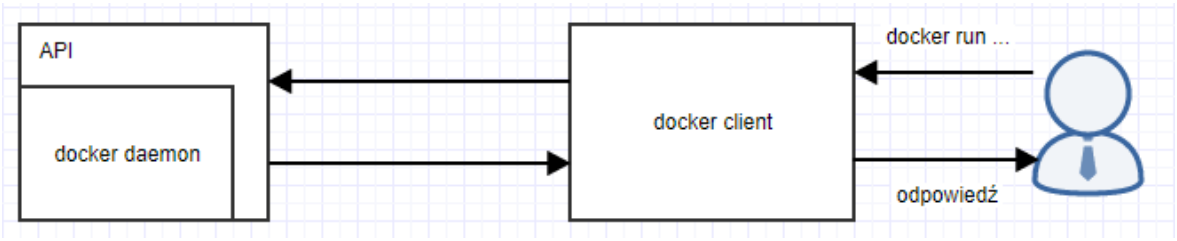
Wolumeny są mechanizmem do trwałego zapisu plików obsługiwanym przez dany system konteneryzacji. W systemie plików kontenera nie ma rozróżnienia pomiędzy katalogiem zwolumenowanym, a zwykłym - zachowuje się jak zwykły katalog. Pozwala to na różne implementacje i wyabstrahowanie wolumenu. Kontener próbuje zapisać do katalogu dane, a te mogą być zapisane zupełnie gdzie indziej i w inny sposób, niż tylko na danym dysku. Najlepszym przykładem są wolumeny w sieci - kontener ma wtedy podpięty katalog, który tak naprawdę jest katalogiem zdalnym na innym serwerze.

2.9. Systemy konteneryzacji

docker

Docker jest najpopularniejszym systemem konteneryzacji zapoczątkowanym przez Docker Inc. jako projekt otwartoźródłowy w marcu 2013. Aktualnie ma wsparcie dla systemu Windows, oraz pełne wsparcie dla Linux'a.

Docker składa się z dwóch głównych komponentów - daemon'a z API, który może wyeksponować swój interfejs jako gniazdo TCP lub zwykło gniazdo Unix i klienta, który może się z tym gniazdem komunikować. Schemat został pokazany na rysunku 2.9.



Rysunek 2.9. Schemat pokazuje komponenty dockera i komunikację między nimi.

Ten system konteneryzacji udostępnia też szereg sterowników przechowywania danych. Jednym z nich jest sterownik overlay2 który zmniejsza wielkość systemu plików kontenera na dysku.

Sterownik overlay2, który jest też najpopularniejszym sterownikiem przechowuje dane w taki sposób, że rozróżnia bazowe pliki obrazu względem plików już działającego kontenera. W chwili w której kontener zapisuje coś w swoim systemie plików - zapisuje tak naprawdę tylko w katalogu, gdzie znajdują się zmiany kontenera względem danego obrazu. Stosuje dzięki temu podejście "kopiuj przy zapisie", więc wszystkie pliki w kontenerze dopóki nie są w jakiś sposób zmodyfikowane - są plikami bazowymi obrazu. W przypadku zwykłego sterownika taki kontener zostałby utworzony poprzez skopiowanie całego obrazu. W przypadku kiedy obraz zajmuje fizycznie 1 gigabajt, serwis w zasadzie nic nie zapisuje, a kontenerów mamy 10 - oszczędzamy ponad 10 gigabajtów.

Docker ma też wbudowane dodatkowe sterowniki sieciowe takie jak macvlan, czy overlay.

Sterownik macvlan pozwala na to, aby kontener dockera dostał unikalny adres MAC i zmostkowany z oryginalnej karty sieciowy interfejs. Dzięki temu kontener może mieć swój adres IP, który może być osiągalny z innych kontenerów. I tak kontener A będzie miał przykładowo IP 192.168.0.1, a kontener B 192.168.0.2.

Docker bazuje na implementacji containerD, która jest z kolei API na wysoko poziomowym

API nad systemem konteneryzacji runC. ContainerD jest odpowiedzialny za cykl życia kontenera - zarządzanie obrazami, tworzenie, usuwanie, zarządzenie kontenerami runc.

System ten ma wbudowane narzędzie orchestracji - Docker Swarm. Pozwala to na szybkie orchestrowanie urządzeń z dostępnym narzędziem do wdrażania skalowalnych aplikacji.

rkt

System konteneryzacji rkt został zapoczątkowany w 2014 jako projekt otwartoźródłowy. Podstawową różnicą między Dockerem, a rkt jest brak procesu zarządzającego kontenerami. Docker ma swój daemon, a rkt zaleca użycie wbudowanego w system menedżera serwisami (na przykład systemd). Powoduje to teoretyczne zmniejszenie narzutu centralnego procesu, który zarządzałby kontenerami.

Jedną z zalet tego systemu jest inne podejście do tworzenia kontenera - nie ma tutaj daemon'a uruchomionego jako główny użytkownik. Dzięki temu kontenery mogą być uruchamiane przez użytkowników nieuprzywilejowanych, a użytkownik uruchamiający w tym przypadku będzie użytkownikiem, który ten kontener uruchomił.

Obrazy rkt są kompatybilne z obrazami docker'a - obrazy z oficjalnych repozytoriów dockera mogą być wykorzystywane przy tworzeniu i uruchamianiu kontenerów rkt.

Głównym wyróżnikiem wśród systemów konteneryzacji jest podejście do bezpieczeństwa. Przede wszystkim obrazy muszą być podpisane odpowiednim, znanym dla systemu kluczem, lub użytkownik świadomie będzie musiał podać w argumencie wywołania opcję insecure.

runc

Narzędzie runc służy do uruchamiania kontenerów, które spełniają standard OCI. Jest podstawą innych platform do konteneryzacji, np. Docker'a.

Kontenery tworzy się na podstawie specyfikacji opisanej szczegółowo w dokumentacji. Możemy ustawić przestrzeń nazw procesu (kontener powinien widzieć procesy włączone tylko w nim, ale czasami trzeba to zmienić), ustawienia systemu plików, zasoby izolowane poprzez wewnętrzny mechanizm jądra systemu Linux - cgroup, odpowiednie zezwolenia jądra (capabilities), zmienne środowiskowe kontenera, punkty montowania [7].

Ten system konteneryzacji wspiera również punkty kontrolne - zapisanie aktualnego stanu kontenera i możliwość jego przywrócenia w dowolnym momencie. Jest to technologia podobna do migawek znanych z innych technik wirtualizacji.

Narzędzie nie posiada daemon'a - wszelkie operacje związane z ciągłym monitorowaniem stanu kontenera, czy restartowaniu go należy zapewnić samemu.

Runc wymaga do działania bazowego systemu plików w którym uruchomi się proces. Najczęściej realizuje się to przez eksport systemu plików z obrazu, który spełnia standard OCI (np. export obrazu dockerowego jako bazowego systemu plików).

LXC

LXC (Linux Containers) jest narzędziem, które umożliwia uruchamianie kontenerów. Zwykle uruchamia się na tym systemie kontenery, które mają pełny system bazowy, pełny obraz systemu. W przypadku np. Docker'a - wystarczy wycinek tego systemu, lub w skrajnych przypadkach (np. obraz traefik) - jest to tylko jeden plik ze wszystkimi zależnościami [32]. Wczesne wersje Docker'a bazowały na LXC jako jego platforma do kontenerów.

LXC również rozdziela zasoby poprzez wbudowany mechanizm cgroup, a maszyna wirtualna uruchomiona jest odpowiednio z właściwym profilem AppArmor i w odpowiedniej przestrzeni nazw - dzięki temu jest rozdzielona od innych kontenerów.

LXC ma wbudowane API, które można obsłużyć w popularnych językach oprogramowania [1]. Ten system konteneryzacji wspiera migawki, które działają na zasadzie zapisania aktualnego stanu kontenera i możliwości powrotu do tego momentu w dowolnej chwili.

Linux Containers pozwala na bardziej szczegółowe zarządzanie kontenerem. Obsługa sieci, wolumenów i mechanizmów znanych z innych systemów konteneryzacji nie jest w tym przypadku obudowana w wysokopoziomowe API. Pozwala to na szczegółowe zarządzanie kontenerami i tworzenie dokładnie takiego środowiska, jakie zaplanowaliśmy.

ROZDZIAŁ 3

Algorytmy przetwarzania obrazu

Rozdział ten jest poświęcony algorytmom użytym do badań wydajności systemów wirtualizacji w tej pracy - algorytmu filtrowania obrazu i algorytmu tworzenia termogramu.

3.1. Algorytm filtrowania obrazu

Działanie algorytmu filtrowania obrazu polega na obliczeniu wartości koloru pixela na podstawie pixeli w pobliżu badanego i pewnej wagi zapisanej w masce filtra. Dzięki temu każdy punkt obok badanego pixela ma swoje przełożenie na pixel wynikowy [21].

W przypadku punktów skrajnych najczęstszą metodą jest nieobliczanie ich, lub usunięcie w obrazie wynikowym brzegów.

Maska to macierz z dowolnymi liczbami całkowitymi, najczęściej ma nieparzysty wymiar (3x3, 5x5) ze względu na to, że środkowa wartość tej macierzy odpowiada badanemu pixelowi.

Ogólny wzór na obliczenie wartości punktu jest następujący:

$$g(x,y) = w * f(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x-s, y-t) \tag{3.1}$$

gdzie $g(x,y)$ to piksel o współrzędnych x i y na obrazie wynikowym, $f(x,y)$ to piksel na obrazie wejściowym, w to maska. Każdy element maski jest rozważany jako $-a \leqslant s \leqslant a$ i $-b \leqslant t \leqslant b$.

Wśród filtrów możemy wyróżnić następujące kategorie:

- Filtry dolnoprzepustowe - są to głównie filtry uśredniające - usuwające różnice między sąsiadami. Przepuszczają elementy obrazu o małej częstotliwości. W tej grupie nalezy również zaliczyć też filtry Gaussa - oparte o krzywą rozkładu normalnego.
- Filtry górnoprzepustowe - w odróżnieniu od filtrów dolnoprzepustowych - obraz zwiększa ilość szumów i ulega wyostrzeniu
- Filtry przesuwania i odejmowania - dzięki nim można wykryć krawędzie w obrazie
- Gradientowe filtry kierunkowe - służą do wykrycia krawędzi w obrazie w określonym kierunku

- Filtry uwypuklające - uwypuklają obraz na krawędziach
- Filtry Laplace'a - również służą do wykrywania krawędzi ale wielokierunkowo

Rysunek 3.1 przedstawia przykładową maskę dolnoprzepustową.

1	1	1
1	1	1
1	1	1

Rysunek 3.1. Przykładowa maska filtra uśredniającego. W tym przypadku każdy otaczający pixel ma wagę 1, przez co w takim samym stopniu wpływa na pixel wynikowy.

Rysunek 3.2 przedstawia zmodyfikowaną maskę gradientową stworzoną przeze mnie, która wyodrębnia kontur obrazu.

-1	0	0	0	0	0	1
0	-2	0	0	0	2	0
0	0	-3	0	3	0	0
0	0	0	0	0	0	0
0	0	3	0	-3	0	0
0	2	0	0	0	-2	0
1	0	0	0	0	0	-1

Rysunek 3.2. Przykładowa maska gradientowa (opracowanie autorskie)

Rysunek 3.3 przedstawia maskę filtru wykorzystującego rozkład Gaussa.

1	1	2	2	2	1	1
1	2	2	4	2	2	1
2	2	4	8	4	2	2
2	4	8	16	8	4	2
2	2	4	8	4	2	2
1	2	2	4	2	2	1
1	1	2	2	2	1	1

Rysunek 3.3. Maska wykorzystująca krzywą rozkładu normalnego. Została wykorzystana w badaniach jako jedna z masek wykorzystywanych do filtracji obrazu.

Na rysunku 3.4 został przedstawiony obraz przetworzony za pomocą maski opracowanej przeze mnie z rysunku 3.2 - można na niej zaobserwować wyodrębnione kwadraty kolorów.

Na rysunku 3.5 został przedstawiony obraz przetworzony za pomocą maski wykorzystującej rozkład Gaussa z tabeli 3.3 - można zaobserwować, że cały obraz został rozmyty.

Rysunek 3.4. Przykład obrazu przetworzonego za pomocą maski gradientowej

Rysunek 3.5. Przykład obrazu przetworzonego za pomocą maski uśredniającej

3.2. Algorytm tworzenia termogramu

Promieniowanie cieplne to promieniowanie elektromagnetyczne (podczerwień) generowane przez ruch termiczny cząstek obiektu. Wszystkie obiekty, które mają temperaturę większą niż zero bezwzględne emitują to promieniowanie [26][13][14]. Długość fal tego promieniowania nie mieści się w świetle widzialnym, zakres to od 700nm do 1mm przy częstotliwości 430THz do 300GHz [28].

Promieniowanie to można zmierzyć za pomocą kamery termowizyjnej. Urządzenia te mają szerokie zastosowanie np. w noktowizji i medycynie (wykrywanie ognisk zapalnych).

Termogram to zapis promieniowania cieplnego danego obiektu. Może być zapisany jako dwuwymiarowa tablica wartości cieplnych.

Pierwszym krokiem w procesie tworzenia termogramu jest dobranie odpowiedniej palety kolorów i wybraniu odpowiedniej skali do dobierania odpowiednio koloru do danej przetwarzanej aktualnie wartości [23]. Przykładową paletą kolorów może być paleta kolorów tęczy, gdzie kolory są odpowiednio przekształcane. Wzór dla operacji skalowania i późniejszego wyznaczenia koloru, gdzie realValue jest wartością dla której ma być wyliczony kolor:

$$
\begin{aligned}
yMin = 0; yMax = 255; xMin = max(values); xMax = min(values) \\
scaled = yMin + \frac{yMax - yMin}{xMax - xMin} \times (realValue - xMin) \\
R = (1 + \cos(\frac{4 \times \pi}{3 \times 255} \times scaled)/2) \times yMax \\
G = (1 + \cos(\frac{4 \times \pi}{3 \times 255} \times scaled - \frac{2 \times \pi}{3})/2) \times yMax \\
B = (1 + \cos(\frac{4 \times \pi}{3 \times 255} \times scaled - \frac{4 \times \pi}{3})/2) \times yMax
\end{aligned}
$$

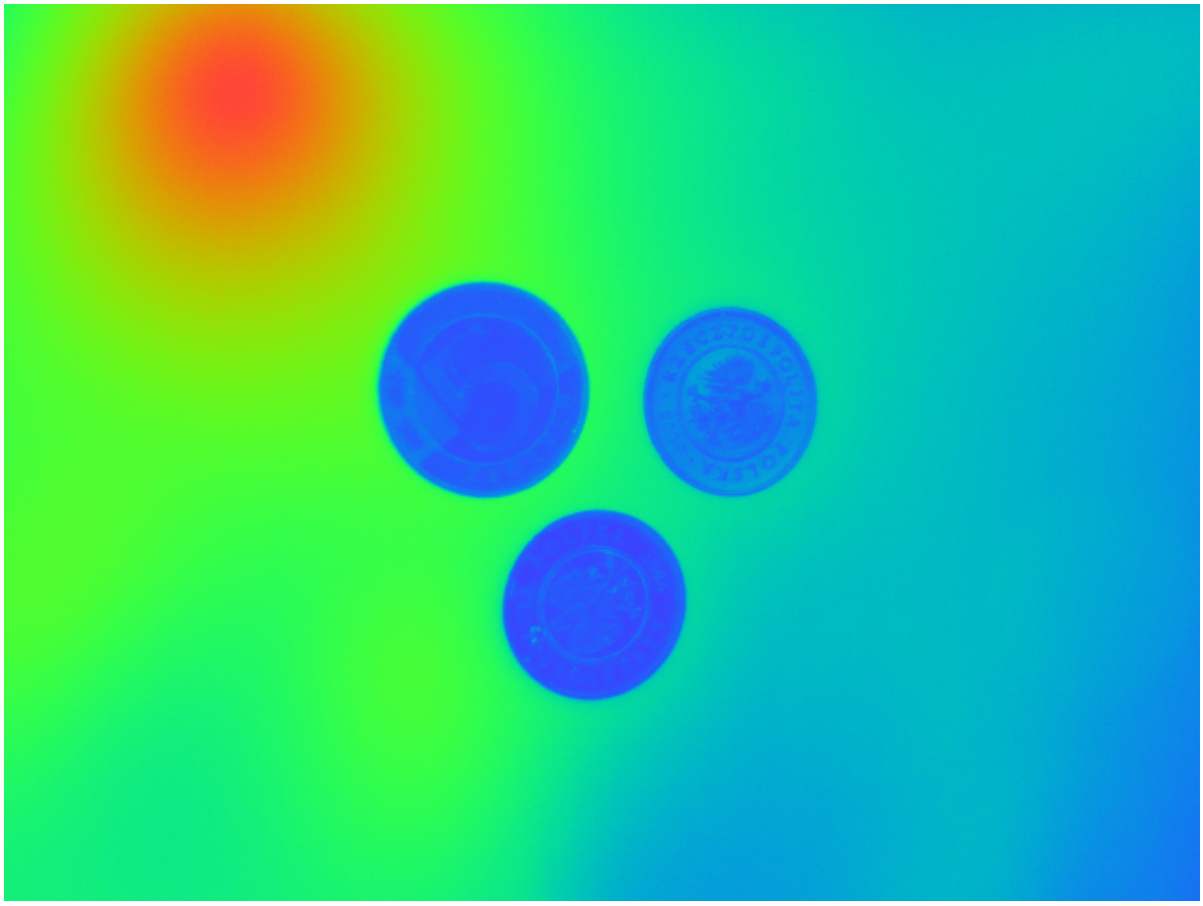
Dzięki temu otrzymamy kolor zgodny z paletą kolorów RGB.

Na rysunku 3.6 przedstawiono przykładowe wartości temperatury zapisane w macierzy 10x10. Realne rozmiary odpowiadają rozdzielczości obrazu, który został zarejestrowany przez kamerę termowizyjną.

Przykład przetworzonego termogramu wykorzystującego wyżej wymieniony wzór znajduje się na rysunku 3.7.

37.14	37.13	37.06	37.30	37.42	37.46	37.57	37.61	37.66	37.65
37.14	37.04	37.28	37.30	37.43	37.32	37.58	37.69	37.68	37.75
37.28	37.29	37.28	37.31	37.41	37.71	37.64	37.79	37.74	37.81
37.15	37.31	37.28	37.43	37.48	37.64	37.49	37.72	37.83	37.75
37.17	37.31	37.35	37.46	37.47	37.57	37.73	37.63	37.60	37.87
37.42	37.32	37.44	37.51	37.49	37.45	37.70	37.72	37.77	37.96
37.24	37.50	37.43	37.54	37.55	37.67	37.83	37.74	37.74	37.72
37.27	37.28	37.43	37.50	37.54	37.63	37.58	37.89	37.85	37.90
37.24	37.37	37.52	37.61	37.45	37.72	37.71	37.87	37.86	38.02
37.23	37.60	37.41	37.57	37.72	37.59	37.72	37.74	38.01	37.99

Rysunek 3.6. Fragment termogramu o rozmiarze 10x10 (zarejestrowane wartości temperatury). Rzeczywisty termogram z kamery termowizyjnej ma rozmiar 640 x 480



Rysunek 3.7. Przykładowa wizualizacja termogramu za pomocą palety kolorów rainbow (wynik działania algorytmu)

ROZDZIAŁ 4

Metodologia badań

4.1. Plan badania

Badania zostały przeprowadzone dla trzech próbek:

- Pierwsza próbka to przetwarzanie obrazu o dużej rozdzielczości za pomocą algorytmu filtrowania obrazów i maski mojego autorstwa.
- Druga próbka to przetwarzanie obrazu o dużej rozdzielczości za pomocą algorytmu filtrowania obrazów i maski uśredniającej Gaussa.
- Trzecia próbka to tworzenie termogramu przy pomocy zapisu cieplnego o rozdzielczości 640x480 i powtórzenie tego tworzenia 750 razy

Wskaźnikami dla badania są punkty czasowe w nanosekundach opisujące odpowiednio:

- Czas uruchomienia programu
- Czas uruchomienia przetwarzania
- Czas zakończenia przetwarzania
- Czas zakończenia programu

Dzięki tym wskaźnikom można obliczyć odpowiednio:

- Czas wykonania programu (czas zakończenia programu - czas uruchomienia programu)
- Czas przetwarzania obrazu (czas zakończenia przetwarzania - czas rozpoczęcia przetwarzania)
- Czas załadowania obrazu (czas rozpoczęcia przetwarzania - czas uruchomienia programu)
- Czas zapisu obrazu (czas zakończenia programu - czas zakończenia przetwarzania)

Próbki zostaną przetworzone w następujących środowiskach wirtualizacyjnych:

- Docker

- lxc
- rkt
- runc
- XenServer
- QEMU wirtualizacja wspomagana sprzętowo architektury i686
- QEMU wirtualizacja niewspomagana sprzętowo architektury i686
- QEMU wirtualizacja wspomagana sprzętowo architektury x86_64
- VirtualBox
- VMware Workstation Pro

Dla odniesienia zostało również przeprowadzone badanie dla systemu operacyjnego niewirtualizowanego

Badania zostały powtórzone stukrotnie dla każdej próbki. Wszystkie wyniki można zobaczyć na nośniku dołączonym do pracy.

Badanie przeprowadzono we wszystkich przypadkach na tym samym sprzęcie komputerowym.

W tabeli 4.1 można zobaczyć szczegóły sprzętu na którym przeprowadzono badanie. Szczegółowe dane na temat sprzętu zostały umieszczone na nośniku dołączonym do pracy.

Procesor	Intel Core i7 4770K CPU @ 3.50GHz
Ilość aktywnych CPU	8
Tryb pracy CPU	32-bit, 64-bit
Ilość wątków na rdzeń	2
Ilość rdzeni na gniazdo	4
Ilość gniazd	1
Procesor pakiet	Socket 1150 LGA
Interfejs wirtualizacji	VT-x
Pamięć RAM	16 GB
Bazowy system operacyjny	Ubuntu 18.04

Tablica 4.1. Specyfikacja parametrów urządzenia na którym przeprowadzono badanie

4.2. Implementacja algorytmów

Algorytmy zostały zaimplementowane w języku Go. Dostarczają interfejs CLI, dzięki któremu odpowiednio dla algorytmu pierwszego można wczytywać dowolną maskę, a dla algorytmu drugiego dowolny plik z zapisem termicznym.

Język Go

Język Go to statycznie typowany, kompilowalny język stworzony przez firmę Google [18]. Go ma wbudowany „garbage collector" - proces, który niejawnie usuwa nigdzie więcej niereferowane struktury.

Język ten ma prostą obsługę wielowątkowości. Na listingu 4.1 pokazano przykładowy kod uruchamiający wątek wraz z obsługą kanałów (ang. channel).

Kanał jest to wbudowana struktura pewnego typu z ograniczoną, lub nieograniczoną ilością danych. Jest bezpieczny w kontekście wielowątkowości. Na kanałach można odpowiednio wykonywać operacje wysłania, odebrania danych i zamknięcia kanału. Wysłanie i odebranie realizuje się poprzez operator „<-". Wątek, który próbuje odebrać coś z kanału w którym aktualnie nic się nie znajduje - przełącza się w stan oczekiwania i automatycznie odblokowuje kiedy coś się w nim pojawi. Wątek, który próbuje wysłać coś do kanału pełnego - również przechodzi w stan czuwania i odblokowuje się kiedy może wysłać coś do kanału [15].

Listing 4.1. Wielowątkowość w Go

```
package main
import (
        "fmt"
        "strconv"
        "math/rand"
)
func say(s string, channel chan string) {
        channel <- s + strconv.Itoa(rand.Intn(10)) // wyslanie do kanalu
}
```

```
func main() {
        channel := make(chan string) // utworzony kanal typu string
        go say("1nowy", channel)     // kanal przekazany do funkcji
        go say("2nowy", channel)
        fmt.Println(<-channel, <-channel, <-channel)
        // wypisuje wartosci odebrane z kanalu.
        //Dopoki nie moze odebrac z kanalu - watek glowny czeka
}
```

Język ten oferuje też szeroko rozbudowaną bibliotekę standardową zorganizowaną w pakietach. Między innymi są to implementacje algorytmów funkcji mieszających, algorytmów pakowania (tar, zip, gzip), wbudowana obsługa obrazów (bmp, png, jpeg), wbudowana obsługa SQL, funkcji matematycznych, protokołu http.

W języku Go są napisane najpopularniejsze systemy konteneryzacji i orchestracji - rkt, docker, runc, kubernetes. Jest używany w popularnych usługach jak Google Cloud, Dropbox, AWS czy Azure [22].

Język ten cechuje również brak pewnych wspólnych cech dzisiejszych języków programowania - nie ma konstruktorów, adnotacji, generycznych typów i wyjątków. Wspiera za to wskaźniki i interfejsy, ale w inny sposób - w Go można dopisywać metody do każdego typu, w związku z tym interfejs jest spełniony w momencie, kiedy dany typ implementuje wszystkie jego metody i nie musi jednoznacznie wskazywać, że typ ten będzie implementował ten interfejs [33]. Na listingu 4.2 został ukazany przykładowy kod interfejsu.

Listing 4.2. Interfejsy w Go

```
package main
import ("fmt")
type Thesis interface {
        isDone() bool
}
type MasterThesis struct {
        done bool
}
type OtherThesis struct {}
// Metoda dodana do struktury dzieki czemu spelnia interfejs Thesis
func (mThesis MasterThesis) isDone() bool { return mThesis.done }
func (mThesis OtherThesis) isDone() bool { return false }
```

```
func main() {
        var myThesis Thesis
        myThesis = MasterThesis{true};
        var otherThesis Thesis
        otherThesis = OtherThesis{};
        fmt.Println(myThesis.isDone()) // true
        fmt.Println(otherThesis.isDone()) // false
}
```

Algorytm filtrowania obrazu

Na listingu 4.3 można zobaczyć implementację funkcji, która filtruje aktualny pixel dla każdej maski. Został użyty typ danych float64 ze względu na możliwość obsługi maski, która nie składa się z liczb całkowitych.

Listing 4.3. Implementacja algorytmu filtrowania

```
func (pixelIterator *PixelFilterIteratorStruct) filterCurrentPixel() bool {
...
        maxMaskY := len(*pixelIterator.mask)
        maxMaskX := len((*pixelIterator.mask)[0])
        for y := 0; y < maxMaskY; y++ {
                for x := 0; x < maxMaskX; x++ {
                        maskVal := (*pixelIterator.mask)[y][x]
                        if maskVal == 0 {
                                continue
                        }
                        pixelVal := pixelIterator.getRelativePixel(x-center, y-center)
                        if pixelVal == nil {
                                continue
                        }
                        r, g, b, _ := pixelVal.RGBA()
                        summed[0] = summed[0] + float64(r)*maskVal
                        summed[1] = summed[1] + float64(g)*maskVal
                        summed[2] = summed[2] + float64(b)*maskVal
                }
        }
...
}
```

W obrębie całego programu implementującego algorytm 1 istnieje też intefejs, który spełnia implementacja. Na listingu 4.4 można zobaczyć nagłówek tego interfejsu. Metoda filterWithMask filtruje cały obraz, a metody nextX i nextY powinny odpowiednio ustawiać aktualny pixel na kolejny względem osi X lub Y.

Listing 4.4. Interfejs PixelFilterIterator

```
type PixelFilterIterator interface {
        calculateMask() int64
        nextX() bool
        nextY() bool
        filterWithMask() bool
        getRelativePixel(int, int) color.Color
}
```

Na rysunku 4.1 ukazano sposób włączenia programu i prosty interfejs CLI.

```
/ # ./algho
Usage: imageToProcess imageResult maskFile
```

Rysunek 4.1. Prosty interfejs pozwalający na wybór maski, pliku do przetwarzania i pliku do zapisania po przetwarzaniu

Program został skompilowany i dołączony do wirtualizowanych środowisk.

Algorytm tworzenia termogramu

Na listingu 4.5 pokazano implementację głównej części algorytmu - obliczania wartości koloru względem zeskalowanej wartości i obliczonej z palety kolorów wartości.

Listing 4.5. Implementacja algorytmu tworzenia termogramu

```
func (processor *termoImageProcessor) processImage() bool {
  for y, xArray := range *processor.values {
    for x, value := range xArray {
      l := processor.scalatorStruct.scalator(value)
      pi4 := math.Pi * 4
      pi2 := math.Pi * 2
      threeMult := 3.0 * 255
      rCalculated := ((1 + math.Cos((pi4/threeMult)*l))/2)*255
      gCalculated := ((1 + math.Cos((pi4/threeMult)*l-(pi2/3)))/2)*255
```

```
        bCalculated := ((1 + math.Cos((pi4/threeMult)*1-(pi4/3)))/2)*255
        r := uint8(rCalculated)
        g := uint8(gCalculated)
        b := uint8(bCalculated)
        newPixelData := color.NRGBA{R: r, G: g, B: b, A: 255}
        processor.image.SetNRGBA(x, y, newPixelData)
          }
    }
    ...
}
```

Obliczone wartości należy zamienić do wartości typu uint8 ze względu na to, że z obliczonej formuły może wyjść liczba zmiennoprzecinkowa, a kolory typu RGB przyjmują liczbę całkowitą.

Implementacja tego algorytmu również posiada interfejs konsolowy do wczytywania zapisu i określenia ścieżki zapisu do pliku.

Pełne źródła są dostępne na nośniku dołączonym do pracy.

Dockerfile

Dockerfile to specyfikacja tworzenia obrazu Docker'a (aktualnie kompatybilnego z OCI). Kontenery zostały utworzone na podstawie Dockerfile ukazanego na listingu 4.6 i 4.7.

W Dockerfile umieszcza się dyrektywy, które służą do określenia jak dany obraz ma być zbudowany. Ponieważ obraz musi być powtarzalny - pliki są załadowane do bazowego systemu plików kontenera dzieki dyrektywie COPY. Dyrektywa „ENTRYPOINT" mówi jaki proces ma się uruchomić przy starcie kontenera. Warstwę bazową wybiera się przy pomocy dyrektywy „FROM".

Obraz alpine został wybrany jako warstwa bazowa, ze względu na jego niewielki rozmiar (3MB) i dostępnej minimalnej powłoce „ash".

Obrazy kontenera są dostępne w publicznym repozytorium dockerhub - razikus/virtvscontainer:1.0.sample1, razikus/virtvscontainer:1.0.sample2 i razikus/virtvscontainer:2.0.

Listing 4.6. Dockerfile algorytm 1

```
FROM alpine
COPY algho /
COPY sample1.jpg /sample1.jpg
COPY mask.txt /mask.txt
ENTRYPOINT ["/algho", "sample1.jpg", "sample1-processed.jpg", "mask.txt"]
```

Listing 4.7. Dockerfile algorytm 2

```
FROM alpine
COPY algho2 /
COPY example.txt /example.txt
ENTRYPOINT ["/algho2", "/example.txt", "/example.png", "750"]
```

ROZDZIAŁ 5

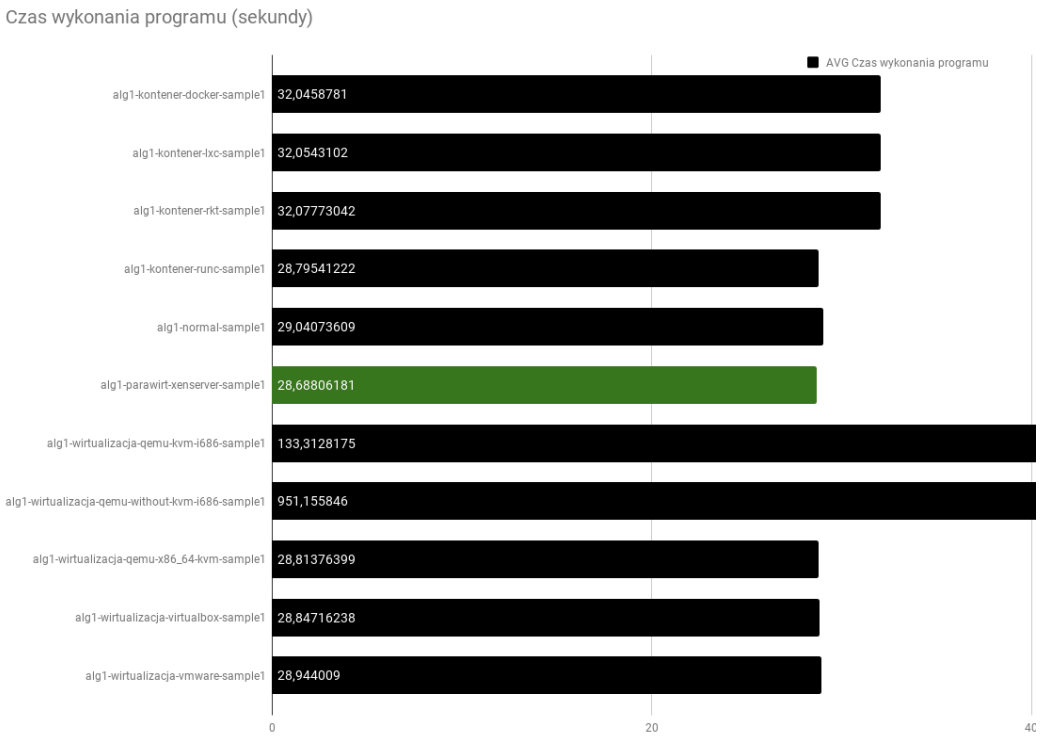
Wyniki badań i porównanie wydajności obliczeniowej

W tym rozdziale ukazano graficzne opracowanie wyników dla testów wedle czterech wskaźników - czasu wykonania programu, czasu przetwarzania obrazu, czasu załadowania obrazu i czasu zapisu obrazu. Dla każdego testu wykonano 100 próbek pomiarowych, aby wyliczyć średnie wartości.

5.1. Algorytm filtrowania obrazu maska gradientowa

Czas wykonania programu

Na rysunku 5.1 wyniki dotyczące czasu wykonania programu zostały zwizualizowane na wykresie słupkowym.

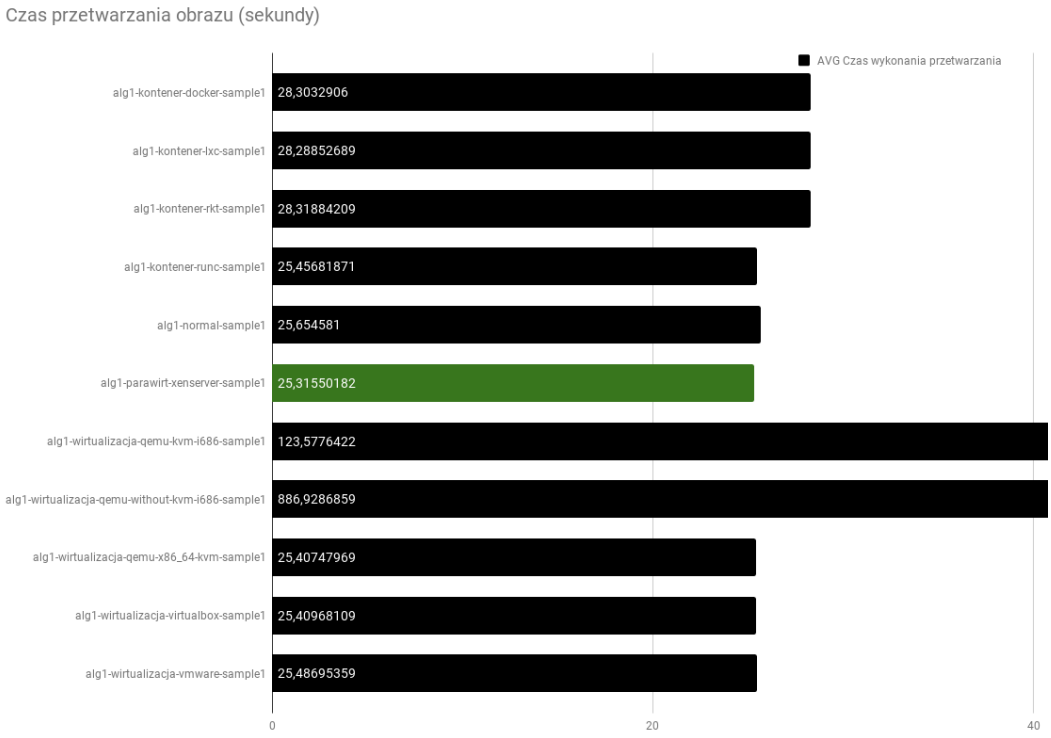


Rysunek 5.1. Czas wykonania całego programu algorytmu filtrującego, maski gradientowej

W tym przypadku najszybciej wykonanym procesem był proces wykonany przez XenServer. Najgorzej wypadła pełna wirtualizacja bez sprzętowo wspieranej wirtualizacji i bez wykorzystania interfejsu KVM.

Czas przetwarzania obrazu

Na rysunku 5.2 można zauważyć, że najszybciej wypadł XenServer, a najwolniej QEMU bez KVM z architekturą i686. Można też zaobserwować pewien narzut czasowy w przypadku dockera, lxc i rkt.



Rysunek 5.2. Czas przetwarzania obrazu algorytmu filtrującego, maski gradientowej

Czas załadowania obrazu

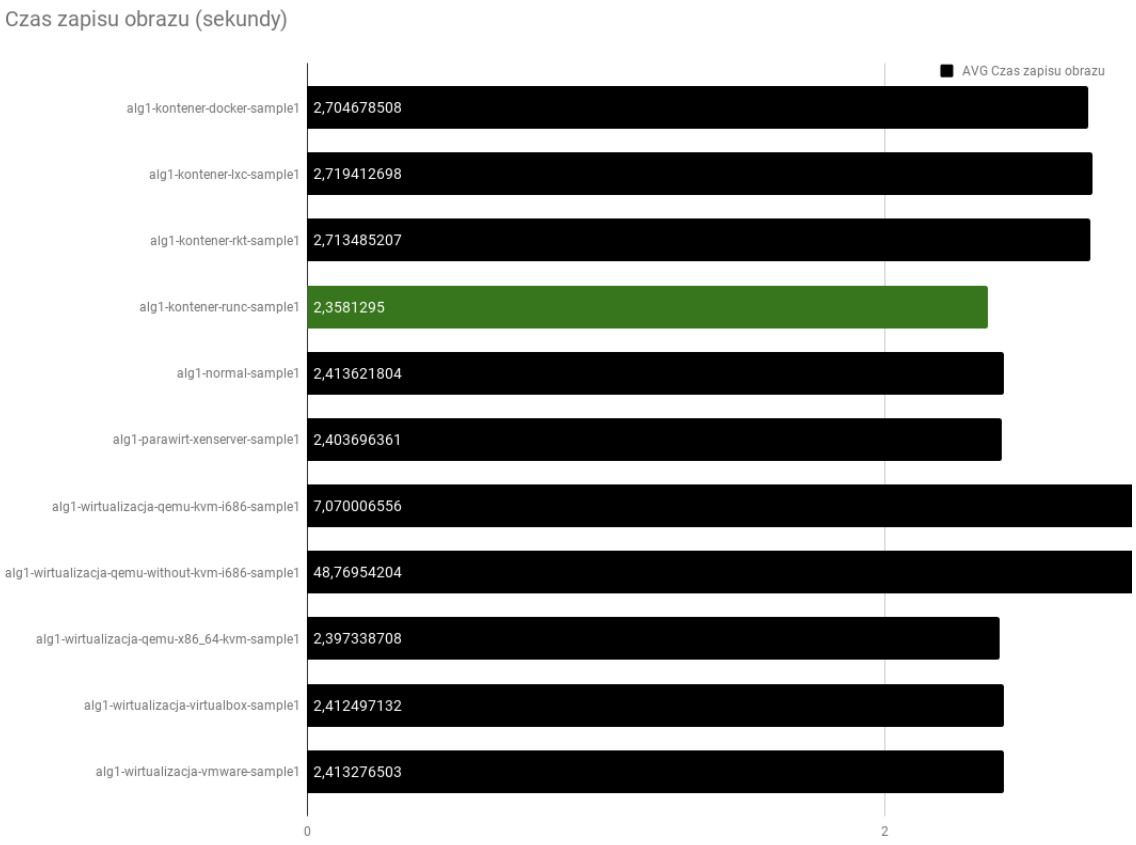
Na rysunku 5.3 ukazano zobrazowanie graficzne uśrednionego czasu załadowania obrazu.

Rysunek 5.3. Czas załadowania obrazu algorytmu filtrującego, maski gradientowej

Najszybciej załadowanie obrazu zostało zrealizowany przez XenServer. Ponownie widać pewien narzut systemów konteneryzacji. Wyniki wirtualizatorów wspomaganych sprzętowo są bardzo do siebie zbliżone.

Czas zapisu obrazu

Na rysunku 5.4 można zauważyć, że najszybciej został zapisany obraz w systemie konteneryzacji runc.



Rysunek 5.4. Czas zapisu obrazu algorytmu filtrującego, maski gradientowej

Docker i rkt również korzystają z runc jako bazowej platformy do uruchamiania kontenerów, ale wypadły gorzej. Najwolniejszy zapis ponownie wypadł dla QEMU z architekturą i686.

5.2. Algorytm filtrujący maska uśredniająca Gaussa

Czas wykonania programu

Na rysunku 5.5 ukazano wykres słupkowy ilustrujący czas wykonania programu w przypadku algorytmu filtrującego przy pomocy maski uśredniającej Gaussa.

Rysunek 5.5. Czas wykonania całego programu algorytmu filtrującego, maski uśredniającej

Najmniejszy czas był w przypadku wirtualizacji wspomaganej sprzętowo przez QEMU. W czołówce również pojawia się XenServer i inne systemy wirtualizacji. Można zauważyć, że i w tym wypadku systemy konteneryzacji mają pewien - widoczny - narzut czasowy.

Czas przetwarzania obrazu

Na rysunku 5.6 można zauważyć, że ponownie najszybciej wypadł system wirtualizacji QEMU ze sprzętowym wspomaganiem.

Rysunek 5.6. Czas przetwarzania obrazu algorytmu filtrującego, maski uśredniającej

Czas załadowania obrazu

Na rysunku 5.7 ukazano wykres słupkowy ilustrujący uśredniony czas załadowania obrazu.

Rysunek 5.7. Czas załadowania obrazu algorytmu filtrującego, maski uśredniającej

W tym przypadku czas załadowania obrazu był najszybszy dla próbki, która została uruchomiona na normalnym systemie operacyjnym.

Czas zapisu obrazu

Na rysunku 5.8 znajduje się wykres słupkowy uśrednionego czasu zapisu obrazu dla próbki drugiej algorytmu filtrowania.

Rysunek 5.8. Czas zapisu obrazu algorytmu filtrującego, maski uśredniającej

W tym przypadku najszybszy wynik był dla systemu konteneryzacji runc.

5.3. Algorytm tworzenia termogramu

Czas wykonania programu

Na rysunku 5.9 znajduje się wykres słupkowy ilustrujący uśredniony czas wykonania programu dla algorytmu tworzenia termogramu.

Rysunek 5.9. Czas wykonania całego programu dla algorytmu tworzenia termogramu

System konteneryzacji runc osiągnął najlepszy wynik czasowy dla tego przypadku. Pozostałe systemy wirtualizacji również są tutaj widoczne wśród najlepszych. Systemy konteneryzacji docker, lxc i rkt znowu ukazują pewien widoczny dodatkowy narzut czasowy.

Czas przetwarzania obrazu

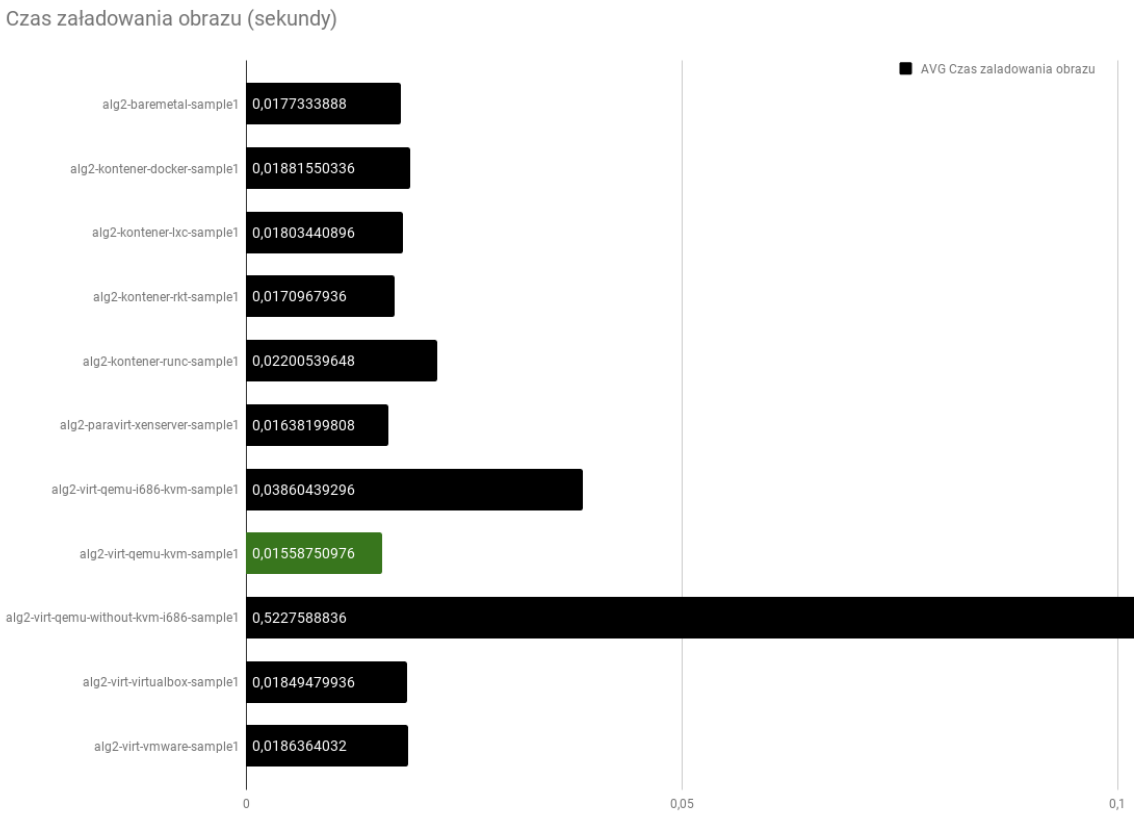
Rysunek 5.10 wizualizuje uśrednione wyniki czasu przetwarzania obrazu w przypadku algorytmu tworzenia termogramu.

Rysunek 5.10. Czas przetwarzania obrazu dla algorytmu tworzenia termogramu

W tym przypadku system konteneryzacji runc osiągnął najszybszy wynik w przetwarzaniu obrazu. Dwudziestotrzykrotnie wolniej przetworzył obraz system wirtualizacji QEMU bez wspomagania sprzętowego.

Czas załadowania obrazu

Na rysunku 5.11 pokazano wizualizację uśrednionego czasu załadowania obrazu.



Rysunek 5.11. Czas załadowania obrazu dla algorytmu tworzenia termogramu

Najszybszy wynik uzyskał system wirtualizacji QEMU ze sprzętowym wspomaganiem wirtualizacji. System parawirtualizacji XenServer był na drugim miejscu.

Czas zapisu obrazu

Rysunek 5.12 pokazuje uśredniony czas zapisu obrazu.

Rysunek 5.12. Czas zapisu obrazu dla algorytmu tworzenia termogramu

W tym teście bezkonkurencyjnie wypadł system konteneryzacji runc, następnie w czołówce plasują się miejsca dla systemów wirtualizacji.

ROZDZIAŁ 6

Podsumowanie i wnioski

Założony cel pracy został osiągnięty - zostały zbadane wyniki czasowe przetwarzania obrazów w wirtualizowanych środowiskach. Przeanalizowane i opisane zostały też różne techniki wirtualizacji i kilka systemów, które te techniki realizują.

XenServer jako środowisko parawirtualizacyjne umożliwia maszynom wirtualnym pełne wykorzystanie zasobów. We wszystkich próbkach Xen pojawia się w czołówce rezultatów. Prawdopodobnie jest to spowodowane tym, że system ten jest zupełnie dostosowany właśnie pod maszyny wirtualne - w odróżnieniu do VirtualBoxa czy VMware, które to są uruchamiane jako oprogramowanie systemie operacyjnym, a nie same nim są.

Popularne systemy konteneryzacji w odróżnieniu do wirtualizowanych środowisk ze sprzętowym wspomaganiem mają pewien widoczny narzut czasowy, który w pewien sposób ogranicza ich wykorzystanie w momencie w którym chcielibyśmy użyć stu procent zasobów sprzętu. Prawdopodobnie jest to spowodowane wewnętrznymi mechanizmami ograniczania tych zasobów, lub ograniczeniem systemu konteneryzacji tak, aby umożliwił dalszą pracę systemowi hosta.

Wśród systemów konteneryzacji najszybszym był runc, który jest dedykowaną platformą używaną chociażby przez Docker'a czy rkt. Platforma ta pozbawiona jest jednak pewnego widocznego dla pozostałych systemów konteneryzacji narzutu czasowego i najlepiej sprawdziła się w tych testach.

Rezultaty badań okazały się niespójne z marketingową wiedzą dostępna w materiałach w internecie. Materiały te podkreślają, że kontenery - skoro są tylko procesem na oryginalnym natywnym systemie - są najszybszym sposobem wirtualizacji, gdyż teoretycznie nie wymagają dodatkowych mechanizmów jak translacja binarna, a ograniczają tylko odpowiednio ten proces. Systemy konteneryzacji wprowadzają jednak swoje mechanizmy ograniczania zasobów i monitorowania kontenerów, przez co uniemożliwiają osiągnięcie natywnej wydajności.

Należy pamiętać, że badania przeprowadzone mogą być obarczone pewnymi błędami, które starano się zminimalizować - obecność innych procesów w systemie czy też inne czynniki zewnętrzne.

Badania te ujmują tylko wąski zakres tego co oferuje konteneryzacja i wirtualizacja, w kolejnych pracach należałoby sprawdzić jak systemy wirtualizacji są zoptymalizowane w warunkach ograniczonych zasobów, co jest normalną sytuacją dla wszystkich technik wirtualizacji.

DODATEK A

Zawartość nośnika

Struktura dołączonego nośnika:

```
CD
├─ repozytorium #Skrypty i nieprzetworzone rezultaty badań.
│  ├─ algorytmy #katalog z zaimplementowanymi algorytmami
│  ├─ alg1 #katalog z wynikami badań dla algorytmu 1
│  │  ├─ baremetal #system operacyjny hosta
│  │  ├─ kontener #systemy konteneryzacji
│  │  │  ├─ docker
│  │  │  ├─ rkt
│  │  │  ├─ lxc
│  │  │  └─ runc
│  │  ├─ paravirt #systemy parawirtualizacji
│  │  │  └─ XenServer
│  │  └─ virt #systemy wirtualizacji
│  │     ├─ VMware
│  │     ├─ VirtualBox
│  │     ├─ QEMU-kvm
│  │     ├─ QEMU-i686-kvm
│  │     └─ QEMU-without-kvm-i686
│  ├─ alg2 #katalog z wynikami badań dla algorytmu 2
│  │  └─ ...
│  └─ sample #katalog z próbkami użytymi do badań
└─ PracaMagisterska.pdf #niniejsza praca.
```

Bibliografia

[1] Dokumentacja lxc. https://linuxcontainers.org/lxc/documentation/. Dostęp: 20.05.2019.

[2] Domeny ochrony. http://kik.pcz.pl/soold/mainpage/subject17/chapt7.htm. Dostęp: 20.05.2019.

[3] Everything you need to know about the intel virtualization technology. https://www.hardwaresecrets.com/everything-you-need-to-know-about-the-intel-virtualization-technology/2/. Dostęp: 20.05.2019.

[4] Introduction to vmware vsphere. https://www.vmware.com/pdf/vsphere4/r40/vsp_40_intro_vs.pdf. Dostęp: 20.05.2019.

[5] Qemu dokumentacja. https://www.qemu.org/. Dostęp: 20.05.2019.

[6] Qemu supported devices. https://en.wikibooks.org/wiki/QEMU/Devices. Dostęp: 20.05.2019.

[7] Specyfikacja runc. https://github.com/opencontainers/runtime-spec/blob/master/config-linux.md.

[8] Virtualbox manual. https://www.virtualbox.org/manual/ch01.html#snapshots. Dostęp: 20.05.2019.

[9] Virtualization overview. https://www.vmware.com/pdf/virtualization.pdf. Dostęp: 20.05.2019.

[10] Vmkernel scheduler. https://communities.vmware.com/docs/DOC-5501. Dostęp: 20.05.2019.

[11] Wirtualizacja z akceleracją na poziomie procesora. https://blog.integratedsolutions.pl/wirtualizacja/wirtualizacja-akceleracja-na-poziomie-procesora. Dostęp: 20.05.2019.

[12] S. Wilson A. Muller. *Virtualization with VMware ESX Server*. Syngress, 2005.

[13] S. Dech C. Kuenzer. *Thermal Infrared Remote Sensing: Sensors, Methods, Applications*. Springer Netherlands, 2013.

[14] Douglas A. Christensen. *Introduction to Biomedical Engineering*. Morgan and Claypool Publishers, 2009.

[15] K. Cox-Buday. *Concurrency in Go*. O'Reilly Media, 2017.

[16] D. McCrory D. Marshall, W. A. Reynolds. *Advanced Server Virtualization*. Auerbach Publications, 2006.

[17] P. Dash. *Getting Started with Oracle VM VirtualBox*. Packt Publishing, 2013.

[18] C. Doxsey. *Introducing Go*. O'Reilly Media, 2016.

[19] A. Vettathu H. Devassy Chirammal, P. Mukhedkar. *Mastering KVM Virtualization*. Packt Publishing, 2016.

[20] VMware Inc. *Understanding Full Virtualization, Paravirtualization, and Hardware Assist*. VMware, 2008.

[21] J. Ludwig. Wykład satellite digital image analysis, 581. http://web.pdx.edu/~jduh/courses/Archive/geog481w07/Students/Ludwig_ImageConvolution.pdf.

[22] M. Helmich M. Andrawos. *Cloud native programming with Golang : develop microservice-based high performance web apps for the cloud with Go*. Packt Publishing, 2017.

[23] Klaus-Peter Mõllmann M. Vollmer. *Infrared Thermal Imaging: Fundamentals, Research and Applications*. Wiley-VCH, 2018.

[24] D. Mishchenko. *VMware ESXi: Planning, Implementation, and Security*. Course Technology PTR, 2010.

[25] D. Ruest N. Ruest. *Virtualization, A Beginners Guide*. McGraw-Hill, 2009.

[26] B. Kuchowicz J. Prochorow A. Kujawski E. Skrzypczak A. Śliwiński J. Świderski Z. Trzaska Durski K. Wierzchowski P. Decowski, M. Grynberg. *Encyklopedia fizyki współczesnej*. Państwowe Wydawnictwo Naukowe, 1983.

[27] T. Ritzau R. Warnke. *QEMU*. BoD, 2009.

[28] K.N. Rao. *Wavelength Standards in the Infrared*. Elsevier Academic Press, 1966.

[29] A. Raźniewski. Docker. kurs video. poziom drugi. sieci, wolumeny i ustawienia daemona, 2018. ISBN: 978-83-283-5398-5.

[30] A. Raźniewski. Docker. kurs video. praca z systemem konteneryzacji i docker swarm, 2018. ISBN: 978-83-283-5093-9.

[31] A. Raźniewski. Linux server. kurs video. usługi serwerowe, skrypty i środowisko graficzne, 2019. ISBN: 978-83-283-5760-0.

[32] S. Kumaran Senthil. *Practical LXC and LXD. Linux Containers for Virtualization and Orchestration*. Apress, 2017.

[33] M. Tsoukalos. *Mastering Go: Create Golang production applications using network libraries, concurrency, and advanced Go data structures*. Packt Publishing, 2018.

[34] W. von Hagen. *Professional Xen Virtualization*. Wrox/Wiley Pub, 2008.

[35] D. E. Williams. *Virtualization with Xen: including XenEnterprise, XenServer, and XenExpress*. Syngress, 2007.

Spis tablic

Spis rysunków

Spis listingów

Streszczenie

Praca porusza temat różnych technik wirtualizacji - wirtualizacji pełnej i wspomaganej sprzętowo, parawirtualizacji i konteneryzacji.

Głównym celem pracy jest przeanalizowanie tych systemów pod kątem czasu przetwarzania obrazów, wydajności algorytmu filtrowania i algorytmu tworzenia termogramu w popularnych systemach wirtualizacji oraz zbadania działania tych systemów.

Zakres prac między innymi obejmuje przygotowanie środowisk wirtualizacji, implementacje algorytmów filtrowania i tworzenia termogramu w języku Go, scharakteryzowanie systemów wirtualizacji i analiza wyników pomiarowych.

Rozdział „Wprowadzenie" przybliża techniki wirtualizacyjne i systemy wirtualizacji, które te techniki wykorzystują. Jest w nim również zawarte wprowadzenie do zagadnień takich jak pierścienie ochrony i emulacja.

Rozdział „Algorytmy przetwarzania obrazu" opisuje algorytmy, które zostały przedstawione w pracy - algorytm filtrowania i algorytm tworzenia termogramu. Ukazuje też przykładowe przetworzone obrazy.

Rozdział „Metodologia badań" opisuje plan badania, pokazuje implementację algorytmów wykorzystanych do testów, jest w nim zawarty opis języka Go, oraz ukazuje specyfikację Dockerfile do stworzenia obrazów w standardzie OCI.

Rozdział „Wyniki badań i porównanie wydajności obliczeniowej" ukazuje opracowane uśrednione wyniki wszystkich testów i próbek oraz pokazuje ich opracowanie graficzne.

W rozdziale „Podsumowanie i wnioski" znajdują się wnioski i podsumowanie na temat przeprowadzonych badań.

Słowa kluczowe

Wirtualizacja, parawirtualizacja, konteneryzacja, algorytm filtrowania obrazu, algorytm tworzenia termogramu

Abstract

This thesis is about testing different virtualization techniques - full virtualization, paravirtualization and containerization.

The main aim is to analize these systems in case of image processing time, performance of filter and thermogram creation algorithms in popular virtualization systems and study these systems.

The scope is also preparation of virtualization execution environment, algorithms implementation in Go language, characteristics of different virtualization systems and analysis of test results.

Chapter „Introduction” is about virtualization techniques and systems. There is also introduction to topics like protection rings and emulation.

Chapter „Image processing algorithms” is about alghoritms that were used in thesis - filter and termogram creation. There are also example processed images.

Chapter „Study methodology” is describing test plan and shows implementation of alghoritms that were used, there is also description of Go language and specification of Dockerfiles that were used to build OCI images.

Chapter „Results and comparision of computational efficiency” shows average tested samples and shows graphical representation of results.

In chapter „Conclusions and summary” there are conclusions and summary about tests.

Imię i nazwisko: Adam Raźniewski
Nr albumu: 129201
Kierunek: Informatyka
Wydział: Inżynierii Mechanicznej i Informatyki
Politechnika Częstochowa

Częstochowa, dn. 08.06.2019r

OŚWIADCZENIE

autora pracy dyplomowej

Pod rygorem odpowiedzialności karnej oświadczam, że złożona przeze mnie praca dyplomowa pt. „Analiza wydajności technik wirtualizacji, para-wirtualizacji i konteneryzacji na podstawie implementacji wybranych algorytmów przetwarzania obrazów”

jest moim samodzielnym opracowaniem i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Jednocześnie oświadczam, że praca w całości lub we fragmentach nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w uczelni.

Wyrażam/ ~~nie wyrażam~~[**] zgodę na nieodpłatne wykorzystanie przez Politechnikę Częstochowską całości lub fragmentów ww. pracy w publikacjach Politechniki Częstochowskiej.

Ponadto oświadczam, że treść pracy przedstawionej przeze mnie do obrony zawarta na przekazywanym nośniku elektronicznym jest identyczna z wersją drukowaną.

……
podpis studenta

[*] w przypadku zbiorowej pracy dyplomowej, dołącza się oświadczenia każdego ze współautorów pracy dyplomowej

[**] niepotrzebne skreślić