WYŻSZA SZKOŁA
HUMANISTYCZNO – EKONOMICZNA
W ŁODZI

WYDZIAŁ Informatyki, Zarządzania i Transportu

KIERUNEK Informatyka

Przemysław Adam Kupisz
53105

Implementacja autorskiego środowiska monitorującego maszyny wirtualne w systemie operacyjnym z/VM

Przyjmuję pracę jako inżynierską

podpis promotora.....

data .....

Praca napisana pod kierunkiem
dr Ścibór Sobieski

ŁÓDŹ 2009

Spis treści

Wstęp

Niniejsza praca ma na celu zaimplementowanie autorskiego środowiska monitorującego maszyny wirtualne w systemie operacyjnym z/VM kooperującego z rozwiązaniami typu Open Source.

Celem pierwszego rozdziału pracy będzie zapoznanie czytelnika z architekturą systemów zSeries. W jego skład wchodzi wprowadzenie do powyższej architektury, charakterystyka procesorów, pamięci głównej, partycji logicznych oraz urządzeń I/O.

W kolejnym rozdziale zostanie przedstawiona konfiguracja wirtualnej maszyny, stosu TCP/IP oraz usługi *MONITOR wraz z DCSS (Discontinuous Save Segment).

Trzeci rozdział będzie mówił o instalacji i konfiguracji systemu operacyjnego z/Linux od momentu pobrania instalatora z internetu, przez instalacje systemu, po końcową konfiguracje systemu, łącznie z instalacją dodatkowych pakietów.

Cały czwarty rozdział poświęcony będzie wpierw programowi zpmd (z/VM Performance Monitor Daemon). Wprowadzi czytelnika w jego charakterystykę i funkcjonalność. Przedstawi interfejs programistyczny dla usługi *MONITOR systemu z/VM V5R3.0 oraz zaprezentuje wybrany kod źródłowy programu z krótkim opisem. Kolejnym opisywanym programem będzie zpm (z/VM Performance Monitor). Tu również pojawi się opis i charakterystyka aplikacji. Ponadto ukażą się rysunki ukazujące działanie graficznego interfejsu użytkownika i pobierane dane z bazy MySQL.

Kolejnym działem z kolei będzie podsumowanie opisujące otrzymane wyniki oraz dalszy kierunek rozwoju projektu. Dalej w kolejności pojawi się streszczenie w dwóch wersjach: polskiej i angielskiej. Następne rozdziały zawierające słowa kluczowe, bibliografię, spis tabel i rysunków.

W prezentowanej pracy we wszystkich ramkach kolor czarny jest oznaczeniem, że tekst w nich zawarty jest wierszem poleceń wpisywanym do konsoli. Natomiast ramki koloru niebieskiego, że dany tekst został wyświetlony na standardowym wyjściu.

1 Central Processing Complex

1.1 Wprowadzenie do architektury zSeries

Komputery klasy mainframe są to jednostki charakteryzujące się wysoką wydajnością operacji I/O w przetwarzaniu dużych wolumenów danych, niezawodnością oraz skalowalnością.

Mainframe'y różnią się od przeciętnych komputerów osobistych popularnie zwanych PC (Personal Computer). Tak jak komputery klasy PC tworzone są z myślą o konkretnym profilu zastosowania, tak i komputery mainframe zostały stworzone do ściśle określonych celów, jakimi są m.in. masowe przetwarzanie dużej ilości wolumenów danych oraz szybka obsługa dużej ilości zapytań (użytkowników) w krótkim czasie.

Sercem komputera klasy mainframe jest CPC (Central Processing Complex) w którym znajdują się procesory, pamięć operacyjna i kanały (ang. channels) służące do połączeń pozostałych komponentów komputera, jak np. macierze dyskowe.

Zadbano, by operacje były wydajne i bezpieczne, dlatego też w każdym newralgicznym miejscu zdublowane zostały części maszyny. Od dwurdzeniowego procesora, w którym każdy z rdzeni przetwarza te same dane, a następnie na ich podstawie tworzone są sumy kontrolne i porównywane pomiędzy rdzeniami, aż do zdublowania urządzeń zwanych support element, które to są zwykłymi laptopami podłączonymi bezpośrednio do CPC służącymi jako fizyczne terminale.

1.2 Charakterystyka procesorów systemu z9

Architekturą procesorów centralnych w systemach mainframe jest CISC (Complex Instruction Set Computers). Ich moc mierzona jest w MIPS (Million Instructions Per Second), co znaczy, że mierzy się liczbę milionów operacji stałoprzecinkowych wykonywanych w czasie jednej sekundy przez określoną jednostkę obliczeniową.

Jednakże przeliczanie mocy systemów mainframe tylko na MIPS'y jest błędem.

W tak złożonym środowisku istotnie problematyczne jest porównywanie maszyn do siebie. Dlatego też firma IBM udostępnia szereg narzędzi i dokumentów prezentujących osiągi maszyn o podobnych konfiguracjach.

Obecnie seria „*z*" ma kilka różnych procesorów, więc używa się pojęcia PU (Processing Units) w którego skład wchodzą:

CP – Central Processor, jeden ze strategicznych procesorów, nazywany procesorem ogólnego przeznaczenia. Wymagany przez systemy operacyjne takie jak: z/OS i z/VM. Posiada CPACF (CP Assist Cryptographic Function) oferujący szyfrowanie metodami: AES dla 128 bitowych kluczy, DES, TDES, PRNG, SHA-1, SHA-256.

IFL - Integrated Facility for Linux, przeznaczony do obsługi systemu z/Linux, wykorzystywany również przez z/VM oraz OpenSolaris działający pod jego kontrolą. IFL jest to zredukowany o kilka instrukcji CP, tak by nie mógł obsługiwać systemu operacyjnego z/OS.

zAAP - System z9 Application Assist Processor jest to dodatkowy procesor, służący do zwiększenia wydajności aplikacji, działających pod kontrolą systemu z/OS. Pełni rolę wsparcia dla CP. Nie potrafi wykonać IPL (Initial Program Load). Przejmuje na siebie wykonywanie aplikacji napisanych w języku programowania Java wykonywanych w Wirtualnej Maszynie Javy firmy IBM. Także jest odpowiedzialny za analizę składni dokumentów XML.

zIIP – System z9 Integrated Information Processor. Jest to dodatkowy procesor, służący do zwiększenia wydajności w przetwarzaniu zasobów bazodanowych pod kontrolą systemu z/OS. Pełni rolę wsparcia dla CP. Nie potrafi wykonać IPL. Przejmuje na siebie wykonywanie zapytań do baz danych, również posiada wsparcie dla funkcji ładowania, reorganizacji i przebudowywania indeksów w DB2. Działa tylko z aplikacjami wykorzystującymi tryb SRB (Service Request Block) oraz architekturę DRDA (Distributed Relational Database Architecture).

ICF – Internal Coupling Facility jest procesorem obsługującym klaster komputerów mainframe działających w technologii Parallel Sysplex.

SAP – System Assistance Processor zapewnia wewnętrzne wsparcie obsługi I/O. Zarządza CHPIDs[1] (Channel Path Identifiers). Odpowiada za rozwiązywanie drobnych tymczasowych błędów I/O. Jest niewidoczny dla systemów operacyjnych

1 Patrz podrozdział 1.5

znajdujących się na każdym z LPAR'ów[2] (Logical Partitions).
Spare – Zapasowy procesor włączany w zastępstwie wadliwego.

1.3 Pamięć główna

W środowisku mainframe pamięć operacyjną nazywa się pamięcią główną (ang. main storage).

Wyróżnia się dwa rodzaje pamięci:

- CS (ang. central storage)
- ES (ang. expanded storage)

W komputerach serii z9 BC (Business Class) możliwa jest obsługa do 64GB pamięci głównej. Dodatkowo, każdy LPAR może posiadać adresowanie 24, 31 lub 64 bitowe w przydzielonej mu pamięci głównej.

Pamięć fizyczna umiejscowiona jest w bloczku (ang. book), który jest w stanie pomieścić aż do ośmiu kart pamięci (ang. memory cards) o rozmiarze 2, 4 lub 8 GB. Uzyskano to umiejscawiając je na dwóch przeciwnych brzegach po cztery karty pamięci na każdym z nich. Możliwe jest umiejscowienie bloczków różniących się od siebie wielkością posiadanej pamięci w obrębie tego samego CPC.

1.4 Partycje logiczne

LPAR (Logical Partition) jest to logiczna partycja stworzona na poziomie sprzętowym. Dzięki wirtualnym partycjonowaniu możliwe jest uzyskanie do 60 odizolowanych systemów na platformie mainframe. Na każdej logicznej partycji może zostać uruchomiony tylko i wyłącznie jeden system operacyjny. Dodatkowo dla każdej partycji należy zdefiniować zasoby systemowe (np. karty sieciowe, dyski, ilość i przydział mocy procesorów), które mogą zostać podzielone między logicznymi partycjami oraz dedykowane na wyłączność jednego LPAR'a. Daje to możliwości pozwalające w pełni wykorzystać zasoby serwera, gdyż sprzyja to np. rozłożeniu mocy

2 Patrz podrozdział 1.4

procesora na kilka systemów operacyjnych, a więc zwiększa się szansa całkowitego wykorzystania mocy obliczeniowej procesora przez całą dobę.

LPAR'y definiuje się za pomocą IOCP (I/O configuration program) lub (nowszego) HCD (Hardware Configuration Definition). Bezpieczeństwo, a więc izolacja poszczególnych partycji rozwiązana jest na poziomie sprzętowym. Co jest wykorzystywane do umieszczania systemów produkcyjnych i developerskich całkowicie odizolowanych od siebie, a jednocześnie dających możliwość łączenia systemów produkcyjnych tak by pracowały jako jeden, zwiększając przy tym bezpieczeństwo i niezawodność.

1.5 Urządzenia I/O

Central Processing Complex może posiadać aż 1024 osobne kanały dla urządzeń I/O za których dostęp z poziomu logicznych partycji w systemach mainframe odpowiada CSS (Channel Subsystem), jest on niezależny od CP, więc cały ruch I/O w systemie mainframe odbywa się asynchronicznie bez udziału procesorów centralnych. Dzięki temu, gdy przychodzi operacja I/O do procesora CP, Channel Subsystem ją przechwytuje, a CP nie marnuje czasu oczekując na dane z urządzeń, lecz zajmuje się przetwarzaniem oczekujących zadań.

Każdy LPAR w CPC komunikuje się z urządzeniami I/O poprzez LCSS (Logical Channel Subsystem), który jest identyczny pod względem funkcjonalności do CSS, jednak w pojedynczym CPC zdefiniować można maksymalnie cztery LCSS, służące w celu zwiększania liczby CHPID (Channel Path ID) w systemie, ponieważ CHPID zdefiniowane jest w LCSS to jest unikalne w jego obrębie. Następnie komunikacja dociera do podkanałów (ang. subchannels) reprezentujących urządzenia i stamtąd do kanałów (ang. channels) by za ich pośrednictwem wydostać się z CPC.

Urządzenia wejścia/wyjścia oraz kanały I/O umiejscowione są w klatce (ang. cage) I/O. Kanały te różnią się od siebie w zależności od typu urządzeń do których mają być podłączone.

Obecnie najpopularniejszym, a zarazem najnowszym medium transmisji jest kanał FICON Express (Fiber Connection), ulepszona wersja i następca technologii ESCON (Enterprise Systems Connection), korzystająca z protokołu FCP (Fibre Channel

Protocol). Jest wydajniejszy od kanału ESCON, umożliwia łączenie ze sobą urządzeń na znacznie większym dystansie. Powyższe kanały łączą się z jednym urządzeniem lub jednym portem switcha. Switche natomiast podłączane są do bibliotek dyskowych oraz taśmowych. Możliwe jest także połączenie innych systemów.

Systemy zSeries posiadają fizyczną kartę OSA-Express (Open Systems Adapter-Express), która ma jeden port dla serwerów G5/G6 oraz dwa porty dla zSeries umożliwiające bezpośrednie podłączenie do sieci LAN (Local Area Network) lub ATM (Asynchronous Transfer Mode), dodatkowo karta może pracować w dwóch trybach: OSD (Queued Direct I/O) lub OSE (non-Queued Direct I/O). W trybie non-QDIO dane z karty OSA-Express by dostać się do pamięci głównej muszą przejść przez warstwę Control Unit, Channel oraz IOP (Input Output Processor), natomiast w trybie QDIO następuje pominięcie tych warstw, co za tym idzie dostęp do pamięci jest bezpośredni. Wiąże się to z poprawą wydajności o 20% w stosunku do trybu non-QDIO oraz zredukowania pracy procesora SAP. Przy czym obsługiwane standardy przez OSA-Express to:

Gbe (Gigabit Ethernet) - wspiera wyłącznie tryb QDIO i TCP/IP, transmisję full duplex z oddzielnymi ścieżkami dla odczytu i zapisu przy prędkości 1Gbps. Dwa typy ramek Ethernetowych: Ethernet V2.0 (DIX) i IEEE 802.3 CSMA/CD (ISO/IEC 802.3). Także ruch multicastowy, jednakże wysyła tylko ramki formatowane na Ethernet 2.0 (DIX).

1000BASE-T Ethernet - wspiera wyłącznie ruch TCP/IP w trybie QDIO. W trybie non-QDIO obsługuje ruch SNA/APPN/HPR i TCP/IP. W trybie full duplex pracują prędkości 1000Mbps, 100Mbps, 10Mbps. Tryb half-duplex jest wspierany dla prędkości 100Mbps lub 10Mbps.

FENET (Fast Ethernet) - w trybie non-QDIO wspiera ruch TCP/IP oraz SNA/APPN/HPR, dla trybu QDIO wyłącznie TCP/IP. Może pracować w trybie half-duplex lub full-duplex. Obsługuje prędkości 10Mbps lub 100Mbps.

Token Ring - wspiera wyłącznie ruch TCP/IP w trybie QDIO. W trybie non-QDIO obsługuje SNA/APPN/HPR i TCP/IP, bezpośrednie połączenia z prędkościami 4Mbps, 16Mbps lub 100Mbps. Gdy karta wykryje, że jest pierwsza w pierścieniu automatycznie ustawia prędkość 16Mbps.

ATM - w trybie QDIO wpiera wyłącznie TCP/IP przy włączonym trybie emulacji sieci

LAN. w trybie non-QDIO wspiera SAN/APN/HPR i TCP/IP. Maksymalna prędkość osiągalna to 155Mbps.

Komputery zSeries mają jeszcze jeden interfejs sieciowy, tym razem wirtualny nazywany HiperSockets. Mogący zdefiniować do 16 niezależnych wirtualnych sieci LAN. HiperSockets umożliwia bardzo wydajne połączenia TCP/IP, gdyż cały ruch odbywa się w pamięci głównej, do tego rodzaju połączeń używa iQDIO (internal Queued Direct Input/Output). Technologia ta umożliwia niezawodne, bezpieczne i wydajne uzyskiwanie połączeń między systemami operacyjnymi pracującymi na osobnych logicznych partycjach, a także pracujących jako wirtualne maszyny pod kontrolą systemu z/VM.

Ostatnimi elementami połączonymi do CPC są Support Element. Są to dwa notebooki IBM ThinkPad na których istnieje możliwość zalogowania się do systemu jako Access Administrator, System Programmer, Operator, Advanced Operator.

2 Konfiguracja systemu operacyjnego z/VM

2.1 Wirtualna maszyna

Autor postanowił zainstalować 31-bitową dystrybucję Debian (Etch) systemu operacyjnego z/Linux. Wymagane do tego jest odpowiednie przygotowanie zwirtualizowanego środowiska.

Do tego celu należy stworzyć wirtualna maszynę w systemie z/VM, dodając następujący wpis do pliku USER DIRECT na VDEV 2CC:

```
===== USER LINUX LINUX 256M 256M EG
=====  IPL CMS
=====  IUCV ANY
=====  IUCV ALLOW
=====  IUCV *MONITOR
=====  NAMESAVE MONDCSS
=====  MACHINE XA
=====  OPTION MAINTCCW RMCHINFO
=====  XSTORE 32M
=====  CONSOLE 0009 3270 A
=====  SPOOL 000C 2540 READER *
=====  SPOOL 000D 2540 PUNCH A
=====  SPOOL 000E 3203 A
=====  LINK MAINT 190 190 RR
=====  LINK MAINT 19E 193 RR
=====  DEDICATE 803 803
=====  DEDICATE 804 804
=====  DEDICATE 805 805
=====  MDISK 200 3390 0001 1249 ZLINUX MW READ WRITE MULTI
=====  MDISK 999 3390 8026 334 530RES MW READ WRITE MULTI
=====  LINK MAINT 191 191 RR
===== *
```

Kolejnym krokiem jest podlinkowanie dysku 5012 od użytkownika LABSTUD:

```
LINK 5012 LABSTUD 5012 MW MULTI
```

Następnie sformatowanie i nadanie etykiety na potrzeby instalacji systemu z/Linux wykorzystując do tego program CPFMTXA:

```
FORMAT
5012
0-END
ZLINUX
YES
END
PERM 0 END
END
```

MDISK 999 o etykiecie 530RES formatuje się analogicznie, jak MDISK 200.

2.2 Stos TCP/IP

Stos TCP/IP w systemie operacyjnym z/VM pełni rolę komunikacyjną. Pozwala uruchamiać usługi, takie jak np. FTP, HTTP dla systemu CMS. Standardowo do obsługi stosu używane są dwie wirtualne maszyny:

- TCPMAINT (pełni funkcję zarządzającą)
- TCPIP (uruchomia stos)

Aby go aktywować należy zalogować się jako użytkownik TCPMAINT. Następnie skopiować plik PROFILE STCPIP, jako PROFILE TCPIP na VDEV 591. Kolejnym krokiem jest jego edycja (dopisanie konfiguracji sieci) za pomocą edytora XEDIT:

```
=====  DEVICE MYNET OSD 0800
=====  LINK ETH1 QDIOETHERNET MYNET

===== HOME
=====   10.1.223.12 255.255.0.0 ETH1

===== GATEWAY
=====   10.1.0.0 255.255.0.0 = ETH1 1500
=====   DEFAULTNET 10.1.0.1 ETH1 1500

=====   START MYNET
```

Analogicznie należy postąpić z plikiem TCPIP SDATA na VDEV 592. Kopiując go jako TCPIP DATA, po czym dopisać:

```
===== TCPIPUSERID TCPIP
===== HOSTNAME HLAB0012
===== DOMAINORIGIN LOCAL.WSHE.LODZ.PL
```

Na końcu utworzyć plik SYSTEM DTCPARMS na VDEV 198.

Zapisując w nim:

```
===== :NICK.TCPIP  :TYPE.SERVER
=====              :CLASS.STACK
```

Gdy cała konfiguracja przebiegnie pomyślnie to z poziomu użytkownika TCPIP należy uruchomić stos wpisując polecenie:

```
START TCPIP
```

2.3 MONITOR & DCSS

System operacyjny z/VM V5R3.0 posiada usługę systemową monitorującą zasoby, zarządzaną przez polecenie MONITOR z poziomu CP (Control Program). MONITOR umożliwia zmianę ustawień monitorowania systemu w trybie rzeczywistym (bez zatrzymywania usługi). Do tego celu służą:

- MONITOR EVENT (dla rekordów typu EVENT)
- MONITOR SAMPLE (dla rekordów typu SAMPLE)

Do uruchomiania i wyłączania trybu monitorowania:

- MONITOR START
- MONITOR STOP

Wpisując polecenie w maszynie wirtualnej z uprawnieniami A/E:

```
Q MONITOR
```

Wyświetlają się ustawienia domyślne dla usługi monitorującej system:

```
MONITOR EVENT INACTIVE    BLOCK    4     PARTITION    0
```

```
MONITOR DCSS NAME - NO DCSS NAME DEFINED
CONFIGURATION SIZE      68 LIMIT        1 MINUTES
CONFIGURATION AREA IS FREE
USERS CONNECTED TO *MONITOR - NO USERS CONNECTED
MONITOR   DOMAIN ENABLED
PROCESSOR DOMAIN DISABLED
STORAGE   DOMAIN DISABLED
SCHEDULER DOMAIN DISABLED
SEEKS     DOMAIN DISABLED
USER      DOMAIN DISABLED
I/O       DOMAIN DISABLED
NETWORK   DOMAIN DISABLED
APPLDATA  DOMAIN DISABLED
MONITOR SAMPLE INACTIVE
          INTERVAL    0 MINUTES    PENDING INTERVAL    1 MINUTES
          RATE     STOP          PENDING RATE     2.00 SECONDS
MONITOR DCSS NAME - NO DCSS NAME DEFINED
CONFIGURATION SIZE      241 LIMIT        1 MINUTES
CONFIGURATION AREA IS FREE
USERS CONNECTED TO *MONITOR - NO USERS CONNECTED
MONITOR   DOMAIN ENABLED
SYSTEM    DOMAIN ENABLED
PROCESSOR DOMAIN DISABLED
STORAGE   DOMAIN DISABLED
USER      DOMAIN DISABLED
I/O       DOMAIN DISABLED
NETWORK   DOMAIN DISABLED
APPLDATA  DOMAIN DISABLED
```

Do uruchomienia w/w usługi wymagane są:

- Zdefiniowany DCSS (Discontiguous Saved Segment)
- Pomyślnie wykonane polecenie MONITOR START
- Podłączony przynajmniej jeden użytkownik do usługi *MONITOR

Jeśli więc wykonane zostanie polecenie:

```
MONITOR START
```

Nastąpi komunikat o wstrzymaniu monitorowania do czasu, aż podłączy się przynajmniej jeden użytkownik:

```
Monitor event started -- recording is pending
HCPMNC6224I Event recording is pending because there are no users connected to *
```

```
MONITOR for this type of data.
Monitor sample started -- recording is pending
HCPMNC6224I Sample recording is pending because there are no users connected to
*MONITOR for this type of data.
```

Gromadzone dane przechowywane są w DCSS mieszczącgcym się w SPOOL'u[3]. DCSS (Discontiguous Saved Segment) jest to specjalna przestrzeń definiowana z poziomu CP umożliwiająca dostęp do danych każdemu systemowi operacyjnemu uruchomionemu w wirtualnej maszynie, widziana jako część pamięci operacyjnej. Standardowo zdefiniowany i gotowy do użycia DCSS nosi nazwę MONDCSS. Istnieje możliwość zdefiniowania DCSS o innym rozmiarze i nazwie.

W większości środowisk wystarczający jest rozmiar standardowego DCSS (MONDSS). Jednak przy rozległych środowiskach mogą wystąpić problemy z wyliczeniem pojemności segmentu. Dlatego Firma IBM udostępnia oprogramowanie wspierające oszacowanie potrzebnego rozmiaru.

Poniższe polecenie sprawdza, czy w systemie istnieje standardowo zdefiniowany DCSS:

```
Q NSS NAME MONDCSS
```

Otrzymany wynik potwierdza jego istnienie:

```
OWNERID  FILE TYPE CL RECS DATE  TIME     FILENAME FILETYPE ORIGINID
*NSS     0011 NSS  R  0001 04/27 18:33:21 MONDCSS  DCSS     MAINT
```

Jednakże, brakuje rozmiaru, tak więc ponownie należy wpisać polecenie uzupełnione o kolejny argument:

```
Q NSS NAME MONDCSS MAP
```

W otrzymanym wyniku wyświetlił się adres początkowej i końcowej strony zapisany w systemie heksadecymalnym:

```
FILE FILENAME FILETYPE MINSIZE  BEGPAG ENDPAG TYPE CL #USERS PARMREGS V
MGROUP
0011 MONDCSS  CPDCSS   N/A    09000  097FF  SC  R  00001  N/A  N/A
```

Początkowy adres jest równy wartości 09000 oraz końcowy 097FF. Wartości

3 Obszar przestrzeni dyskowej, bufor przechowujący końcowe, bądź wymagające kolejnego przetworzenia dane

te są obcięte o 12 bitów (włączonych). Tak więc przed dokonaniem obliczeń następuje uzupełnienie ich od prawej strony znakami „FFF”. W tym przypadku wartości kolejno wynoszą:

- BEGPAG = 09000FFF
- ENDPAG = 097FFFFF.

Odejmując BEGPAG od ENDPAG uzyskuje się wartość 7FF000 do której należy dodać jedną stronę (ang. page) wynoszącą 4 KB (wartość 1000 w systemie szesnastkowym) ostatecznie otrzymując wartość 800000 w systemie heksadecymalnym. Po przeliczeniu na system dziesiętny wielkość DCSS jest równa 8388608 bajtów, co jest równoważne 8192 KB == 8 MB.

Dane w DCSS podzielone są na dwie sekcje, gdzie każda z nich posiada dwie osobne podsekcje:

SAMPLE (Monitor Sample Records):

- Sample Data
- Sample Config

EVENT (Monitor Event Records):

- Event Data
- Event Config

Rekordy w DCSS podzielone są na określone domeny (ang. domains):

Nr Domeny	**Nazwa Domeny**	**Ilość Rekordów**
0	System	24
1	Monitor	20
2	Scheduler	12
3	Storage	20
4	User	10
5	Processor	12[4]
6	I/O	30
7	Seek	1
8	Virtual Network	3
10	Application Data	2

Tab. 1. Domeny w z/VM V5R3.0.

4 W dokumentacji dwa rekordy „4” oraz „5” zostały oznaczone jako „Niedostępne od tej wersji systemu z/VM”.

W każdej z domen (Tabela nr 1) znajduje się ściśle określona dla tej wersji systemu struktura oraz ilość rekordów zarządzana przez usługę *MONITOR. Jest to związane ze zmianami jakie firma IBM wprowadza dla nowych wersji systemu z/VM.

Zdaniem Autora programista systemowy aktywując wszystkie domeny na rozległych instalacjach mainframe'owych powinien wziąć pod uwagę nagły spadek wydajności systemu. Szczególnie przy dopisywaniu parametru „ALL” wymuszającego monitorowanie wszystkich uczestników poszczególnej domeny oraz przy deklarowaniu interwału czasowego i częstotliwości zbieranych danych w pojedynczym interwale mających duży wpływ na obciążenie systemu.

Istnieją dwie metody na zdefiniowanie obszaru pamięci dla DCSS by był widoczny w systemie z/Linux:

- poprzez parametr „mem” dla ładowanego kernel'a
- zdefiniowanie go z poziomu CMS'a (Conversation Monitor System)

Pierwszy sposób służy do deklaracji DCSS zaczynającego się od adresu wykraczającego poza obszar 128MB pamięci operacyjnej systemu z/Linux.

```
mem=160M
```

Drugi przypadek jest używany, gdy adres początkowy i końcowy DCSS zawiera się w pamięci operacyjnej systemu z/Linux (od 256MB):

```
DEF STOR CONFIG 0.140M 152M.116M
```

W odpowiedzi uzyskany komunikat to:

```
STORAGE = 256M
Storage Configuration:
0.140M 152M.116M
Extent Specification              Address Range
--------------------------------- ---------------------------------
          0.140M          0000000000000000 - 0000000008BFFFFF
        152M.116M         0000000009800000 - 0000000010BFFFFF
Storage cleared - system reset.
```

3 Instalacja i konfiguracja systemu operacyjnego z/Linux

3.1 Pobranie i uruchomienie instalatora systemu operacyjnego z/Linux

Wykorzystując stos TCP/IP z poziomu systemu CMS należy zalogować się jako MAINT i wykonać następujące kroki:

```
LINK TCPMAINT 592 592 MR
```

oraz

```
ACC 592 J
```

W ten oto sposób użytkownik MAINT uzyskuje dostęp do komend PING, IFCONFIG, FTP m.in. potrzebnych przy instalacji. Kolejno Autor łączy się za pomocą programu FTP z serwerem dystrybucji Debian i pobiera potrzebne pliki instalatora sieciowego:

```
-rw-rw-r--    9 21285    21285          340 Jul 19 10:59 debian.exec
-rw-r--r--    1 21285    21285      2522766 Jul 19 10:59 initrd.debian
-rw-r--r--    2 21285    21285      3165256 Jul 19 10:59 kernel.debian
-rw-rw-r--   12 21285    21285           29 Jul 19 11:00 parmfile.debian
```

Na dysk CMS'owy A (VDEV 191).

Przy kopiowaniu ich na dysk CMS'owy należy pamiętać o zamianie formatu rekordów na wartość zrozumiałą dla systemu z/VM:

```
locsite fixrecfm 80
```

Posiadając już pliki potrzebne do uruchomienia instalacji prze internet należy zalogować się jako LINUX, podlinkować VDEV 191 należący do MAINT:

```
LINK MAINT 191 197 MR MULTIPLE
```

```
ACC 197 J
```

Na końcu wpisać komendę uruchamiającą instalator dystrybucji Debian:

```
EXEC DEBIAN EXEC J
```

3.2 Instalacja systemu operacyjnego z/Linux

Po poprawnym załadowaniu instalator prowadzi krok po kroku po całym przebiegu instalacji systemu. Autor konfigurował system tak jak poniższej:

- 2. qeth: OSA-Express in QDIO mode / HiperSockets
- 1. 0.0.0803-0.0.0804-0.0.0805
- 2. No
- 10.1.222.12
- 255.255.0.0
- 10.1.0.1
- 194.204.159.1 194.204.152.34
- debian
- wshe.local

W tym miejscu system generuje klucz SSH hosta. Po czym wymaga o podanie hasła dla użytkownika installer ,wykonującego dalszą cześć instalacji już z poziomu klienta SSH.

Dalsza część instalacji jest podobna, jak na innych platformach, więc Autor postanowił, że warte wspomnienia jest partycjonowanie dysków oraz utworzeni użytkownicy.

System wykrywa dyski danej wirtualnej maszyny, więc należy wybrać:

- DASD 0.0.0200 (920.8MB FREE SPACE)

- DASD 0.0.0999 (246.2MB FREE SAPCE)

Na dysku 200 został wybrany punk montowania „/” oraz system plików EXT3. Dla dysku 999 punkt montowania to „/var/lib/mysql” i również system plików EXT3. Przestrzeń dyskowa dla partycji SWAP została pominięta.

Autor umyślnie pominął tworzenie partycji SWAP, która spowalniałaby pracę systemu. W razie niewystarczającej pamięci operacyjnej zostanie podmontowany dysk tymczasowy.

Utworzono dwóch użytkowników na potrzeby Autora:

- root (użytkownik o pełnych prawach dostępu)
- pk (użytkownik o ograniczonych prawach dostępu)

Po poprawnej instalacji system oznajmi o restarcie. Sesja SSH zostanie zamknięta. Należy zalogować się poprzez terminal 3270 na wirtualną maszynę LINUX, w której uruchomiony jest CP, ponieważ z/Linux nie był na tym poziomie zdolny do ponownego uruchomienia podczas restartu.

Wykorzystując to zdarzenie należy zdefiniować w pamięci przestrzeń dla DCSS widocznego dla systemu z/Linux. W tym wypadku należy wydać polecenie:

```
IPL CMS
```

```
DEF STOR CONFIG 0.140M 152M.116M
```

Po czym uruchomić dystrybucję Debian poleceniem:

```
IPL 200
```

3.3 Konfiguracja systemu operacyjnego z/Linux wraz z instalacją dodatkowych komponentów

Po pojawieniu się znaku zachęty, należy zalogować się jako użytkownik „root” i dokonać konfiguracji systemu, poprzez załadowanie modułu odpowiadającego

za komunikację z DCSS za pomocą interfejsu IUCV (Inter User Control Vehicle). Wymagany moduł nazywa się monreader i w systemie z/Linux jest urządzeniem znakowym (ang. char device). Kroki uruchamiające moduł są następujące:

Sprawdzenie, czy moduł istnieje:

```
modprobe -l | grep monreader
```

Rezultat:

```
/lib/modules/2.6.18-6-s390/kernel/drivers/s390/char/monreader.ko
```

Załadowanie modułu:

```
modprobe monreader mondcss=MONDCSS
```

Komunikat na konsoli po załadowaniu modułu

```
IUCV lowlevel driver initialized
extmem info:segment_load: loaded segment MONDCSS range 09000000 .. 097fffff type SC in shared mode
monreader info: Loaded segment MONDCSS from 09000000 to 097fffff, size = 8388608 Byte
```

Ostatnim krokiem jest zmiana praw dostępu do urządzenia monreader (chmod 660), by użytkownik „pk” miał dostęp do urządzenia w trybie „do odczytu”:

```
chmod 664 /dev/monreader
```

Świeżo zainstalowana dystrybucja Debian wymaga doinstalowania dodatkowych pakietów, by umożliwić przyjazne środowisko do wykonywania, debugowania i kompilacji programów, wraz z obsługą bazy danych . Tak więc wykorzystując polecenie apt-get należy doinstalować poniższe paczki:

- mysql-server (dla Etch jest to wersja 5.0)
- libmysqlclient15-dev (biblioteki do poprawnego skompilowania programu zpmd)
- gcc (kompilator C)

- screen (pełnoekranowy menadżer okien)
- vim (edytor tekstowy)

4 Implementacja autorskich programów

4.1 zpmd (z/VM Performance Monitor Daemon)

Program zpmd (z/VM Performance Monitor Daemon) jest programem pracującym pod kontrolą systemu operacyjnego z/Linux oraz ściśle współpracującym z bazą danych MySQL.

Ma za zadanie przekazywać do bazy danych, w sposób uporządkowany informacje gromadzone w DCSS przez usługę *MONITOR systemu z/VM. Zpmd nie czyta bezpośrednio z samego DCSS lecz za pośrednictwem modułu jądra, który emulując urządzenie znakowe umożliwia dostęp do zasobów systemu z/VM.

4.1.1 Funkcjonalność

Program jest przeznaczony do kolekcjonowania informacji tworzonych przez usługę *MONITOR w bazie danych.

Dzięki wykorzystaniu bazy MySQL jako medium przechowywania danych stworzono spójny i uniwersalny interfejs pozwalający na dostęp wielu klientom jednocześnie.

Cechy programu zpmd v0.8:

- Kod programu napisany w języku C
- Jednowątkowy
- Jest programem przeznaczonym wyłącznie na systemy z/Linux wykonywane w środowisku z/VM
- Kod źródłowy na licencji GPL

Funkcjonalność programu zpmd v0.8 :

- Wykonuje się w trybie zdemonizowanym
- Loguje wszystkie istotne zdarzenia występujące podczas działania programu do pliku korzystając z systemowej usługi syslog
- Dokonuje konwersji formatu:
    - EBCDIC na ASCII
    - TOD na UTC
- Zapisuje dane do bazy MySQL

Program zpmd (z/VM Performance Monitor Daemon) jest programem pobierającym w czasie rzeczywistym dane w formacie binarnym z urządzenia znakowego /dev/monreader, by je kolejno przetworzyć i zapisać w bazie danych. Dane przesyłane są w formie domen[5] zawierających rekordy. Przy czym kolejność każdej z domen oraz każdego rekordu jest dowolna.

Przy uruchomieniu programu zpmd wymagane jest podanie argumentów (w następującej kolejności):

- Rozmiar segmentu (standardowy rozmiar MONDCSS to 8388608 bajtów)
- Adres IP serwera z bazą danych
- Nazwa użytkownika bazy danych
- Nazwa bazy danych

Po podaniu powyższych informacji program zapyta o hasło do bazy danych dla podanego wcześniej użytkownika.

Następnie program przejdzie w tryb daemon'a, po czym nastąpi połączenie z bazą danych MySQL i jeśli nie istnieją wymagane tabele w bazie to wyśle kwerendę „CREATE TABLE” dla wszystkich obsługiwanych rekordów, po czym utworzy bufor o rozmiarze DCSS, z którego dane są odczytywane. Jest to wymagane, ponieważ urządzenie /dev/monreader jednorazowo jest w stanie wysłać ciąg znaków o maksymalnej długości równej rozmiarowi DCSS.

Tak więc Daemon zpmd otrzymuje ciąg znaków zakończonych znakiem końca

5 Domenę należy traktować jako zbiór zawierający określoną liczbę rekordów.

pliku, po czym następuje tymczasowa przerwa w napływie danych, gdyż usługa *MONITOR generuje kolejną porcję danych do przesłania. Kod źródłowy odpowiadający za odczyt z urządzenia /dev/monreader prezentuje się następująco:

```
//Open /dev/monreader
  monfd = open("/dev/monreader", O_RDONLY, 0);
  errsv = errno;
  if (if_error(errsv) == 2) exit(1);
//Read /dev/monreader > bufor
// OBSZAR TESTOWY ===>
     __u8 load;
     int ilosc_wczytanych;
     __u32 i=0;

     load = 0;
    //load = NULL;

     while (1)
     {
       ilosc_wczytanych = read(monfd, &load, sizeof(__u8));
       errsv = errno;
       if (errsv == EAGAIN)
       {
         continue;
       }

       if (if_error(errsv) == 1)
       {
         bufor=bzero;
         memset(bufor, '0', bytes_read);
         i=0;
```

```
        while (ilosc_wczytanych)
        {
         ilosc_wczytanych = read(monfd, &load, sizeof(__u8));
        }
        load = 0;
        continue;
       }

        if (i > bytes_read)
        {
           syslog(LOG_ERR, "(i > bytes_read)");
           exit(1);
        }

           *bufor = load;
           ++bufor;
           ++i;

     if (ilosc_wczytanych != 0 ) continue;
//jesli ilosc_wczytanych = 0 program przejdzie to przetwarzania danych znajdujacych się w buforze
```

Należy wspomnieć, że zmienna bytes_read ma wartość przypisywaną z parametru programu noszącego nazwę „rozmiar segmentu” (podawane w bajtach). W powyższym kodzie użyta jest funkcja if_error(). Obsługuje ona kody błędów generowane przez urządzenie /dev/monreader. Przedstawia się następująco:

```
// /dev/monreader access exepctions
int if_error(int errsv)
{
    //switch (errsv)
    if (errsv == EFAULT)
    {
    syslog(LOG_ERR, "EFAULT Read /dev/monreader failed. Copy to user failed. Missing data. \n");
```

```
    return 1;
    }

    if (errsv == EIO)
    {
      syslog(LOG_ERR, "EIO Reply failed. Missing data. No IUCV connection to the z/VM *MONITOR could be established. \n");
    return 1;
    }
    if (errsv == EOVERFLOW)
    {
    syslog(LOG_ERR, "EOVERFLOW Message limit reach. \n");
    return 2;
    }

    if (errsv == EBUSY)
    {
    syslog(LOG_WARNING, "EBUSY Open /dev/monreader failed. Device has already been opened by another user. \n");
    return 2;
    }
    return 0;
}
// end exepctions
```

W międzyczasie program zpmd odczytuje z bufora nagłówek rekordu, w którym to m.in. zawarta jest informacja o identyfikatorze domeny i rekordzie. Na tej podstawie jest w stanie nałożyć odpowiednią strukturę na pamięć umożliwiając odczytanie z niej dowolnego pola.

Również zpmd dla każdego przypadku konwertuje czas utworzenia danego rekordu z formatu TOD (Time of Day) do formatu UTC oraz jeśli jest taka konieczność konwertuje znaki w formacie EBCDIC na ASCII. Po czym przesyła wszystkie pola ze struktury (omijając zarezerwowane przez IBM) do odpowiedniej tabeli w bazie

danych MySQL, wykorzystując do tego kwerendę „INSERT”. Po udanej operacji zerowany jest bufor i tymczasowe zmienne, by przygotować się do wczytania kolejnego rekordu.

Autor podczas pisania nagłówka e2a.h skorzystał z gotowego kodu źródłowego znajdującego się w kodach źródłowych jądra systemu GNU/Linux. W pliku e2a.h znajduje się funkcja konwertująca znaki z formatu EBCDIC do ASCII. Zawartość tablicy ebcdic2ascii[256] zaczerpnięto zgodnie z licencją GPL z katalogu „arch/s390/kernel/ebcdic.c” umieszczając w nagłówku komentarz o autorach.

4.1.2 Interfejs Programistyczny dla usługi *MONITOR systemu z/VM V5R3.0

Struktura rekordów oraz ich funkcjonalność zostały zaczerpnięte z udostępnionej dokumentacji znajdującej się na stronie internetowej firmy IBM. Strona została dołączona do opracowania i można ją znaleźć w dziale bibliografia.

W poniższej tabeli uwzględniono całkowitą pulę rekordów w domenie dla systemu z/VM V5R3.0 oraz liczbę rekordów obsługiwanych (słowo „obsługiwanych” należy rozumieć jako stabilnych i poprawnie działających) przez program zpmd:

Nr Domeny	Nazwa Domeny	Całkowita Pula Rekordów	Ilość Obsługiwanych Rekordów
0	System	24	13
1	Monitor	20	9
2	Scheduler	12	8
3	Storage	20	20
4	User	10	4
5	Processor	10	9
6	I/O	30	18
7	Seek	1	1
8	Virtual Network	3	3
10	Application Data	2	0

Tab. 2. Liczba obsługiwanych rekordów przez zpmd dla z/VM V5R3.0

W tabeli nr 2, w kolumnie „Ilość Obsługiwanych Rekordów” przy zliczaniu uwzględniono wyłącznie rekordy, do których Autor miał całkowitą pewność, że są bezbłędnie zaimplementowane. Przy występujących wątpliwościach płynących z nieścisłych opisów w dokumentacji podpierano się wynikami uzyskanymi z pracy programu zpmd. Pozostałe rekordy uznane zostały za niestabilne i nie są brane pod uwagę.

W programie zpmd dla każdej jednorazowej porcji danych, jaka jest przekazywana do bufora stworzono następujące struktury:

- MCE (Monitor Control Element)
- Header
- Osobną strukturę dla każdego rekordu nazwaną „dXrY”, gdzie „X” jest numerem domeny, a „Y” numerem rekordu

MCE jest strukturą zawierającą typ aktualnego rekordu („S” dla rekordu typu SAMPLE oraz „E” dla rekordu typu EVENT) wraz z numerem domeny do której należy, posiada również zarezerwowany przez firmę IBM obszar, jak i początek wraz z końcem obszaru pamięci w której znajduje się rekord.

```
struct mce
{
  __u8 rec_type;
  __u8 domains;
  __u16 ibm;
  __u32 rec_start;
  __u32 rec_end;
};
```

Header jest to struktura nagłówka należącego do każdego z rekordów. Zawiera długość rekordu, obszar wypełniony zerami, numer domeny, obszar zarezerwowany przez firmę IBM, numer rekordu i czas utworzenia danego rekordu.

```
struct header
{
```

```
  __u16 rec_length;
  __u16 zeros;
  __u8 domain_id;
  __u8 ibm;
  __u16 rec_id;
  __eu64 build_time;
  __u32 ibm2;
};
```

Struktury rekordów dla każdej z domen są zróżnicowane. Jednym z mniejszych rekordów jest rekord „d1r1” zawierający aktualnie włączone domeny dla rekordów typu EVENT:

```
struct d1r1
{
   __u8 filler[20];
   __u8 mtrepr_edomains;
   __u8 ibm;
   __u16 mtrepr_config;
};
```

Każda struktura zdefiniowana dla rekordu zawiera pole „filler”. Filler ma długość 20 bajtów. Nakłada się na obszar nagłówka i pozwala na dopasowanie reszty pól do pamięci z danymi.

Kolejne pola są zależne od danego typu domeny i rekordu. W dużej ilości rekordów często powtarzają się pola o nazwie „ibm”, bądź „ibmX”, gdzie „X„ jest numerem dalszego w kolejności pola „ibm”. Nazwa to oznacza, że dane pole zawiera obszar zarezerwowany na potrzeby firmy IBM.

Przy deklaracji użyto typów zmiennych z nagłówka types.h, ponieważ rozmiar zmiennych w strukturach musi być ściśle określony. W typie zmiennej znak „u” oznacza „unsigned”, a „s” to „signed”, następna wartość to liczba bitów.

W źródłach programu używana jest nazwa „ibm” dla pól oznaczanych w dokumentacji jako „*”. Istnieje możliwość wystąpienia pola, gdzie najczęściej jest to pole o rozmiarze 1 bajtu z zarezerwowanymi „*” kilkoma bitami. Ma to miejsce w strukturze o nazwie „d1r1”. W polu „ibm” część informacji została udokumentowana

i jest wykorzystywana przez program:

```
21  15  Bitstring    1  *
      1... ....     MTREPR_EDOVNT         Virt. Net. domain act.
      .1.. ....     *
      ..1. ....     MTREPR_EDOMAPL        Appldata domain active
      ...1 ....     *
      .... 1...     *
      .... .1..     *
      .... ..1.     *
      .... ...1     *
```

Widać więc, że dwa bity zostały udostępnione i udokumentowane. Reszta jest wciąż zarezerwowana.

Podczas opracowywania struktur rekordów Autor napotykał trudności z odczytywaniem samej dokumentacji, która nie trzyma się jednego standardu oraz z powodu występujących w kilku miejscach braków, bądź niedokończonych opisów pól. Co skutkowało testowaniem, a więc porównywaniem każdego wątpliwego, pod względem deklaracji pola w strukturze z wynikami otrzymanymi z urządzenia „/dev/monreader" lub wpisami w samej bazie danych. W przypadku niemożności sprawdzenia poprawności danego rekordu, rekord był pomijany.

Format zapisu czasu dla rekordów to TOD (Time Of Day) zajmujący obszar 8 bajtów (64 bity). Ze względów na obecnie przyjęty standard UTC (Universal Time Clock) zaimplementowano konwersję do nowego formatu.

Z racji przetwarzania zmiennych 64 bitowych na 31 bitowym systemie operacyjnym skorzystano z możliwości kompilatora gcc i do konwersji czasu zadeklarowano zmienną „__uint64_t". Dla wygody przedefiniowano ją w kodzie na „__eu64".

```
__eu64 tod2utc(__eu64 time)
{
   time = (__eu64) time - (__eu64) 0x7d91048bca000000ULL;
   time = time >> 12;
   time = time / 1000000;
   return time;
```

```
};
```

Powyższy kod przedstawia konwersję formatów TOD na UTC. Wartość „0x7d91048bca000000” jest datą rozpoczęcia naliczania czasu dla formatu UTC zapisaną w formacie TOD. Odejmuje się ją od aktualnego czasu, gdyż format TOD był naliczany kilkadziesiąt lat wcześniej zanim format UTC został wprowadzony. Po tym zabiegu należy jeszcze przesunąć wszystkie bity o 12 w prawo, co spowoduje usunięcie 12 bitów, w których zapisane były informacje niezwiązane z czasem.

Następnym krokiem jest przypisanie rezultatu zwracanego przez funkcję tod2utc() do 32 bitowej zmiennej:

```
__u32 utc;
utc = (__eu64) tod2utc( (__eu64) hptr->build_time);
```

Zmienna „utc” w formie 32 bitowej jest przekazywana do bazy danych MySQL, która dokonuje translacji za pomocą funkcji FROM_UNIXTIME() do wartości TIMESTAMP.

Rekordy wychodzące z programu zpmd zapisywane są w bazie danych według wspomnianego dla struktur schematu „dXrY”. Z pominięciem pól filler i ibm.

Przykładowy kod tworzący dwie tabele w bazie danych:

```
const char *query_create_d1r1 =
"CREATE TABLE IF NOT EXISTS d1r1 (\
id INT AUTO_INCREMENT, \
time TIMESTAMP UNIQUE, \
mtrepr_edomains TINYINT(1) UNSIGNED, \
ibm TINYINT(1) UNSIGNED, \
mtrepr_config SMALLINT(1) UNSIGNED, \
PRIMARY KEY (id), \
INDEX (id), \
INDEX (time DESC)) ENGINE=INNODB";

const char *query_create_d6r2 =
"CREATE TABLE IF NOT EXISTS d6r2 (\
```

```
id INT AUTO_INCREMENT, \
time TIMESTAMP, \
iodvof_rdevsid INT(1) UNSIGNED, \
iodvof_rdevdev SMALLINT(1) UNSIGNED, \
PRIMARY KEY (id), \
INDEX (id), \
UNIQUE INDEX (time, iodvof_rdevsid)) ENGINE=INNODB";
```

W powyższym kodzie zauważyć można że każda tabela posiada unikalne pole, bądź unikalny indeks. Powoduje to niedopuszczenie do sytuacji by w tabeli istniały identyczne rekordy. Chroni to dane przed ewentualnymi błędami programu zpmd, wynikających z nieodkrytych błędów w programie, bądź niezamierzonych błędów użytkowników.

Kod źródłowy odpowiadający za przekazywanie danych do bazy MySQL wygląda następująco:

- dla tabeli „d1r1”:

```
//SQL QUERY
        sprintf(qbuf, "INSERT INTO d1r1 VALUES (",FROM_UNIXTIME(%lu),'%u','%u','%u')",
                utc,
                d1r1p->mtrepr_edomains,
                d1r1p->ibm,
                d1r1p->mtrepr_config);

        if (mysql_query(&mysql, qbuf))
        {
           syslog(LOG_ERR, "%s \n", mysql_error(&mysql));
        }
```

- dla tabeli „d2r3”:

```
//EBCDIC 2 ASCII
        e2a(d2r3p->sclwrr_vmduser, 8);
//SQL QUERY
        sprintf(qbuf, "INSERT INTO d2r3 VALUES (",FROM_UNIXTIME(%lu),'%s','%u','%u','%u')",
```

```
                utc,
                d2r3p->sclwrr_vmduser,
                d2r3p->sclwrr_calflags,
                d2r3p->sclwrr_rdevsid,
                d2r3p->sclwrr_calbyct);

if (mysql_query(&mysql, qbuf))
{
   syslog(LOG_ERR, "%s \n", mysql_error(&mysql));
}
```

4.2 zpm (z/VM Performance Monitor)

Program zpm (z/VM Performance Monitor) jest aplikacją napisaną w języku programowania Java, dzięki czemu jest niezależny od platformy, na której jest uruchamiany. Ściśle współpracuje z programem zpmd (generowanym przez niego formatem tabel). Do połączenia z bazą MySQL użyto interfejsu JDBC (Java DataBase Connectivity) oraz sterownika MySQL JDBC Driver (mysql-Connector-java-5.1.6). Program przeznaczony jest dla końcowego użytkownika, z którym w prosty i przejrzysty sposób komunikuje się za pomocą GUI (Graphical User Interface) w języku angielskim.

4.2.1 Funkcjonalność

Aplikacja umożliwia przedstawienie informacji potrzebnych do nadzorowania wirtualnych maszyn uruchomionych w systemie z/VM, jak i całego systemu. Wykorzystując bibliotekę JFreeChart-1.0.12 oraz JCommon-1.0.15 program jest w stanie generować wykresy i statystyki.

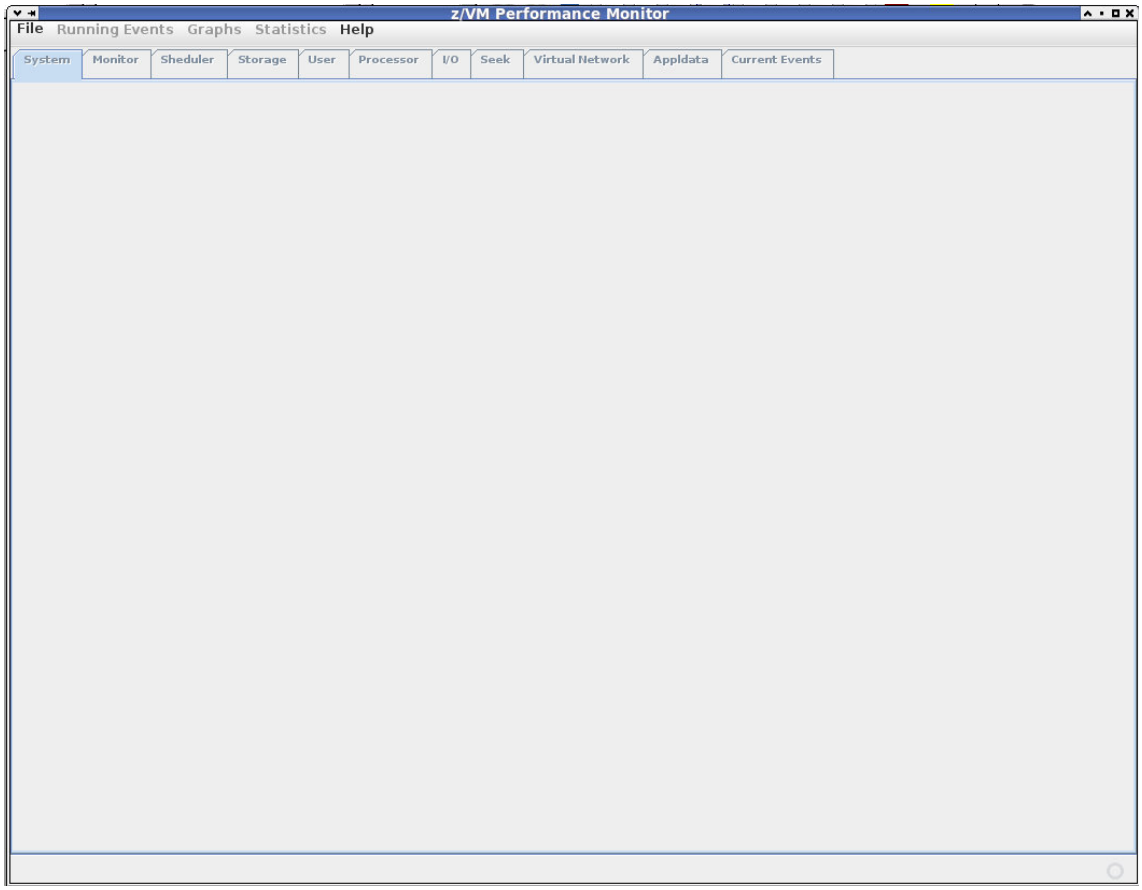
Cechy programu zpm v0.6:

- Kod programu w języku Java
- Wykorzystuje interfejs JDBC do komunikacji z baza MySQL
- Wykorzystuje biblioteki JFreeChart, JCommon do tworzenia dynamicznych wykresów
- Wielowątkowy
- Multiplatformowy
- Komunikuje się z użytkownikiem w języku angielskim
- Kod źródłowy na licencji GPL

Funkcjonalność programu zpm v0.6 :

- Graficzny interfejs użytkownika
- Monitorowanie bieżących zdarzeń (ang. events) z przedziałem czasowym definiowalnym przez użytkownika
- Prezentowanie szczegółowych informacji o systemie
- Prezentowanie szczegółowych informacji z możliwością wybierania przez użytkownika przedziałów czasowych
- Dynamiczne generowanie:
    - Wykresów słupkowych
    - Wykresów typu PieChart
    - Statystyk

Aplikacja jest przeznaczona dla monitorów pracujących z rozdzielczością powyżej 1024x768 pikseli, ponieważ główny panel jak i generowane statystyki oraz wykresy uruchamiają się z właśnie takim nominalnym rozmiarem okna.



Rys. nr. 1 Główne okno aplikacji

W skład głównego panelu wchodzi:

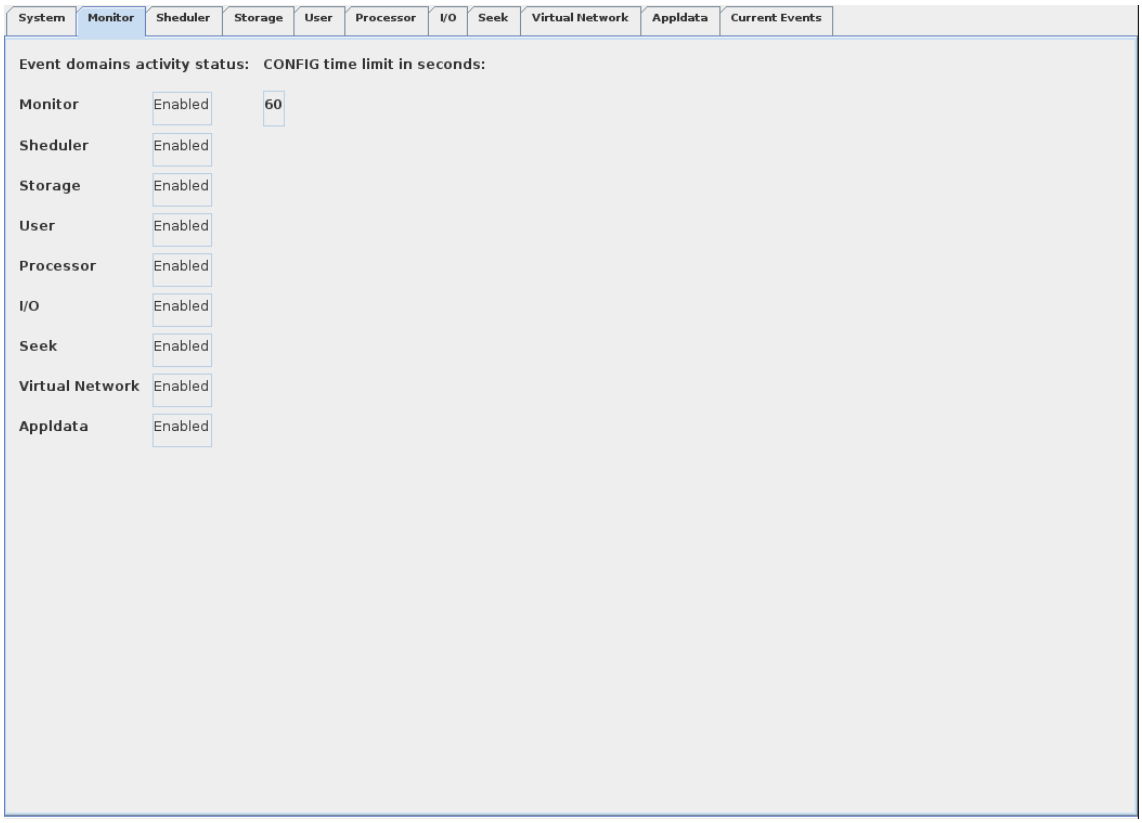
- jMenuBar

Rys. nr. 2 Okno menu

- jTabbedPane zawierający jPanel'e o nazwie:

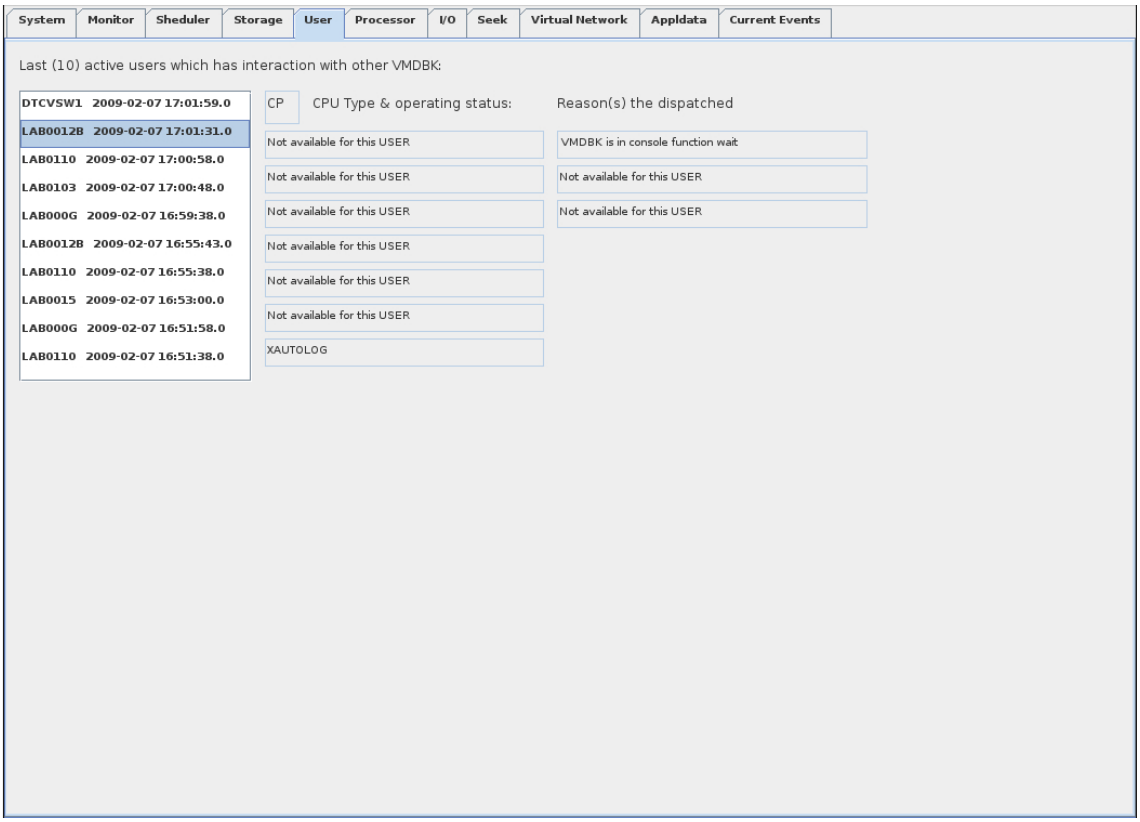
Rys. nr. 3 Okno zakładek

  - System
  - Monitor



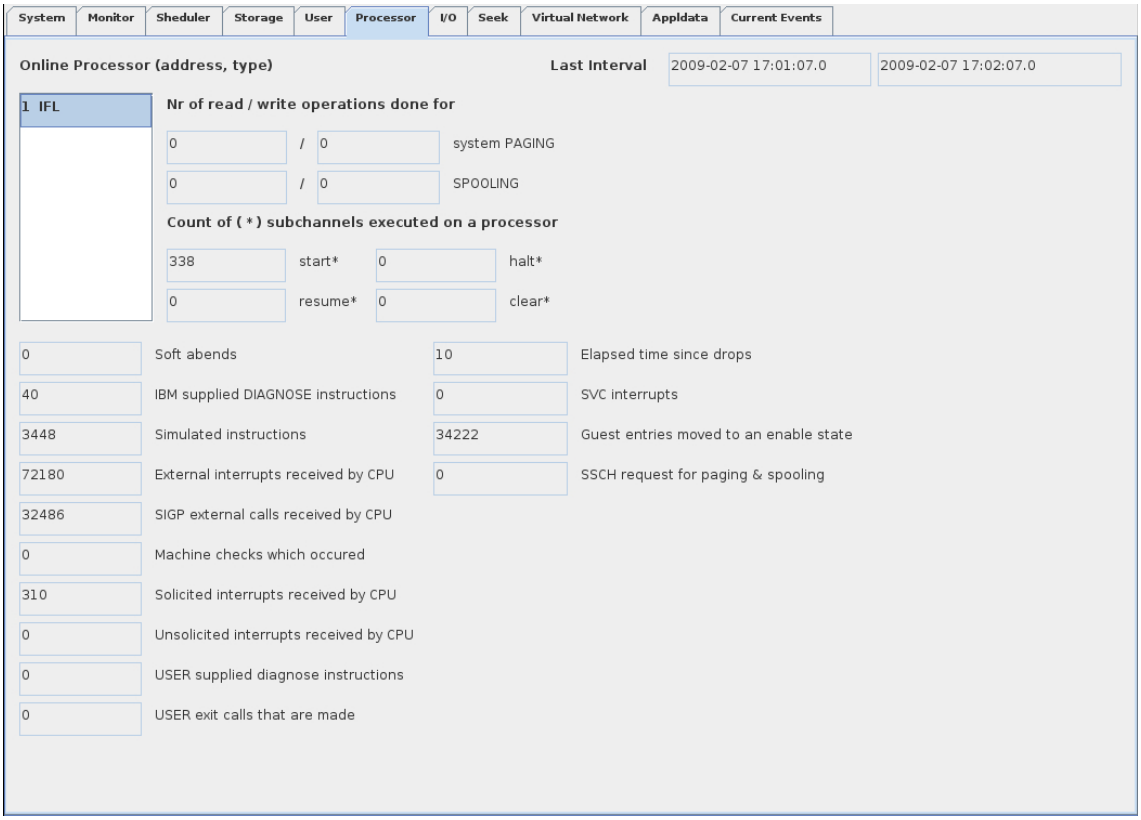
Rys. nr. 4 Zakładka Monitor

- Sheduler
- Storage
- User



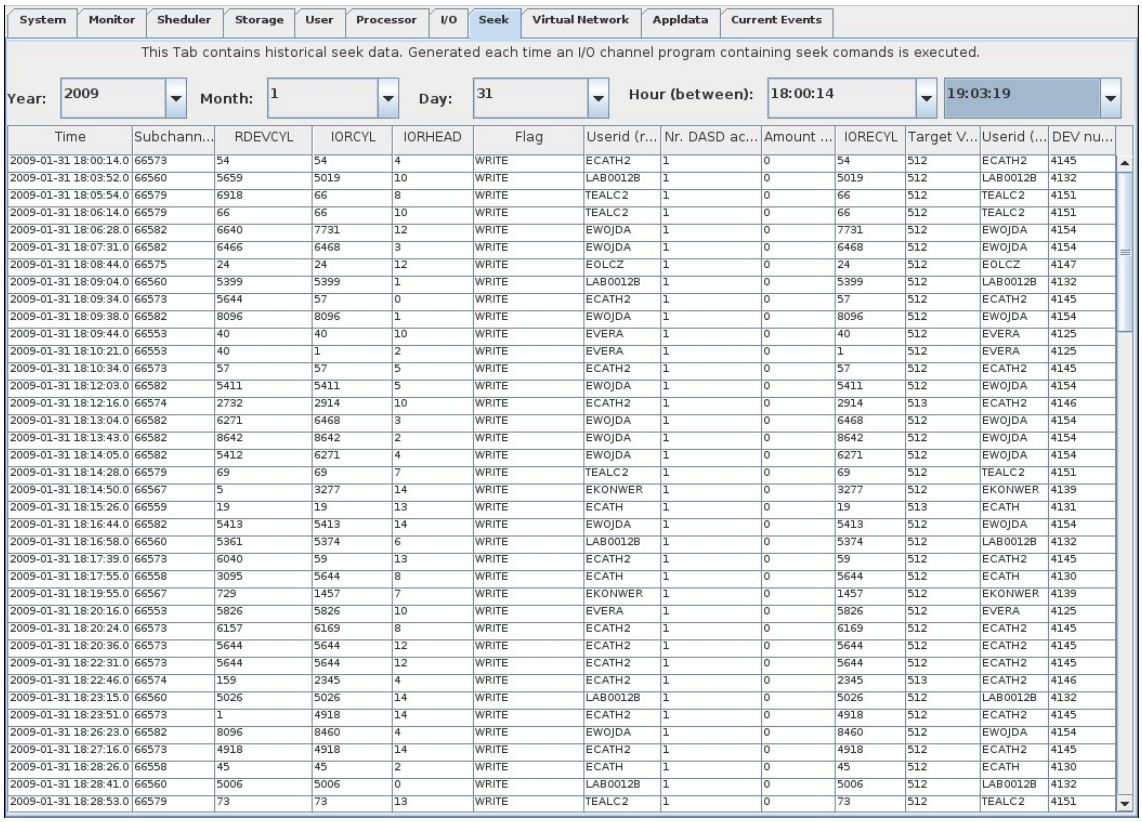
Rys. nr. 5 Zakładka User

- Processor



Rys. nr. 6 Zakładka Processor

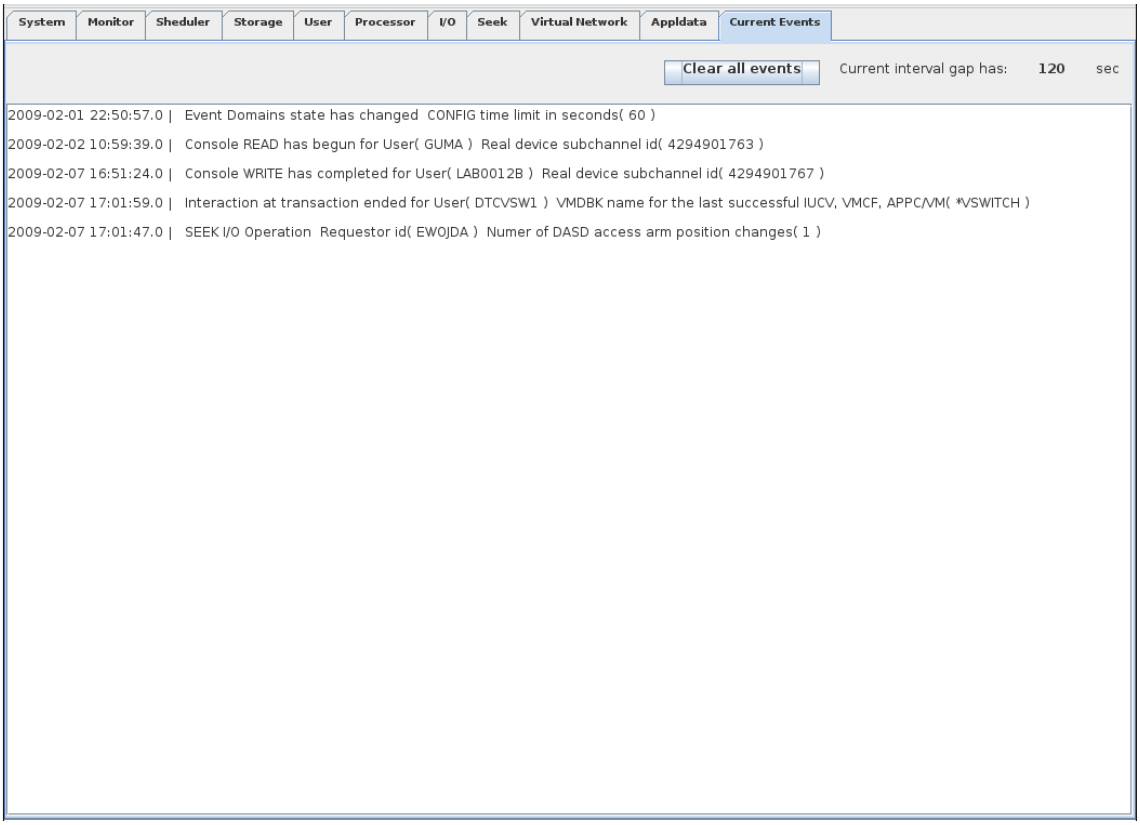
- I/O
- Seek



Time	Subchann...	RDEVCYL	IORCYL	IORHEAD	Flag	Userid (r...	Nr. DASD ac...	Amount ...	IORECYL	Target V...	Userid (...	DEV nu...
2009-01-31 18:00:14.0	66573	54	54	4	WRITE	ECATH2	1	0	54	512	ECATH2	4145
2009-01-31 18:03:52.0	66560	5659	5019	10	WRITE	LAB0012B	1	0	5019	512	LAB0012B	4132
2009-01-31 18:05:54.0	66579	6918	66	8	WRITE	TEALC2	1	0	66	512	TEALC2	4151
2009-01-31 18:06:14.0	66579	66	66	10	WRITE	TEALC2	1	0	66	512	TEALC2	4151
2009-01-31 18:06:28.0	66582	6640	7731	12	WRITE	EWOJDA	1	0	7731	512	EWOJDA	4154
2009-01-31 18:07:31.0	66582	6466	6468	3	WRITE	EWOJDA	1	0	6468	512	EWOJDA	4154
2009-01-31 18:08:44.0	66575	24	24	12	WRITE	EOLCZ	1	0	24	512	EOLCZ	4147
2009-01-31 18:09:04.0	66560	5399	5399	1	WRITE	LAB0012B	1	0	5399	512	LAB0012B	4132
2009-01-31 18:09:34.0	66573	5644	57	0	WRITE	ECATH2	1	0	57	512	ECATH2	4145
2009-01-31 18:09:38.0	66582	8096	8096	1	WRITE	EWOJDA	1	0	8096	512	EWOJDA	4154
2009-01-31 18:09:44.0	66553	40	40	10	WRITE	EVERA	1	0	40	512	EVERA	4125
2009-01-31 18:10:21.0	66553	40	1	2	WRITE	EVERA	1	0	1	512	EVERA	4125
2009-01-31 18:10:34.0	66573	57	57	5	WRITE	ECATH2	1	0	57	512	ECATH2	4145
2009-01-31 18:12:03.0	66582	5411	5411	5	WRITE	EWOJDA	1	0	5411	512	EWOJDA	4154
2009-01-31 18:12:16.0	66574	2732	2914	10	WRITE	ECATH2	1	0	2914	513	ECATH2	4146
2009-01-31 18:13:04.0	66582	6271	6468	3	WRITE	EWOJDA	1	0	6468	512	EWOJDA	4154
2009-01-31 18:13:43.0	66582	8642	8642	2	WRITE	EWOJDA	1	0	8642	512	EWOJDA	4154
2009-01-31 18:14:05.0	66582	5412	6271	4	WRITE	EWOJDA	1	0	6271	512	EWOJDA	4154
2009-01-31 18:14:28.0	66579	69	69	7	WRITE	TEALC2	1	0	69	512	TEALC2	4151
2009-01-31 18:14:50.0	66567	5	3277	14	WRITE	EKONWER	1	0	3277	512	EKONWER	4139
2009-01-31 18:15:26.0	66559	19	19	13	WRITE	ECATH	1	0	19	513	ECATH	4131
2009-01-31 18:16:44.0	66582	5413	5413	14	WRITE	EWOJDA	1	0	5413	512	EWOJDA	4154
2009-01-31 18:16:58.0	66560	5361	5374	6	WRITE	LAB0012B	1	0	5374	512	LAB0012B	4132
2009-01-31 18:17:39.0	66573	6040	59	13	WRITE	ECATH2	1	0	59	512	ECATH2	4145
2009-01-31 18:17:55.0	66558	3095	5644	8	WRITE	ECATH	1	0	5644	512	ECATH	4130
2009-01-31 18:19:55.0	66567	729	1457	7	WRITE	EKONWER	1	0	1457	512	EKONWER	4139
2009-01-31 18:20:16.0	66553	5826	5826	10	WRITE	EVERA	1	0	5826	512	EVERA	4125
2009-01-31 18:20:24.0	66573	6157	6169	8	WRITE	ECATH2	1	0	6169	512	ECATH2	4145
2009-01-31 18:20:36.0	66573	5644	5644	12	WRITE	ECATH2	1	0	5644	512	ECATH2	4145
2009-01-31 18:22:31.0	66573	5644	5644	12	WRITE	ECATH2	1	0	5644	512	ECATH2	4145
2009-01-31 18:22:46.0	66574	159	2345	4	WRITE	ECATH2	1	0	2345	513	ECATH2	4146
2009-01-31 18:23:15.0	66560	5026	5026	14	WRITE	LAB0012B	1	0	5026	512	LAB0012B	4132
2009-01-31 18:23:51.0	66573	1	4918	14	WRITE	ECATH2	1	0	4918	512	ECATH2	4145
2009-01-31 18:26:23.0	66582	8096	8460	4	WRITE	EWOJDA	1	0	8460	512	EWOJDA	4154
2009-01-31 18:27:16.0	66573	4918	4918	14	WRITE	ECATH2	1	0	4918	512	ECATH2	4145
2009-01-31 18:28:26.0	66558	45	45	2	WRITE	ECATH	1	0	45	512	ECATH	4130
2009-01-31 18:28:41.0	66560	5006	5006	0	WRITE	LAB0012B	1	0	5006	512	LAB0012B	4132
2009-01-31 18:28:53.0	66579	73	73	13	WRITE	TEALC2	1	0	73	512	TEALC2	4151

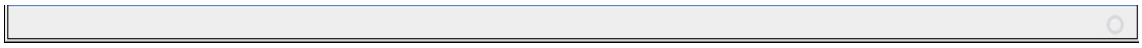
Rys. nr. 7 Zakładka Seek

- Virtual Network
- Appldata
- Current Events



Rys. nr. 8 Zakładka Current Events

- StatusPanel

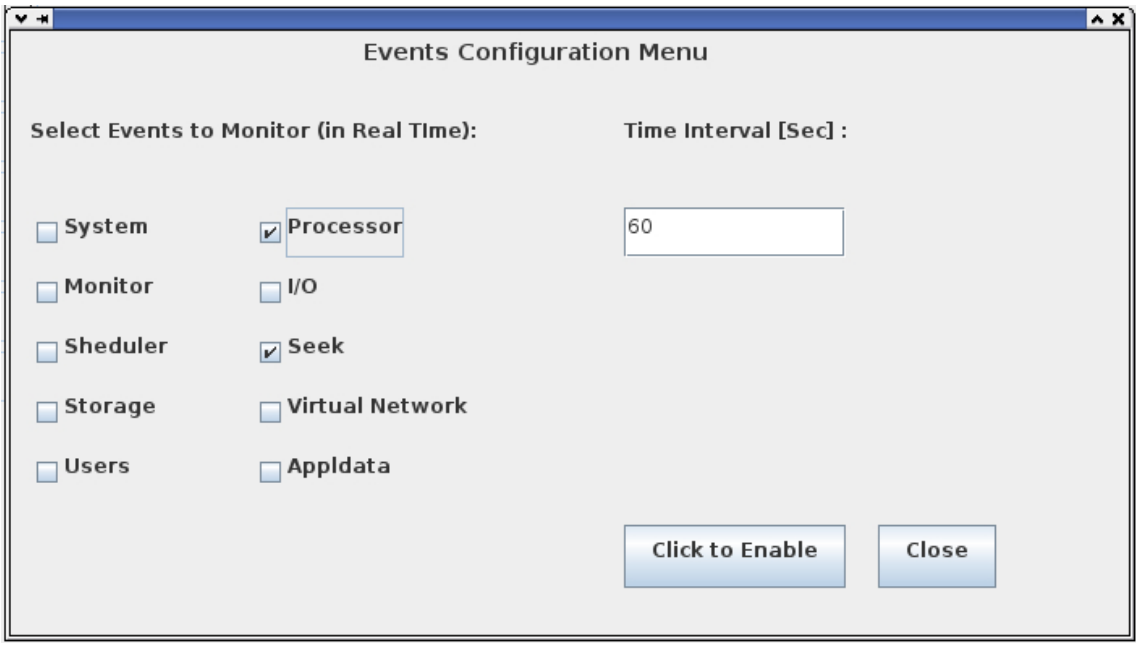


Rys. nr. 9 Panel statusu

Po uruchomieniu programu aktywne są tylko trzy przyciski znajdujące się w jMenuBar. Pierwszym z nich jest przycisk „Connect to the Database” pozwalający nawiązać połączenie ze zdalną bazą danych, a następnie aktywujący pozostałe przyciski oraz panele. Drugim przyciskiem jest „About” w menu „Help” zawierający informacje o programie i jego autorze. Trzecim natomiast „Exit” kończący pracę aplikacji.

Po nawiązaniu połączenia z bazą bazą danych Przycisk „Connect to the Database” zostaje przełączony w tryb disabled co powoduje że możliwe jest utworzenie tylko jednego połączenia z bazą danych, z którego (na przemian) korzystają wszystkie panele (zakładki) wchodzące w skład jTabbedPane. Jednym wyjątkiem jest Panel o nazwie „Current Events” konfigurowalny i uruchamiany z poziomu górnego paska menu w „Running Events” w menu kryjącym się pod przyciskiem „Settings”. Po wciśnięciu na nim przycisku „Click to Enable” zostaje uruchomiony osobny wątek nawiązujący

kolejne połączenie ze zdalną bazą danych, korzystając przy tym z ustawień pierwotnie wpisanych dla pierwszego połączenia. Zadaniem wątku jest śledzenie zmian w wybranych przez użytkownika tabelach co pewien interwał czasowy, aby następnie powiadomić końcowego użytkownika o najnowszych zdarzeniach przez panel „Current Events" gdzie wyświetlane są komunikaty w polu jTextArea.



Rys. nr. 10 Current Events Configuration Menu

W zakładce „Monitor" widnieje aktualny status dla wszystkich domen typu EVENT. Pobranie danych i pokazanie ich na panelu odbywa się podczas wyświetlenia jPanel'u, a więc podczas kliknięcia na zakładkę „Monitor". Do ponownego odświeżenia zawartości dojdzie, gdy użytkownik zmieni zakładkę na inną, po czym ponownie wróci na zakładkę „Monitor".

Zakładka „User" zawiera panel wyświetlający m.in w formacie listy (jList) ostatnich dziesięciu użytkowników, którym interakcja z inną wirtualną maszyną zakończyła się pomyślnie. Dodatkowo opisuje Typ procesora, status operacji oraz powód wysłania informacji.

Następną z kolei jest zakładka o nazwie „Processor" prezentująca wszystkie dostępne w trybie on-line procesory w systemie z/VM. W komponencie jList zostaje wyświetlony adres i typ procesora. Po kliknięciu na wybrany obiekt (procesor) następuje pobranie z bazy danych (ze względu na format danych) dwóch, najnowszych

rekordów dla wybranego adresu procesora, po czym od wartości w rekordzie o dacie najświeższej odjęcie wszystkich wartości z rekordu z datą wcześniejszą. Tak otrzymane wartości przekazywane są do pól typu jTextField. Pomimo informacji stricte powiązanych z danym procesorem widnieje pole z datami w formie „od A do B", gdzie A jest datą pobraną z rekordu starszego, a B z rekordu najświeższego. Rozwiązanie to pozwala na lepsze zorientowanie się użytkownikowi z jakim przedziałem czasu ma do czynienia.

Ostatnią zakładką jest „Seek". Zawiera informacje generowane za każdym razem, gdy program kanałowy dla I/O zawierający komendę „seek" jest wykonywany. Ze względu na wysokie (pod względem częstotliwości) generowanie rekordów zaimplementowano przedstawienie danych w formie tabeli. Użytkownik by wygenerować interesującą go porcję informacji zmuszony jest do wybrania z pól typu JcomboBox w kolejności: roku, miesiąca, dnia, godziny (godzinę początkową oraz godzinę końcową). Po czym następuje dynamiczne wygenerowanie tabeli.

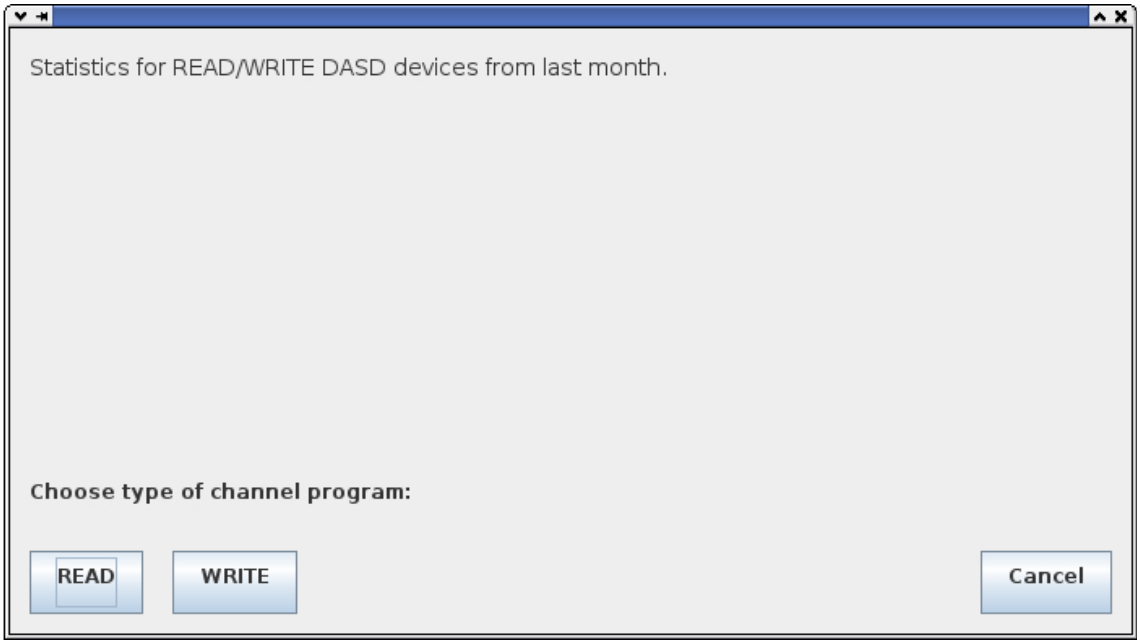
Pozostałe zakładki pozostały niewykorzystane, ponieważ usługa *MONITOR systemu z/VM nie generowała dla nich rekordów. Co uniemożliwiało testowanie i poprawną implementację zwiększenia funkcjonalności programu.

Wykresy statystyk zostały umieszczone osobno na górnym pasku menu pod przyciskiem „Statistics". Wszystkie wykresy w aplikacji uruchamiane są jako nowe wątki tworzące osobne połączenia z bazą danych. Rozwiązanie to uniemożliwia zamarzaniu głównego okna aplikacji oraz pozwala na odizolowane pobieranie i przetwarzanie danych.

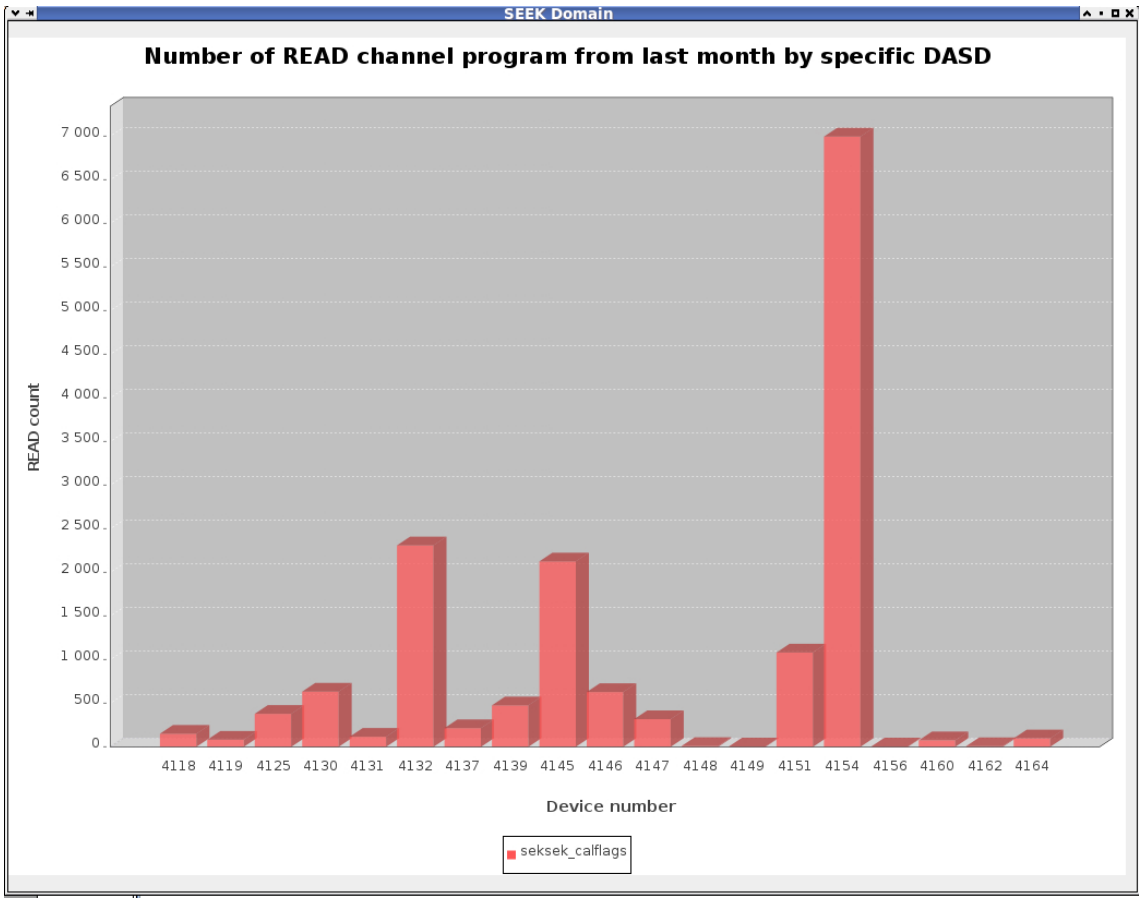
Pod przyciskiem „Statistics" mieszczą się wyresy:

- DASD Seek I/O
- VMDDISPL Queue
- VMDELIG Queue
- Read/Write operations for all processors

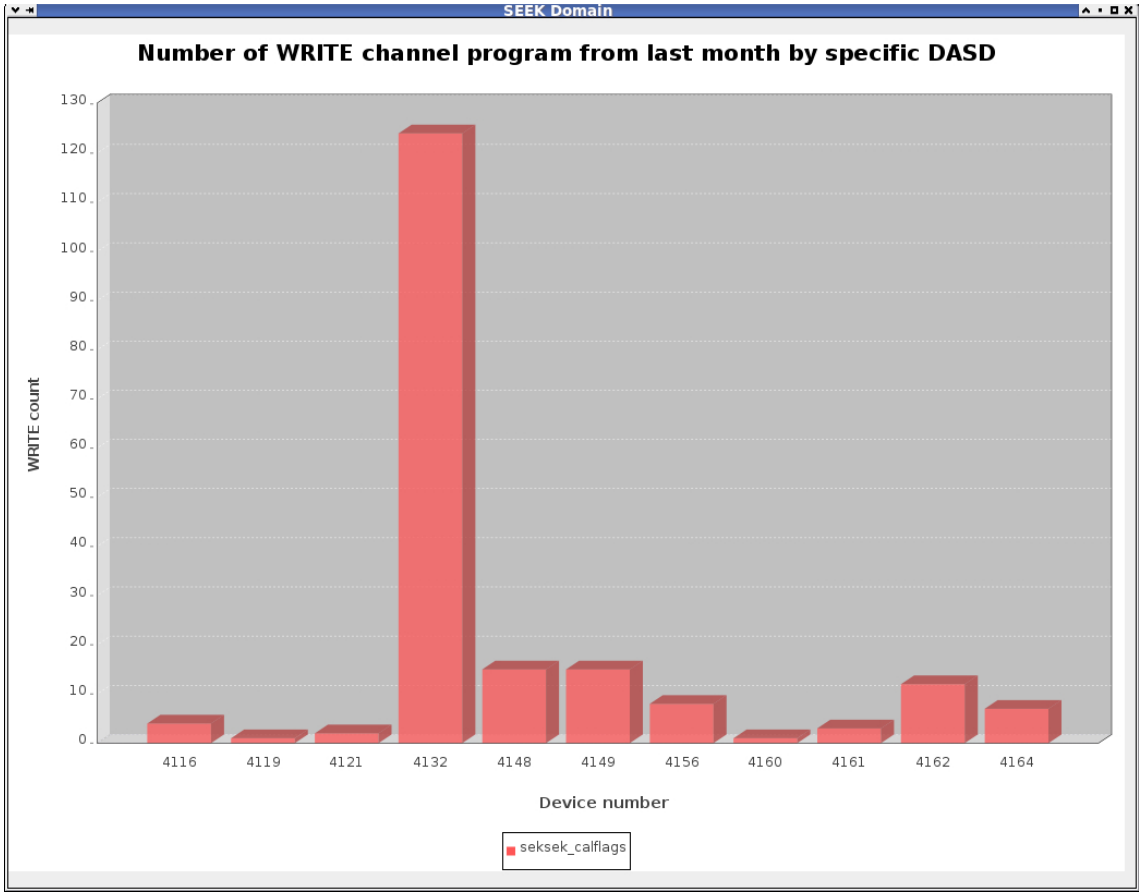
Użytkownik wybierając opcję „DASD Seek I/O" jest w stanie podejrzeć statystyki odczytu i zapisu zgromadzone w ostatnim miesiącu dla wszystkich dysków DASD w postaci słupków w 3D.



Rys. nr. 11 Panel wyboru wykresu do wygenerowania

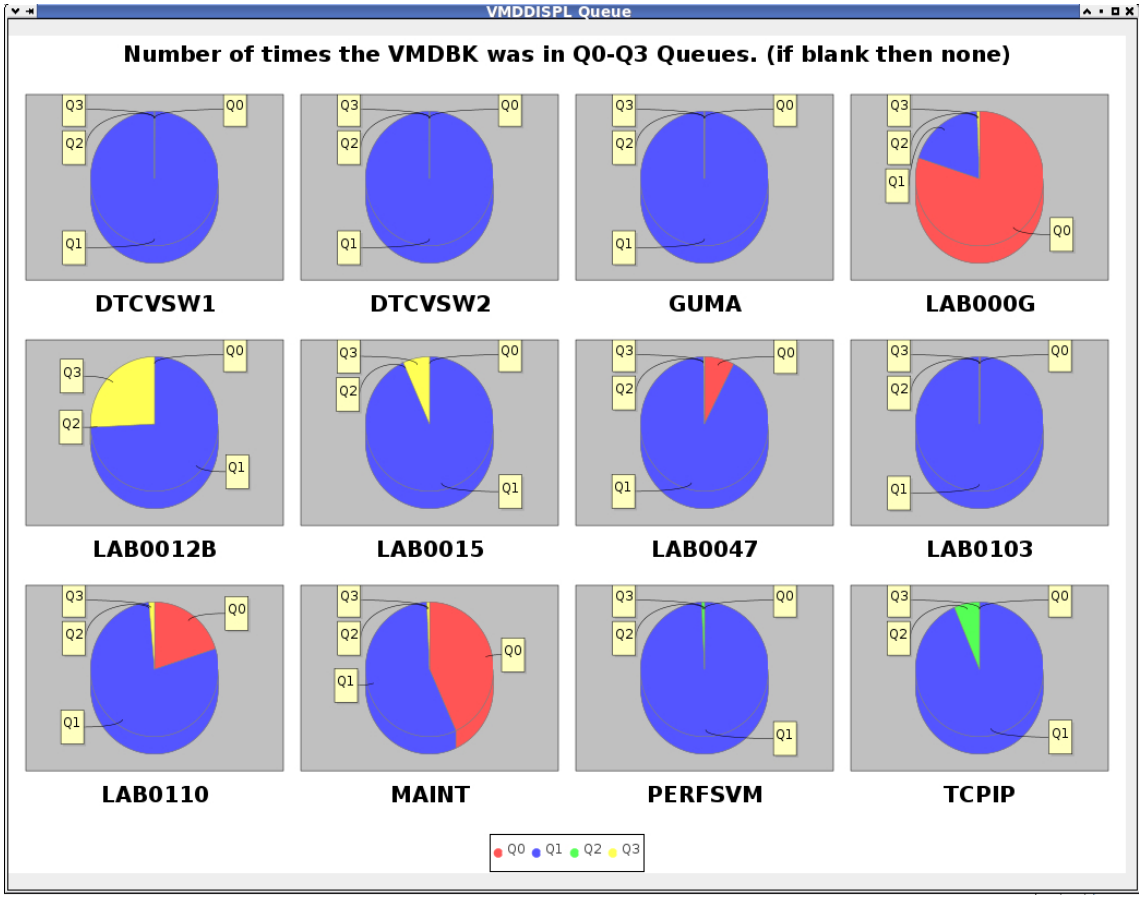


Rys. nr. 12 READ count by device number

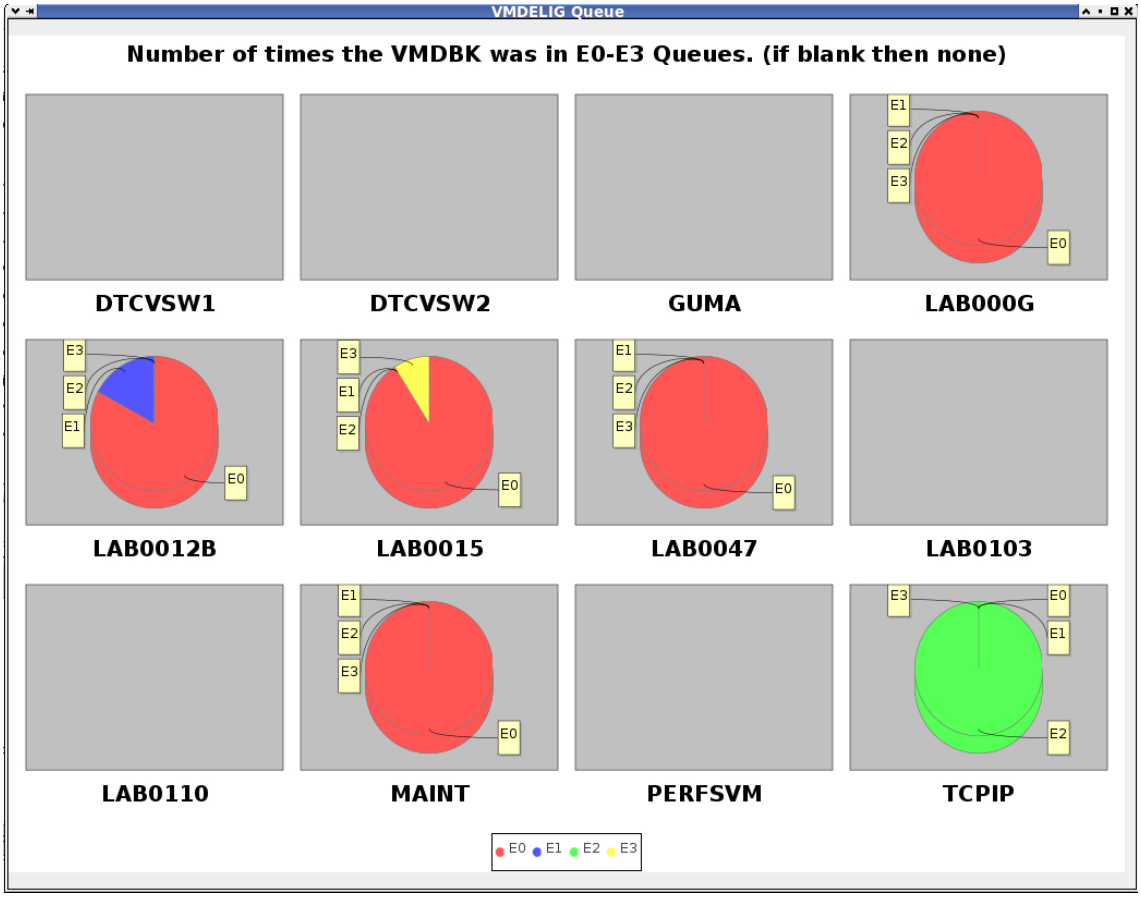


Rys. nr. 13 WRITE count by device number

Kolejne opcje to „VMDDISPL Queue” oraz „VMDELIG Queue” wyświetlające w postaci Multiple PieChart 3D (dla każdej wirtualnej maszyny osobny PieChart mieszczący się w tym samym oknie aplikacji). Pobierane są najświeższe wpisy z bazy danych dotyczące wszystkich wirtualnych maszyn po czym tworzony jest osobny PieChart składający się z czterech wartości. Dla VMDDISPL są to Q0, Q1, Q2, Q3, dla VMDELIG są to E0, E1, E2, E3.

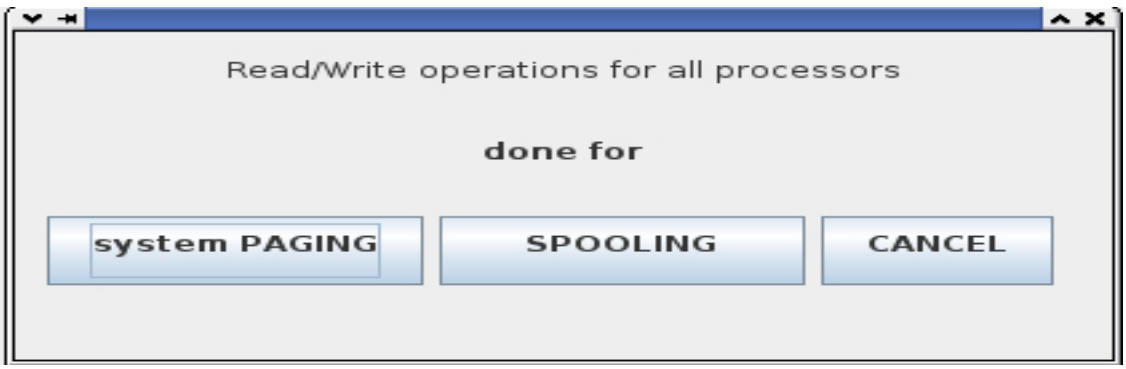


Rys. nr. 14 VMDBK Q0-Q3 Queues

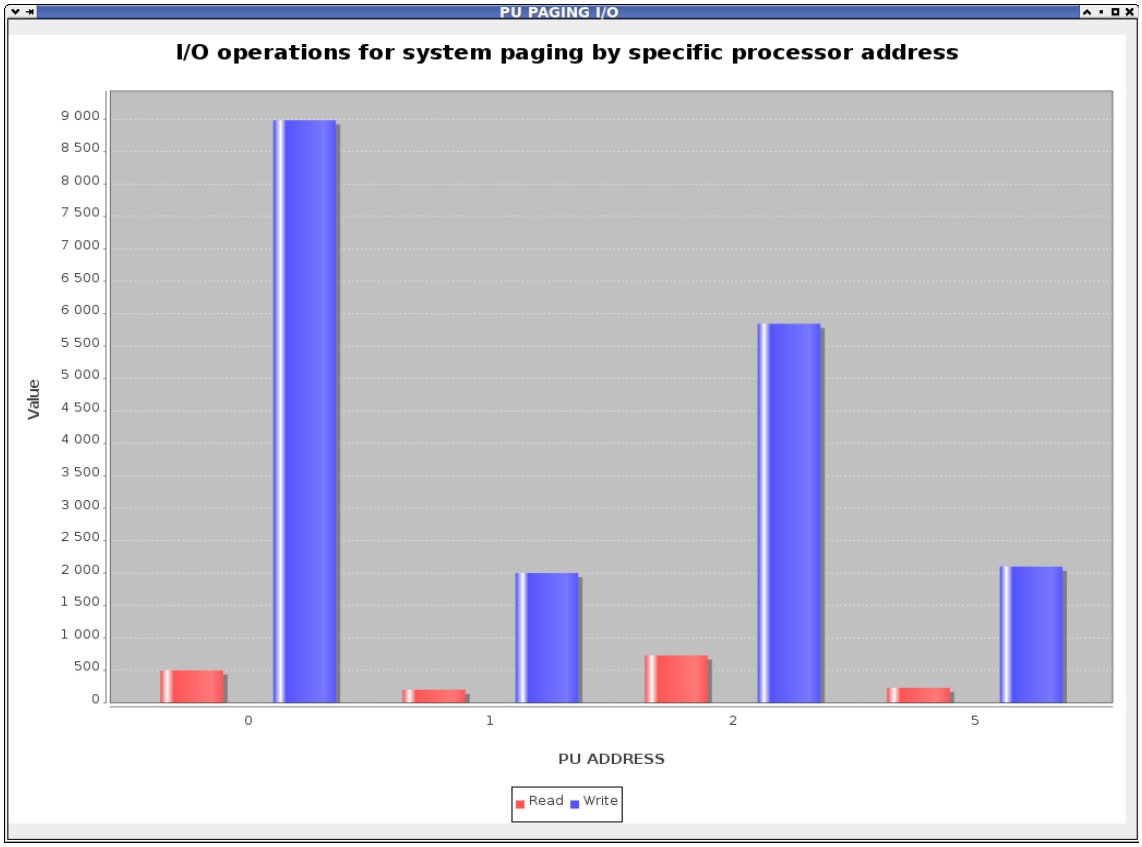


Rys. nr. 15 VMDBK E0-E3 Queues

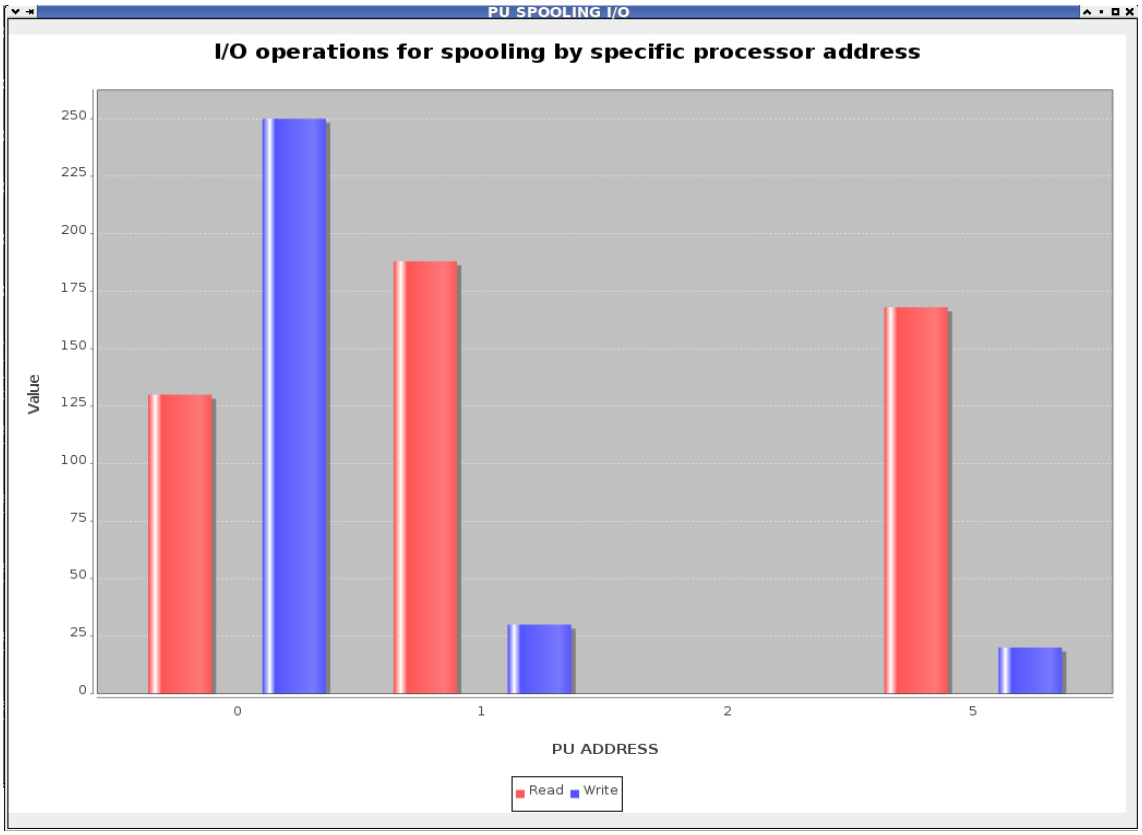
Ostatni przycisk wyświetla panel wyboru pomiędzy dwoma grupowymi wykresami, w skład każdego z nich wchodzą wszystkie dostępne procesory. Wyświetlane są dla nich dwie wartości „odczyt” i „zapis”. Dla wybranej opcji „Read/Write operations for all processors” użytkownik dostaje do wyboru wygenerowanie najświeższego wykresu dla operacji odczytu/zapisu dla systemowego page'a lub spool'a.



Rys. nr. 16 Read/Write operations for all processors Menu



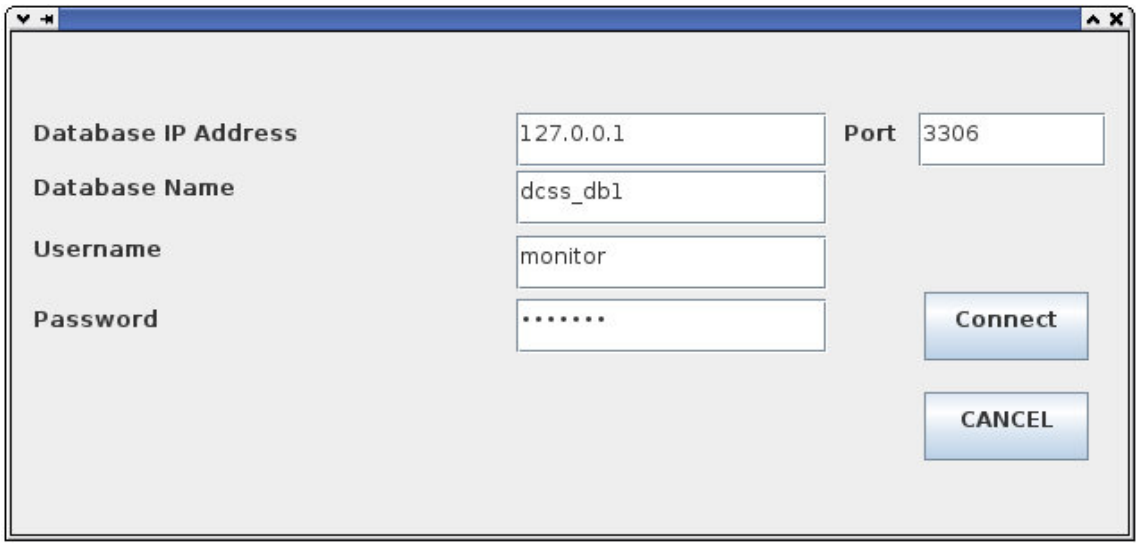
Rys. nr. 17 Read/Write operations for system paging



Rys. nr. 18 Read/Write operations for spooling

4.2.2 Kwerendy baz danych w JDBC

Program zpm zaprojektowany został z myślą o strukturze danych pochodzących z daemon'a zpmd. Do uzyskania połączenia używany jest Java Database Connector.



Rys. nr. 19 Menu do łączenia z bazą danych

Powyższe argumenty wpisywane w polach jTextField umieszczane są w zmiennych globalnych dostępnych z poziomu każdej klasy. Używane są przy nawiązaniu pierwszego (głównego) połączenia oraz przy każdym kolejnym połączeniu nawiązywanym przez kolejne wątki.

Klasa nawiązująca główne połączenie wygląda następująco:

```
public class JDBCinit {

public JDBCinit() throws SQLException, ClassNotFoundException, InstantiationException, IllegalAccessException{

   Class.forName("com.mysql.jdbc.Driver").newInstance();

   ZpmApp.conn = (Connection) DriverManager.getConnection("jdbc:mysql://"

         + ZpmApp.ip + ":" + ZpmApp.port + "/" + ZpmApp.dbname, ZpmApp.user, ZpmApp.passwd);



   ZpmApp.s = (Statement) ZpmApp.conn.createStatement();

}

public JDBCinit(Connection conn, boolean m) throws SQLException {

   if (m == true) {

      conn.close();

      ZpmView.eDBConn.setEnabled(false);

      ZpmView.sDBConn.setEnabled(true);

   }
```

```
  }
}
```

Przeciążono w niej konstruktor, gdzie wywołując pierwszy (bez parametrów) nawiązuje się połączenie z baza MySQL. Natomiast drugi konstruktor (z dwoma parametrami) powoduje zamknięcie głównego połączenia wraz ze zmianą na przeciwną przycisków odpowiadających za połączenie i rozłącznie programu z bazą.

Dla kolejnych połączeń kod źródłowy wygląda, jak poniżej:

- Stworzenie wątku:

```
new CurrentEvents();
```

- Zatrzymanie wątku:

```
CurrentEvents.stop();
 Thread.sleep(1000); //w8 1 second to let CurrentEvents thread stopped
```

- Wybrane fragmenty klasy CurrentEvents:

```
public class CurrentEvents implements Runnable {
private static boolean thread_is_on;
 /**
 * Define new thread
 */
 Thread t_cevent;

 /**
 *Define statements for new DB connection
 */
 Connection conn = null;
 Statement s = null;
 ResultSet rs = null;

/**
 * Creates new thread & goes to function run()
```

```
 */
public CurrentEvents() {
    t_cevent = new Thread(this, "CurrentEvents");
    t_cevent.start();
}

/**
* Killing active thread
*/
public static void stop() {
    thread_is_on = false;
}

/**
* Here starts new Thread (run())
*/
public void run() {
    try {
        try {
            Class.forName("com.mysql.jdbc.Driver").newInstance();
        } catch (InstantiationException ex) {
            Logger.getLogger(CurrentEvents.class.getName()).log(Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex) {
            Logger.getLogger(CurrentEvents.class.getName()).log(Level.SEVERE, null, ex);
        }
    } catch (ClassNotFoundException ex) {
        Logger.getLogger(CurrentEvents.class.getName()).log(Level.SEVERE, null, ex);
    }

    try {
   conn = DriverManager.getConnection("jdbc:mysql://" + ZpmApp.ip + ":" + ZpmApp.port + "/"
```

```
  +ZpmApp.dbname, ZpmApp.user, ZpmApp.passwd);
   } catch (SQLException ex) {
      Logger.getLogger(CurrentEvents.class.getName()).log(Level.SEVERE, null, ex);
   }
   try {
      s = conn.createStatement();
   } catch (SQLException ex) {
      Logger.getLogger(CurrentEvents.class.getName()).log(Level.SEVERE, null, ex);
   }
conn.close();
}
```

Powyższy fragment kodu źródłowego umieszczony został łącznie z komentarzami. Tak więc uznano za zbyteczne dodatkowe opisywanie kodu.

Kwerendy wysyłane do bazy danych umieszczane są w funkcji executeQuery(). Poniżej zostały umieszczone wybrane przykłady funkcji wysyłających i pobierających informacje z bazy danych:

```
public int StartingValueSelectSQL(String rec) throws SQLException {
   int x = 0;
   rs = s.executeQuery("SELECT id FROM ("+rec+") ORDER BY id DESC LIMIT 1");
   while(rs.next()) {
   rs.last();
   x = rs.getInt(1);
   }
   rs.close();
   return x;
}
private void Query_d2r3() throws SQLException {
rs = s.executeQuery("SELECT * FROM d2r3 WHERE id > ("+id2r3+")");
while (rs.next()) {
            id2r3 = rs.getInt("id");
            String cell2 = rs.getString("time");
```

```
            String cell3 = rs.getString("sclwrr_vmduser");
            String cell4 = "NOT TERMINAL";
            if ((rs.getInt("sclwrr_calflags") & 128) == 128) {
              cell4 = rs.getString("sclwrr_rdevsid");
            }
            ZpmView.jTextArea1.append(cell2 + " |   " + "Console WRITE has completed for User( "
            + cell3 + " )  " + "Real device subchannel id( " + cell4 + " ) \n");
                }
rs.close();
}
```

5 Podsumowanie

Celem pracy było stworzenie środowiska monitorującego maszyny wirtualne w systemie operacyjnym z/VM. W środowisku tym zaimplementowano autorskie rozwiązania bazujące na licencji GPLv2, takie jak programy zpmd oraz zpm kooperujące z pozostałymi elementami Open Source, do których należą system operacyjny z/Linux uruchomiony pod kontrolą systemu z/VM, baza danych MySQL, biblioteki libmysqlclient15-dev, JFreeChart oraz JCommon.

Omówiona została architektura systemów z serii „z". Przeprowadzono proces konfiguracji systemu z/VM wraz z instalacją systemu z/Linux. By w tak przygotowanym środowisku zaprezentować działania autorskich programów zpmd oraz zpm. W kolejnym etapie rozwoju program z/VM Performance Daemon może zostać uzupełniony o obsługę dodatkowych rekordów oraz przejść proces optymalizacji kodu. Natomiast dalszy rozwój aplikacji klienckiej z/VM Performance Monitor powinien się skupić na szerszej obsłudze rekordów znajdujących się w bazie danych, logicznym przedstawieniu ich w aplikacji za pomocą wykresów i tekstu.

Reasumując do dalszego rozwijania w/w programów potrzebne jest środowisko testowe generujące wszystkie rekordy oznaczone, jako niestabilne i nieobsługiwane do tej pory przez program zpmd. Ostatecznie obie aplikacje zostaną umieszczone na stronie internetowej sourceforge.net jako osobno rozwijane projekty.

6 Streszczenie

6.1 Streszczenie

Wstęp prezentuje główny motyw pracy inżynierskiej wraz z krótkim przedstawieniem głównych rozdziałów znajdujących się w tejże pracy.

Pierwszy rozdział zawiera informacje na temat architektury zSeries. Skupia się na Cental Processing Complex. Podzielony jest on na podrozdziały zawierające informacje o typach procesorów, głównej pamięci, logicznych partycjach i urządzeniach I/O.

Drugi rozdział przedstawia konfigurację systemu operacyjnego z/VM, zaczynając od zdefiniowania wirtualnej maszyny, stosu TCP/IP, aż po konfigurację usługi *MONITOR i Discontiguous Saved Segment'u.

Trzeci rozdział mówi o instalacji i konfiguracji systemu operacyjnego z/Linux uruchamianego pod kontrolą systemu z/VM. Zawiera także listę dodatkowych komponentów wymagających doinstalowania.

Czwarty rozdział jest poświęcony dwóm autorskim programom: z/VM Performance Monitor Daemon oraz z/VM Performance Monitor. Zawiera ich funkcjonalność i opis techniczny.

Piąty rozdział przedstawia podsumowanie pracy.

Szósty rozdział zawiera streszczenie w języku polskim i angielskim.

Siódmy rozdział zawiera słowa kluczowe dla tej pracy.

Ósmy rozdział zawiera bibliografię.

Dziewiąty rozdział przedstawia spis wszystkich tabel.

6.2 Abstract

Introduction presents the base field of graduate work. Morover it discusses chapters

in a short form.

First chapter is about Central Processing Complex and consist useful information about zSeries architecture which includes types of processor units, main storage, logical partitions and last but not least I/O devices.

Second chapter consists configuration of z/VM operating system as from definition of virtual machine for z/Linux guest through TCP/IP Stack and system service called *MONITOR to finish on Discontiguous Saved Segment structure and functionality.

Third chapter describes installation and configuration process of z/Linux operating system on z/VM. However it contains a list of additional components to install with short description.

Fourth chapter is dedicated to author's programs: The z/VM Performance Monitor Daemon and the z/VM Performance Monitor. It describes their technical reference and functionality.

Fifth chapter presents Summary of this graduate work

Sixth chapter includes abstract of this graduate work.

Seventh chapter includes graduate work's keywords.

Eighth chapter includes bibliography.

Ninth chapter includes list of all tables in this graduate work.

7 Słowa kluczowe

Słowa kluczowe w języku polskim:

- z/VM
- z/Linux
- DCSS
- zpmd
- zpm

Słowa kluczowe w języku angielskim:

- z/VM
- z/Linux
- DCSS
- zpmd
- zpm

8 Bibliografia

[1] Stephen Prata, *Język C*, Wrocław: „Robomatic”, 1994

[2] Bruce Eckel, *Thinking in Java*, Gliwice: „Helion”, 2006

[3] Leon Atkinson, *Core MySQL*, Gliwice: „Helion”, 2003

[4] IBM, Enterprise *Systems Architecture/s390 Principles of Operation,* SA22-7201-07, 2003

[5] IBM, *CP Commands and Utilities Reference*, SC24-6081-05, 1991

[6] IBM, *Device Drivers, Features, and Commands*, SC33-8411-00, 2008

[7] IBM, *Monitor Record formats for z/VM V5R3.0*, http://www.vm.ibm.com/pubs/mon530/index.html, stan na dzień: 29.01.2009

[8] IBM, *Introduction to the New Mainframe: z/OS Basics*, SG24-6366-00, 2006

[9] IBM, *ABCs of z/OS System Programming Volume 10*, SG24-6990-01, 2006

9 Spis tabel

10 Spis rysunków