VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

Monitorování linuxového clusteru s využitím JNA

LINUX CLUSTER MONITORING USING JNA

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE TOMÁŠ FURCH
AUTHOR

VEDOUCÍ PRÁCE ING. RADEK KOČÍ, Ph.D.
SUPERVISOR

BRNO 2017

Abstrakt

Tato práce se zabývá monitorováním linuxového clusteru. Jejím cílem je implantace nástroje sloužícího k detekci nepovoleného souběhu definovaných procesů. V první části je práce je popsán teorie procesů linuxových operačních systému a popisu využitých technologií. V další části je pak detailně popsána samotná implantace nástroje.

Abstract

This thesis deals with linux cluster monitoring. It´s main goal is implementation tool, which can be use to detectection confluence of defined processes. The first part focuses on theory about linux operation system processes and description of used technologies. The second part provides detailed description of tool implementation.

Klíčová slova

Linux, java, JNA, cluster, monitorování, procesy

Keywords

Linux, Java, JNA, cluster, monitoring, processes

Citace

Furch Tomáš: Monitorování linuxového clusteru s využitím JNA, bakalářská práce, ¨

FIT VUT, Brno 2017

Monitorování linuxového clusteru s využitím JNA

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Radka Kočího, Ph.D.
Další informace mi poskytli Ing. Petr Adámek a Ing. Petr Bartusek.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Tomáš Furch
16.5.2017

Poděkování

Chtěl bych poděkovat Ing. Radku Kočímu, Ph.D. za vedení mojí bakalářské práce, Ing. Petrovi Adámkovi a Ing. Petrovi Bartuskovi za konzultace týkající se implementace nástroje a v neposlední řadě rodičům za jejich podporu během celé doby studia.

Obsah

1 Úvod

V dnešní době se čím dál častěji setkáváme s aplikacemi, které jsou nasazovány a provozovány na výpočetním clusteru z důvodu zvýšení výpočetní rychlosti nebo spolehlivosti běhu. S paralelizací programů roste složitost aplikací a jsou kladeny větší nároky na programátory. Zároveň i monitorování takových programů je často náročnější než u aplikací běžících na jednom stroji.

Cílem této práce je vytvořit nástroj pro monitorování linuxového clusteru s využitím technologie JNA. Hlavním úkolem nástroje bude sledovat vznik a zánik procesů na jednotlivých strojích, jež jsou součástí clusteru a sledovat, zda nedochází k souběhu nepovolených procesů.

Tato práce vznikla ve spolupráci firmou Home Credit International a.s. (HCI), respektive její softwarovou divizí EmbedIT. HCI používá v některých zemích své působnosti otisky prstů jako ochranu před podvody. A právě v rámci komponenty pracující s otisky prstů došlo k nutnosti sledovat, zda nedošlo k souběhu určitých procesů a o případných problém uvědomit monitorovací oddělení.

Text práce je rozdělen do několika kapitol. V první jsou uvedeny teoretické informace o proces se za měřením na operační systémy UNIXového typu a dále přehled technologií použitých při vývoji monitorovacího nástroje. Druhá kapitola se věnuje vlastní implementaci nástroje, ve třetí kapitole jsou nastíněny další možnosti rozvoje nástroje.

2 Teoretická část

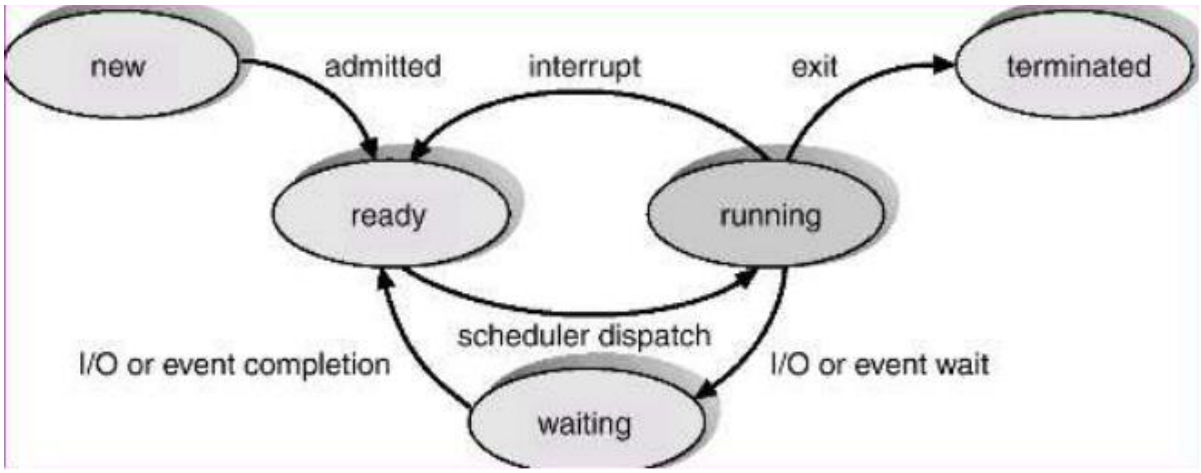
2.1 Procesy

Proces je instance běžícího programu, ale program může sestávat z mnoha procesů. Pro zodpovězení otázky jaký je vztah mezi programem a procesem/procesy je tedy nutné odpovědět na otázku co je to program.

Program je soubor obsahující řadu informací popisujících jak vytvořit proces (instanci) tohoto programu. Tyto informace obsahují: binární identifikační formát, instrukce strojového jazyka, vstupní adresu programu, data, tabulky symbolů a relokační tabulky, sdílené knihovny a dynamicky linkované informace a další informace důležité pro vytvoření procesu [1].

Abychom mohli detekovat souběh námi definovaných procesů, musíme porozumět životnímu cyklu procesů – jak proces vzniká, zaniká a jaké stavy může nabývat. Životní cyklus procesu je řízen jádrem operačního systému, nás budou zajímat především operační systémy UNIXového typu. Tyto operační systémy rozlišují několik stavů procesů [2]:

- **New** (nový) – proces byl vytvořen
- **Running** (běžící) – instrukce procesu jsou vykonávány
- **Waiting** (čekající) – proces čeká na nějakou událost – dokončení vstupně/výstupní operace nebo přijetí přijatí signálu
- **Ready** (připraven) – proces čeká na přidělení procesoru
- **Terminated** (ukončený) -proces ukončil vykonávání



Obrázek 1 Diagram stavů procesu

Pro potřeby monitorovacího nástroje jsou důležité informace, kdy proces vznikl a kdy zanikl. To zajišťují volání systémových funkcí, z kterých nás budou zajímat především tři, *fork(), exec() a exit()*. [1]

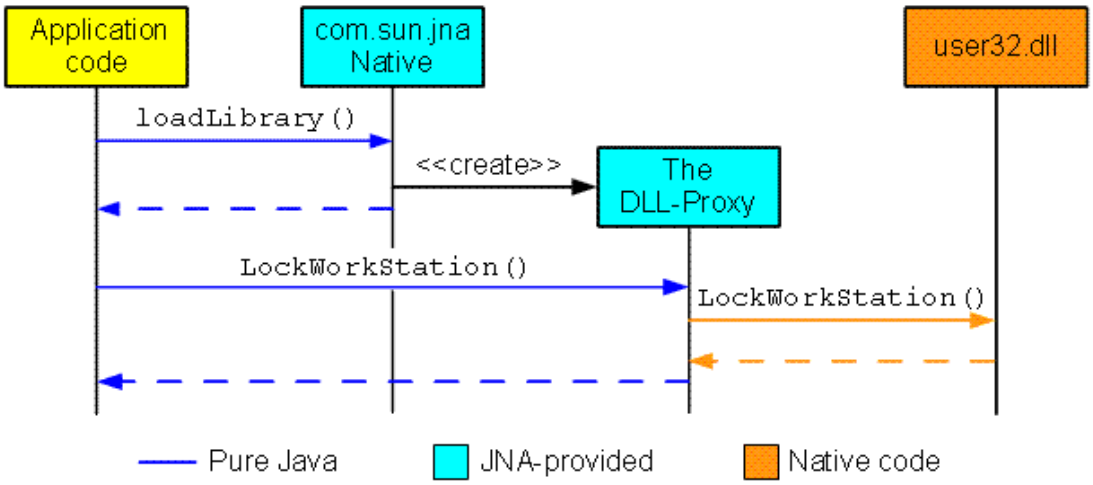
- **Fork()** – volání *fork()* umožnuje jednomu procesu (rodiči) vytvořit nový proces, potomka. Takto vzniklý proces je téměř stejný se svým rodičem, obsahuje kopii zásobníku, dat, heapu (hromady) a dalších částí rodičovského procesu
- **Exec()** – vytvoření nového procesu v paměťovém prostoru procesů. Existující program je zahozen a zásobník, data a heap jsou znovu vytvořeny pro nový program.
- **Exit()** – toto volání ukončí proces, uvolní všechny zdroje (paměť, otevřené popisovače souborů a další) používané procesem a umožní jádru jejich další alokaci pro jiné procesy.

K rozlišení jednotlivých procesů slouží identifikátor procesu (proces ID, PID), jedná se unikátní kladné celé číslo, které je menší než konstanta jádra PID_MAX, která je výchozím nastavení 32 767, což je maximum na 32-bitových systémech. Na 64-bitových systémech je možné tuto konstantu nastavit až do maxima 2^{22} což je více než 4 miliony unikátních hodnot.

Další z identifikátorů procesu je identifikátor procesu rodiče. Každý proces (kromě procesu init s PID 1) má rodiče – proces, který ho vytvořil a na tento proces odkazuje PPID (parent proces ID). Pokud dojdu k ukončení rodičovského procesu je PPID potomků nastaveno na 1, novým rodičem těchto procesů se tak stává proces *init*. Atribut PPID procesu slouží k vytvoření stromu závislostí procesů, kde každý rodič má svého rodiče a takto se postupuje stromem závislostí až k procesu 1 [1].

2.2 Java native access

Java Native Access (JNA) je technologie usnadňující přístup k nativním knihovnám z programu v jazyce Java a to bez nutnosti generování „slepovacího“ kódu a je tak přímočařejší a přehlednější než Java Native Interface (JNI), která je součástí Java API.



Obrázek 2 JNA architektura

Toto zjednodušení je ovšem vykoupeno vyšší výpočetní náročností než je tomu u JNI a je tedy nutno zvážit do jaké míry je výkon aplikace využívající nativní kód důležitý.

Při volání nativních knihoven je nutné řešit několik problémů, především jak mapovat knihovny, funkce a datové typy. Mapování knihoven a názvů funkcí je velice přímočaré:

```
// Knihovny - možnost 1: Využití rozhraní, dynamické
načítání C knihoven
public interface CLibrary extends Library {
 CLibrary INSTANCE =
     (CLibrary)Native.loadLibrary("c", CLibrary.class);
}

// Knihovny - možnost 2: přímé mapování
public class CLibrary {
   static {
       Native.register("c");
    }
}
```

Jména funkcí jsou mapovány přímo z jejich Java rozhraní na názvy z nativních knihoven (možnost 1), popřípadě na deklarované nativní funkce (možnost 2). JNA nabízí možnost přizpůsobit názvy metod Java konvencím s využitím `Library.OPTION_FUNCTION_MAPPER/FunctionMapper`, kde jsou namapovány názvy nativních funkcí na programátorem zvolené názvy odpovídající zvyklostem jazyka.

```
// Funkce – možnost 1:
public interface CLibrary extends Library {
    int atol(String s);
}

// Funkce – možnost 2:
public class CLibrary {
    public static native int atol(String s);
}
```

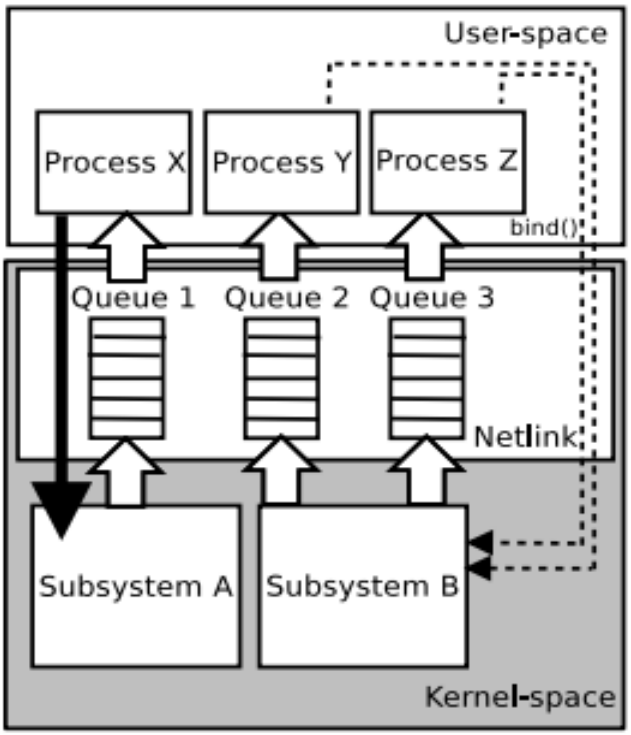
Složitější situace nastává u datových typů, kdy musíme volit typy odpovídající si velikostí. V následující tabulce jsou typy podporovaní knihovnou JNA:

Nativní typ	**Velikost**	**Java**
char	8 bitů	byte
short	16 bitů	short
wchar_t	16/32 bitů	char
int	32 bitů	int
int	Boolean	boolean
long	32/64 bitů	NativeLong
long long	64 bitů	long
float	32 bitů	float
double	64 bitů	double
char*	C string	String
void*	Pointer	Pointer

2.3 Netlink sockets

Netlink sockets je datagramově orientovaná technologie umožňující komunikaci mezi procesy jádra a uživatelskými procesy a naopak [10][11]. Stejně tak může být také využita jako mezi procesorový komunikační systém.

Netlink sokety jsou založeny na implementaci BSD soketů a podporují tak běžné funkce *socket(), bind(), sendmsg()* a *recvmsg()*



Obrázek 3 Netlink sockety - unicast a multicast

Netlink sokety podporují dva typy komunikace:

Unicast – používá pro navázání komunikačního kanálu typu 1:1 mezi jádrem a jedním uživatelským procesem. Unicastové kanály se používají pro posílání příkazů jádru, přijímání výsledků těchto příkazů a vyžádání si informací od jádra operačního systému.

Multicast – vytváří komunikační kanály 1:N. Typicky vystupuje jádro jako odesílatel a N uživatelských procesů jako posluchači. Uživatelské procesy se tak mohou registrovat jako posluchači k jednomu nebo několika typům událostí.

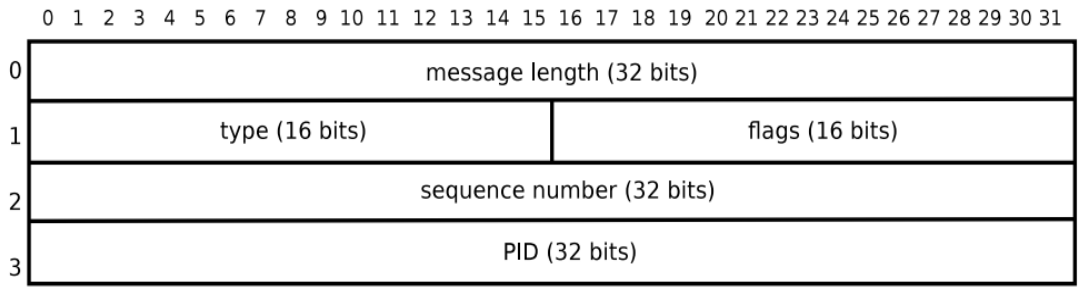
2.3.1 Formát Netlink zpráv

Netlink zprávy jsou zarovnávané na 32 bitů. Každá zpráva začíná hlavičkou fixní velikost 16 bytů, která je tvořena následujícími položkami:

- Délka zprávy (32 bitů): délka zprávy v bytech, včetně hlavičky
- Typ zprávy (16 bitů): zprávy mohou být buď datové, nebo řídící. Datové zprávy závisí na tom, co je povoleno subsystémem jádra. Řídící zprávy jsou společné pro všechny subsystémy.
- Příznaky (16 bitů): v hlavičce zprávy mohou být nastaveny následující příznaky
    - NLM_F_REQUEST – v případě nastavení obsahuje zpráva požadavek
    - NLM_F_CREATE – uživatelský proces posílá subsystému jádra příkaz nebo přidat novou konfiguraci
    - NLM_F_EXCL – používá ve spojení s CREATE pro zachycení chyby, kdy se zpráva snaží přidat konfiguraci, která již v subsystému jádra existuje.
    - NLM_F_REPLACE – změna existující konfigurace v subsystému jádra
    - NLM_F_APPEND – přidává konfiguraci k již existující
    - NLM_F_DUMP – uživatelská aplikace si vyžádá úplnou synchronizaci se subsystémem jádra
    - NLM_F_MULTI – zpráva s více částmi, kterou odpovídá subsystém v případě, že přijat zprávu s nastaveným DUMP příznakem.
    - NLM_F_ACK – uživatelská aplikace vyžaduje potvrzující zprávu o správném vykonání požadované operace. V případě, že tento příznak není nastaven, subsystém informuje o chybě prostřednictvím *sendmsg()* s ernno hodnotou.
    - NLM_F_ECHO – aplikace vyžaduje, aby příchozí zprávy byly posílány prostřednictvím unicastového spojení i v případě, že je aplikace připojena přes multicastový kanál.
- Sekvenční číslo (32 bitů): využívá se při posílání zpráv s nastaveným ACK příznakem, kdy potvrzující zpráva od subsystému obsahuje v hlavičce

stejné sekvenční číslo a uživatelská aplikace může párovat odpovídající si požadavky s odpověďmi.

- Port ID (32 bitů): obsahuje číselný identifikátor přiřazený Netlink subsystémem. Tento identifikátor je používán k rozlišení několika kanálů otevřených stejným uživatelským procesem.

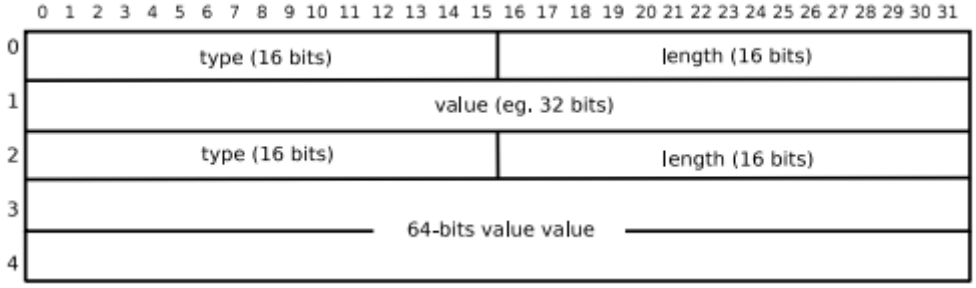


Obrázek 4 Hlavička netlink zprávy

Obsah zprávy je tvořen množinou atributů Type-Length-Value (TLV) formátu, který je používán v mnoha protokolech včetně IPv4 a IPv6. TLV formát:

- Typ (16 bitů): typ atributu podle, záleží na množině typů poskytovaných jádrem operačního systému. Dva nejvýznamnější bity určují, zda se jedná o vnořený atribut (bit 0), umožnuje posílat více atributů jako jeden a zda je obsah řazený podle „network byte order“ (bit 1). Zbývajících 14 bitů slouží k identifikaci typů a je tak možné rozlišit až 2^{14} (16 384) rozdílných typů.
- Délka (16 bitů): délka atributů v bytech, včetně hlavičky, ale bez zarovnání obsahu na 32 bitů.
- Hodnota: vlastní obsah atributu proměnné délky, vždy se zarovnává na 32 bitů

Další obrázek znázorňuje obsah zprávy se dvěma atributy, první s hodnotou délky 32 bitů a druhý atribut s 64 bitovou hodnotou.



Obrázek 5 Netlink zpráva se dvěma atributy

2.4 Java Management Extensions

Java Management extensions (JMX) definuje architekturu, návrhové vzory, API a služby pro aplikace a síťovou zprávu a monitoring v programovacím jazyce Java. Tato technologie poskytuje široké možnosti při sledování virtuálního stroje Javy (JVM) a jeho ovládání. Některé funkce jsou poskytovány již v rámci základní implementace JVM (například počet vláken, počet tříd načtených do paměti virtuálního stroje, možnosti spustit garbage collector a dalších) a o další funkce specifické pro programátorem vyvíjenou aplikaci je možné program rozšířit.

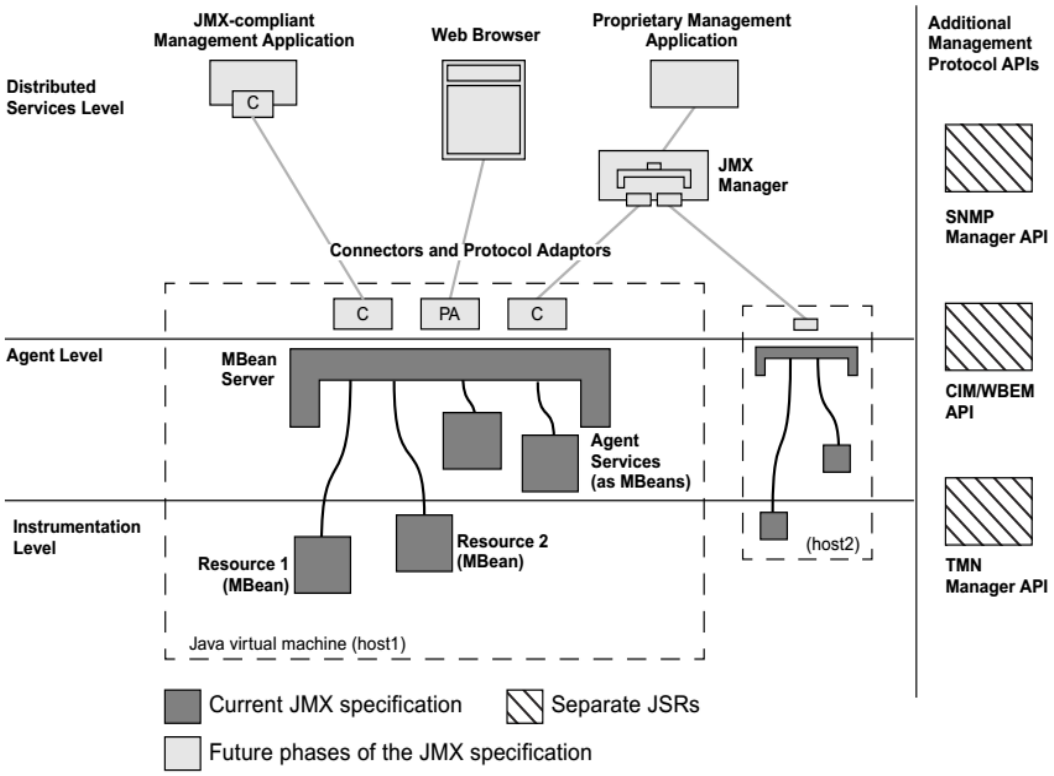
JMX bylo původně vyvíjeno v rámci Java Community Process (JSR) a to ve verzích 1.0, 1.1 a 1.2 jako JSR 003 [3], ve verzi 2.0 jako JSR 255 [4], tento vývoj byl ovšem zastaven a dnes se nevyužívá a podle JSR 160 [5] bylo vyvíjeno ve verzi 1.0 API pro vzdálenou správu a monitorování.

Od verze Java 5 SE je JMX součástí platformy Java a k jeho využívání tak nejsou potřeba další externí knihovny.

2.4.1 Architektura JMX

JMX je postaveno na třívrstvé architektuře, což umožňuje do značné míry nezávislou implementaci jednotlivých vrstev bez nutnosti zásahu do vrstev ostatních a je tak možné například využívat již existujících nástrojů pro tuto technologii. Jednotlivé vrstvy tvořící JMX jsou:

- **Instrumentation level**
- **Agent level**
- **Distributed services level**



Obrázek 6 JMX komponenty a jejich vztah

2.4.1.1 Instrumentation level

Instrumentální vrstva poskytuje specifikaci pro implementaci zdrojů spravovatelných pomocí JMX. Mezi tyto zdroje patří aplikace, služby, zařízení, uživatelé a další. V rámci této vrstvy se vyskytuje výkonný kód JMX části aplikace.

Managed MBeans (MBeans)

MBean je Java objekt implementující specifické rozhraní. Toto rozhraní monitorovaného zdroje obsahuje všechny nezbytné informace, které jsou potřeba pro operaci se zdrojem:

Hodnotové atributy, ke kterým je možno přistupovat

Operace, které mohou být nad zdrojem vyvolány
Notifikace, které mohou být vysílány
Konstruktory pro třídy implementující toto rozhraní

Podle způsobu jakým jsou vytvářena tato rozhraní, rozlišujeme čtyři typy MBean.

Standard MBeans – nejjednodušší typ MBean, jejich je popsáno jmény jejich veřejných metod

Dynamic MBeans – musí implementovat specifické rozhraní, ale rozhraní, které vystavují externím aplikacím pro zprávu, je možné za běhu měnit

Open MBeans – dynamické MBeany s jednodušší implementací založené na základních datových typech

Model MBeans – jde taktéž od typ dynamických MBean, které jsou plně konfigurovatelné za běhu a poskytují generickou třídu MBeanů s defaultním chováním

2.4.1.2 Agent level

Tato vrstva poskytuje specifikaci pro implementaci agentů, kteří přímo řídí zdroje a zpřístupňují je vzdáleným řídícím aplikacím. Agenti jsou obvykle spuštěny na stejných strojích jako zdroje, které kontrolují, i když to není nezbytné.
Tato vrstva leží nad instrumentální vrstvou a jejím základem je MBean server a množina služeb operujících s MBeanami. JMX agent musí navíc obsahovat alespoň jeden komunikační adaptér nebo konektor.

MBean server – registruje objekty (MBeany) a tyto objekty se stávají viditelné pro aplikace sloužících pro správu. MBean server vystavuje pouze rozhraní těchto objektů, nikdy ne objekty samotné. Jakékoliv zdroje, které mají být spravovatelné z vnějšku JVM musí být registrované v nějakém MBean serveru. MBean server také poskytuje

standardizované rozhraní pro přístup k MBeanům uvnitř stejného virtuálního stroje, na kterém běží agent.
Jednotlivé MBeany mohou být registrované jinou MBeanou, agentem samotným nebo vzdálenou aplikací. MBeany jsou registrovány unikátním jménem, které používáno spravují aplikací k vykonávání operací na dané MBeně. Mezi tyto operace patří:

- Čtení a zápis hodnot jejich atributů
- Vykonávání operací poskytnutých MBeanou
- Přijímání notifikací od MBean

Služby agenta - jsou objekty, které mohou vykonávat operace na MBeanách registrovaných MBean serverem. Specifikace JMX definuje následující služby:

Dynamické načítání tříd – umožňuje dynamické načítání tříd ze vzdálených zdrojů
Monitory - sledují hodnoty atributů (číselných nebo řetězců) a mohou ostatní objekty informovat o těchto změnách.
Časovače – poskytují plánovací mechanismus, může jít buď o jednorázové spuštění, nebo periodickou činnost
Související služby – definují vztahy mezi MBeanami a starají se o jejich dodržování

2.4.1.3 Notifikační model

JMX specifikace definuje generický notifikační model založený na Java událostech. Notifikace mohou být emitovány instancemi MBean nebo MBean serverem. Klient implementující specifické rozhraní se můžu k MBean serveru přihlásit jako posluchač zpráv a poté dostávat zprávy, které jsou tímto serverem vydávány.

3 Implementační část

Úkolem této práce je implementovat nástroj, který umožňuje detekovat, zda nedošlo k souběhu nepovolených procesů při běhu ExpressID AFIS 3 clusteru. Jedná se o nástroj, který složí k porovnávání a identifikaci pomocí otisků prstů. Porovnávání otisků prstů není možno indexovat, proto je proces verifikace otisků prstů výpočetně náročný. Při verifikaci dochází k porovnání snímaného otisku se všemi otisky, které jsou v databázi (na rozdíl od identifikace, kdy dochází k porovnání pouze dvou otisků). Pokud navíc budeme předpokládat, že nevíme otisk kterého prstu zrovna porovnáváme, musíme porovnat každý otisk se všemi ostatními, které máme k dispozici. Za předpokladu, že máme databázi čítající 100 000 osob a pro verifikaci používáme vždy všech deset prstů, dostáváme deset milionů porovnání (100 000 osob v databázi po 10 prstech x 10 nově pořízených otisků). I při této relativně malé databázi osob dostáváme velké množství porovnávání a při požadavku na co nejrychlejší verifikaci osoby musíme hledat možnosti jak tento čas zkrátit. Jedním z možných způsobů jak toho docílit je rozložit výpočty na více strojů. Toto řešení (nejenom, dalším je například použitím „in memory“ databáze) bylo využito u ExpressID AFIS 3, který může být nasazen na výpočetní cluster. Pro řízení a běh clustru je využito několik typů procesů s rozdílnou funkcionalitou. Konkrétně jde například o procesy „Matcher“ a „Dispatcher“, kde proces typu matcher běží na každém nodu clusteru a je to proces, který provádí samotné porovnávání. Proces dispatcher, jak už z názvu vyplývá, se stará o rozdělování zátěže napříč jednotlivé nody a je tedy žádoucí, aby tento proces běžel právě na jenom z nodů. Při běhu ExpressID ovšem docházelo k situaci, kdy běželo více procesů typu dispatcher souběžně (na jednom z nodů se proces neukončil korektně, ale zároveň došlo k nastartováni dispatcher procesu na jiném nodu). Právě tyto situace vedli k nutnosti implementace nástroje, který by umožňovat detekovat, že k takovému souběhu došlo.Nad ExpressID AFIS serverem běží Fingerprint server (FP server). Tento server je vyvíjen ve společnosti Home Credit International a.s. a jeho hlavním úkolem je sběr otisků prstů klientů a komunikace s ExpressID serverem, kde dochází k vyhodnocení. Monitorovací nástroj pro detekci souběhu procesů je integrován do řešení FP serveru, kde je součástí rozsáhlejšího monitorovacího rozhraní.

3.1 Základní architektura

Pro úspěšnou detekci musí nástroj monitorovat vznik a zánik procesů na každém stroji clusteru.

Pro tento účel je využito zachytávání meziprocesové komunikace jádra operačního systému a to především o vzniku nového procesu (operace fork() a exec()) a jeho zániku (operace exit()). Při odchycení některé z výše uvedených operací nad definovaným procesem dojde k uložení/aktualizaci záznamu v databázi.

V pravidelných intervalech dochází ke sběru informací z jednotlivých nodů a nad těmito daty dochází k ověření, zda v definovaném intervalu došlo k souběhu či nikoliv. Tento stav je následně zobrazován na dashboardu FP serveru a zároveň je poskytovaná informace ve formátu JSON pro centrální monitorovací systém (konkrétně NAGIOS).

3.2 Databáze

Základem pro běh nástroje je databáze. Pro běh nástroje v rámci FP serveru je dostačující pouze jedna tabulka s následující strukturou:

Process
id
process_name
host_name
pid
start_time
end_time
last_modification

id – jednoznačný identifikátor záznamu

process_name - jméno procesu

host_name – jméno stroje (nodu)

pid – identifikátor procesu

start_time – čas vzniku procesu

end_time – čas ukončení procesu

last_modification – čas kdy došlo k poslední modifikaci procesu

Při implementaci byla využita databáze *Apache Derby* [7][8]. Jedná se o relační databázi napsanou v jazyce java. Toto řešení bylo zvoleno z několika důvodů, mezi hlavní patří:

- nevyžaduje instalaci databázového stroje – databáze je vytvářena v rámci procesu monitorovacího nástroje
- nízká zátěž aplikace – celá její implementace včetně JDBC ovladače představuje nízké jednotky megabajtů
- podporuje standarty SQL-99 a SQL-2003

V rámci každého uzlu běží samostatná instance databáze a data z jednotlivých strojů jsou na vyžádání poskytovány FP serveru, který poté data z jednotlivých nodů sbírá a interpretuje.

Ačkoliv databáze obsahuje pouze jednu tabulku, tak postup času by docházelo k neúměrnému nárůstu velikosti, která by navíc neměla pro potřeby monitorování téměř žádný význam. Proto dochází k pravidelnému promazávání starých dat.

V případě implantace nástroje jako samostatně běžící aplikace je vhodné databázi doplnit o další tabulku, která slouží o udržování informací o jednotlivých nodech clusteru.

3.3 Start monitorovacího nástroje

3.3.1 Jak detekovat již běžící proces

První problém, se kterým je potřeba se vypořádat nastává v okamžiku spuštění. Pokud nebudeme chtít vynucovat závislosti v pořadí spouštění monitorovacího nástroje a

sledovaného ExpressID clusteru, musíme předpokládat, že námi sledovaný proces již běží. Pro řešení tohoto problému bychom mohli využít nějakého externího nástroje, nabízí se například použití standardního UNIXového příkazu ps a poté parsovat jeho výstup:

```
ubuntu_abhi@ubuntu-Dell: ~
File  Edit  View  Search  Terminal  Help
1000      12628  3294  0 Jul03 ?        00:00:00 [chrome] <defunct>
1000      13126     1  0 Jul01 ?        00:00:00 /usr/lib/d-conf/dconf-service
1000      18460  3294  0 Jul12 ?        00:00:00 [chrome] <defunct>
1000      20128  3294  0 Jul11 ?        00:00:00 [chrome] <defunct>
www-data  20264  1396  0 00:35 ?        00:00:00 /usr/sbin/apache2 -k start
www-data  20265  1396  0 00:35 ?        00:00:00 /usr/sbin/apache2 -k start
www-data  20266  1396  0 00:35 ?        00:00:00 /usr/sbin/apache2 -k start
www-data  20267  1396  0 00:35 ?        00:00:00 /usr/sbin/apache2 -k start
www-data  20268  1396  0 00:35 ?        00:00:00 /usr/sbin/apache2 -k start
1000      20370  3294  0 Jul09 ?        00:00:00 [chrome] <defunct>
1000      21705  3294  0 Jul14 ?        00:00:00 [chrome] <defunct>
1000      22283     1  0 Jul07 ?        00:00:00 /usr/lib/gvfs/gvfsd-http --spawn
root      22291     1  0 Jul07 ?        00:00:00 /usr/bin/python /usr/share/apt-x
1000      22904  3294  0 Jul08 ?        00:00:00 [chrome] <defunct>
1000      23235     1  0 Jul07 ?        00:00:15 /usr/bin/python /usr/bin/zeitgei
1000      23240 23235  0 Jul07 ?        00:00:00 /bin/cat
1000      23242 23235  0 Jul07 ?        00:00:12 /usr/bin/python /usr/bin/zeitgei
1000      25011  3294  0 Jul07 ?        00:00:00 [chrome] <defunct>
1000      26209  3294  0 Jul02 ?        00:00:00 [chrome] <defunct>
1000      29794  3294  0 Jul06 ?        00:00:00 [chrome] <defunct>
1000      31914  3294  0 Jul13 ?        00:00:00 [chrome] <defunct>
ubuntu_abhi@ubuntu-Dell:~$
```

Obrázek 7 příklad výstupu příkazu ps

Tento postup má několik nevýhod. Jednou z nich je závislost na použitém nástroji (zde právě ps).

Dalším problémem je fakt, že nás kromě toho zda proces běží bude zajímat také čas, kdy došlo ke spuštění procesu. Z ukázkového výstupu je vidět, že orientační čas jsme schopni zjistit, ale pokud nás zajímá čas s vyšší přesností než sekundy, pak už jsme limitováni.

Pro řešení tohoto problému bylo využito proc filesystemu (procfs). Procfs je virtuální filesystem, který využívají operační systémy unixového typu pro ukládání informací o běžících procesech. Procfs je umístěn v operační paměti počítače a nedochází k jeho ukládání mezi jednotlivými spuštěními systému. Má adresářovou strukturu, kde každý proces je reprezentován cestou /proc/PID, kde PID identifikuje běžící proces.

```
> cat /proc/stat
cpu  2255 34 2290 22625563 6290 127 456
cpu0 1132 34 1441 11311718 3675 127 438
cpu1 1123 0 849 11313845 2614 0 18
intr 114930548 113199788 3 0 5 263 0 4 [... lots more numbers ...]
ctxt 1990473
btime 1062191376
processes 2915
procs_running 1
procs_blocked 0
```

Obrázek 8 Soubor proc/stat

Jak tedy z proc filesystemu zjistit, zda běží proces, který nás zajímá? V adresářové struktuře procfs jsou kromě adresářů pro každý proces, obsaženy i další adresáře a soubory, ve kterých jsou informace o běžícím systému (např. verze operačního systému, využití paměti, procesu, zdrojů a spousta dalších informací). Většinu těchto informací nevyužijeme, s jednou jedinou výjimkou. V souboru /proc/stat je uložen čas spuštění aktuálně běžícího systému, řádek začínající textem „btime“ (boot time) je následován číslem reprezentujícím čas spuštění. Btime je udáván v sekundách od 1.1.1970 (začátek UNIXové epochy). Tento čas využijeme pro určení času startu konkrétních procesů. Dále nás už zajímají pouze adresáře konkrétních procesů, tedy takové adresáře jejich jména vyhovují regulárnímu výrazu „^[0-9][0-9]*[0-9]$“ (tedy PID jednotlivých procesů). Abychom zjistili, zda běží sledovaný proces, musíme projít všechny takovéto adresáře a zjistit o jaké procesy se jedná. K tomu využijeme jednak jména procesu jehož souběh chceme detekovat, v tomto konkrétním případě je to „IDispatcher“ a souboru /proc/[PID]/stat. Soubor stat obsahuje pole 52 řetězců [6]. Z těchto hodnot nás zajímají tři údaje, na pozici jedna (číslováno od 1) je PID procesu, následuje jméno procesu a na pozici 22 je čas startu procesu udávaný v počtu tiků CLK od startu systéme (od verze

jádrra 2.6, před touto verzí se používaly tzv. „jiffies“). Počet tiků můžeme zjistit vypsáním *sysconf(_SC_CLK_TCK)*, na většině dnešních systému je tato hodnota nastavena na 100. Nyní již máme všechny informace, které potřebuje pro jištění, zda už nějaký „problematický“ proces běží a také pro určení jeho startu. Projdeme tedy celou adresářovou strukturu, najdeme všechny procesy se jménem „IDispatcher“, z údajů btime, počtu tiků a času startu procesu spočítáme čas spuštění procesu v milisekundách od 1.1.1970 a takto získané údaje, společně s PID uložíme do databáze.

Proc filesystem obsahuje už krátce po startu, kdy ještě nejsou spuštěny uživatelské procesy, nebo jen velmi málo těchto procesů, několik tisíc složek (převážně o procesy operačního systému), tento počet dále narůstá na serveru, kde běží mnoho spuštěných programů.

Projít celou strukturu procfs si vyžádá určitý čas a dochází k riziku, že během procházení běžících procesů vznikne proces „IDispatcher“, o jehož vzniku nezachytíme netlink zprávu. Z tohoto důvodu je scanner běžících procesů implantován jako samostatně běžící vlákno.

3.4 Vznik a zánik procesů

Během běhu přijímá nástroj zprávy o vzniku, zániku a dalších událostech, které nad procesy spouští operační systém. K tomuto účelu využívá linuxové jádro Netlink zpráv a také speciálních typů socketů. Právě typ těchto socketů, které jsou platformě závislé si vynucuje použití technologie Java Native Access (Java API obsahuje implementaci síťových socketů, ne ovšem netlink socketů). Abychom mohli číst netlink zprávy využijeme dvou knihoven napsaných v jazyce C a to konkrétně *sys/socket.h* a *netlink.h*.

Jádro operačního systému posílá několik typů netlink zpráv, konkrétně se jedná o:

- **Fork**
- **Exec**
- **Coredump**

- **Uid**
- **Gid**
- **Sid**
- **Ptrace**
- **Comm**
- **Exit**

V rámci nástroje je implantováno odchytávání a mapování všech typů zpráv. Pro jeho běh jsou ovšem důležité tři typy, *fork*, *exec* a *exit*. Pokud dojde k zachycení jiného typu netlink socketu, pak dojde k jeho zahození a dále se nezkoumá jeho obsah.

3.5 Detekované procesy

Nyní již se umíme vypořádat s procesy, které běží v době spuštění monitorování a umíme detekovat vznik a zánik procesů, ke kterých došlo během běhu nástroje. Toto ovšem k úspěš-nému monitoringu nestačí. Musíme se umět vypořádat s nestandartním chováním, které může mít několik příčin. Jedna z nich je ukončení monitorovacího nástroje, a to jak řízenou (není nijak zaručeno pořadí ukončování procesů, monitorovací proces může být ukončen až po procesu monitoringu), tak neřízenou (chyba při běhu nástroje, například nedostatek paměti). Další nenadálou situací může být problém uzlu výpočetního clusteru – výpadek napájení, hardwarová chyba.

3.5.1 Nenadálé události

Jak řešit výše uvedené situace? Nabízí se možnost vyřešení při startu v rámci procházení procesů, které už na daném uzlu běží. Najdeme v databázi procesy, které nebyly řádně ukončeny, tedy procesy, které nemají nastaven *end_time* a pokusíme se čas ukončení takového procesu určit. Známe čas startu procesu (je uložen v databázi) a známe čas

nastartovaní systému (boot time). Pokud došlo ke startu takového procesu před nastartováním systému, pak jako čas ukončení nastavíme právě boot time aktuálně běžícího systému. Pokud proces vznikl za běhu aktuální instance (boot time předchází start time) je situace o něco složitější. V případě, že proces stále běží, pak jsme při procházení procesů detekovali proces se stejným PID a podobným časem startu (s nejvyšší pravděpodobností nezískáme dva identické časy, protože výpočet startu procesu a získání tohoto času z netlink zprávy pracuje s jinou přesností) a nemusíme nic řešit – čas ukončení procesu získáme standartní cestou při odchycení zprávy jádra s událostí *exit* pro tento proces. Pokud ovšem tento proces neběží, pak asi nejlepším způsobem vypořádání je přiřadit takovému procesu do *end_time* aktuální systémový čas.

Výše nastíněný přístup je na první pohled problematický. Čtením zpráv jádra operačního systému získáváme velmi přesné časy začátku a konce procesů a při tomto způsobu vypořádání se s neočekávánými situaci bude docházet k řádově větším nepřesnostem (při restartu uzlu může jít minuty). Toto jistě není žádoucí a při monitorování by to zcela jistě vedlo k falešným poplachům. Při ukončení procesu *IDispatcher* na jednom z uzlů dojde k vytvoření tohoto procesu na jiném stroji v rámci clusteru a tento proces se odehraje v nejhorším případu v řádech sekund.

Pro smysluplnou funkci nástroje musíme tedy najít způsob, jak se s těmito procesy vypořádat. Monitorovací nástroj průběžně aktualizuje záznam/y aktuálně běžícího procesu/ů (identifikace na základě PID a chybějícího *end_time*) a vkládá aktuální čas do hodnoty last_modification. Při startu nástroje poté vytáhnou všechny záznamy bez *end_time* a čas *last_modification* se použije jako čas ukončení. Interval aktualizace záznamů funguje jako parametr a jeho vhodným nastavením se vyřeší oba zmíněné případy nekorektního ukončení nástroje/uzlu. Zároveň je důležité si uvědomit, že tento přístup má vliv na výkon nástroje a není vhodné parametr nastavovat na příliš malé hodnoty (tisíciny sekundy a menší).

3.6 Detekce souběhu

Máme implementovánu databázi a způsob získávání dat o běžících procesech na jednotlivých uzlech. Nyní musíme vyhodnotit, zda v rámci běžícího clusteru došlo k souběhu procesů, rozhodnout o závažnosti a poskytnout informace o výsledku.

3.6.1 Získání dat z uzlů

Pro stažení dat z jednotlivých uzlů bylo využito technologie JMX. K implementaci komunikace potřebuje znát také topologii ExpressID AFIS clusteru. K tomuto účelu bylo využito rozhrání webové služby (REST API), které je v ExpressID implementováno.


```
<?xml version="1.0" encoding="UTF-8"?>
<subafises>
  <manager>192.168.1.20</manager>
  <status>OK</status>
  <subafis id="subafis0">
    <dispatcher status="RUNNING" connection-status="OK">192.168.1.20:21233</dispatcher>
    <!-- multiple ip elements mean that the matcher is partitioned -->
          <matcher><ip   status="RUNNING">192.168.1.21:21233</ip><ip   status="NOT_RUNNING">192.168.1.21:21235</ip></
matcher>
  </subafis>
  <processes>
    <process id="webafis-worker" running_count="1" min_count="1" status="OK"/>
  </processes>
</subafises>
```


Obrázek 9 Topologie ExpressID clusteru - REST response

V rámci běhu FP serveru dochází k pravidelnému spuštění naplánované úlohy a to s využitím technologie *Quartz job scheduler* [9]. Pro získání dat jsou využity dva parametry, první udává periodu spouštění úlohy, druhý jak stará data budou z jednotlivých nodů stažena. Perioda spouštění by neměla být delší než interval dotahovaných dat. Pokud bychom spouštěli úlohu každý den a z jednotlivých uzlů clusteru bychom stahovali data o procesech za poslední hodinu, vznikla by situace, bychom detekovali případné problémy jenom v jedné hodině denně.

3.6.2 Souběh procesů

Před samotnou detekcí souběhu provedeme s nad daty úpravu, která neovlivní výsledek, ale usnadní implementaci výsledného algoritmu. V získané sadě dat mohou existovat procesy, které nemají *end_time* (nejspíše budou). Pro tyto procesy aplikujeme postup, který už jsme jednou použili při startu nástroje, tedy nastavíme jako čas ukončení čas *last_modification.* Nyní mají všechny záznamy čas vzniku a ukončení procesu.

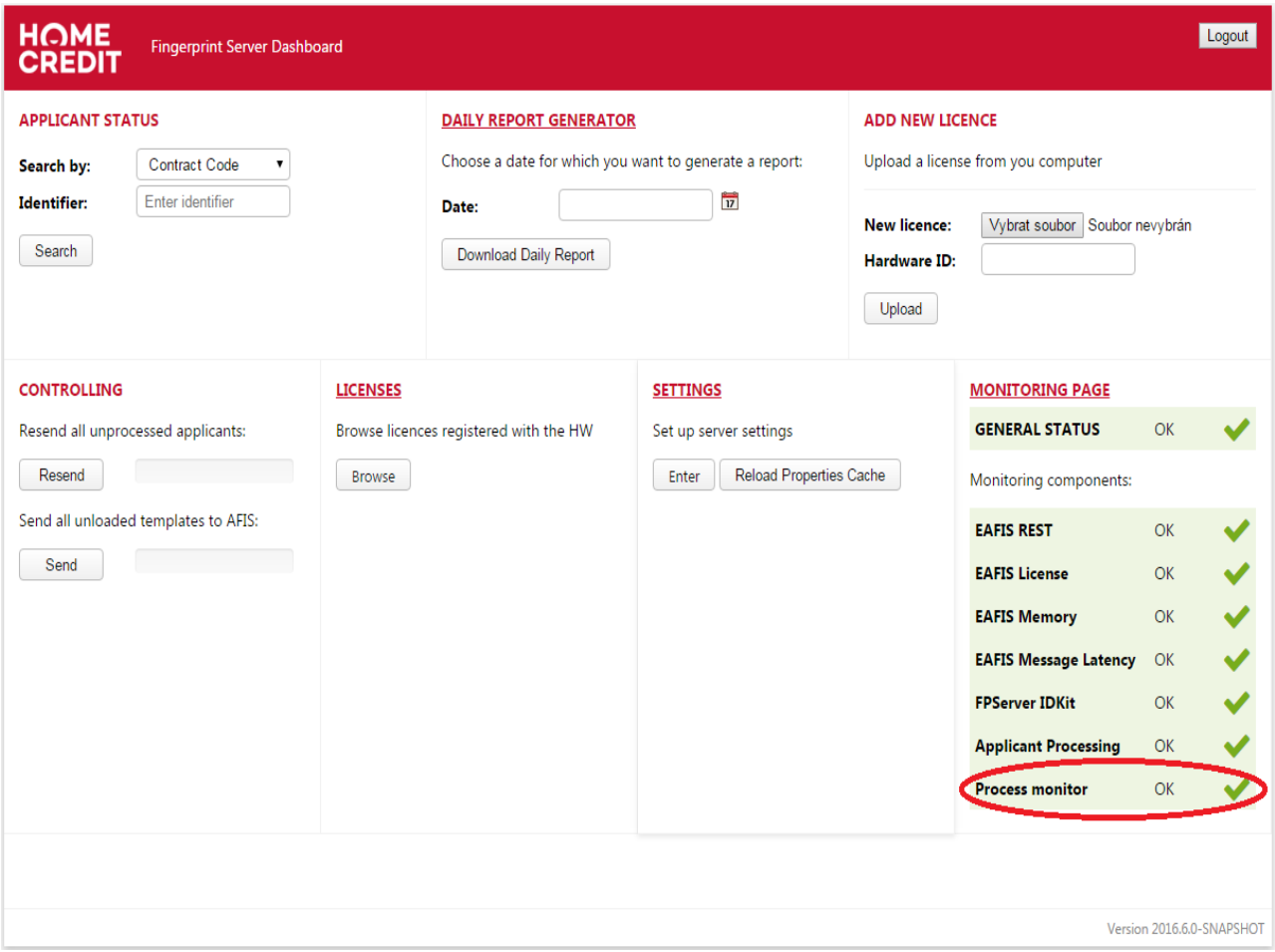
K souběhu procesů dojde v případě, že čas začátku některého z procesů je větší než čas konce jiného z procesů. Implementace algoritmu pro zjištění této situace je poměrně jednoduché. Sta-čí projít získané záznamy pomocí dvou vnořených *for cyklů* a v případě, že narazíme na problém tuto informaci poskytnout. Pro základní informaci, zda došlo k souběhu nepovolných procesů by toto řešení bylo postačující, ale zároveň by bylo dobré poskytnout co nejvíce smysluplných informací, které ze sesbíraných dat můžeme získat. Je možné, že za sledované období došlo k více okamžikům, kdy došlo k souběhu. Pro projdeme celou sadu dat a spočítáme celkový počet těchto událostí. Dále bychom mohli poskytnout informaci, kdy došlo k poslednímu chybovému stavu a také můžeme zobrazit jména (popřípadě IP adresy) uzlů, na kterých došlo k souběhu nepovolených procesů.

3.6.3 Závažnosti stavu a zobrazení

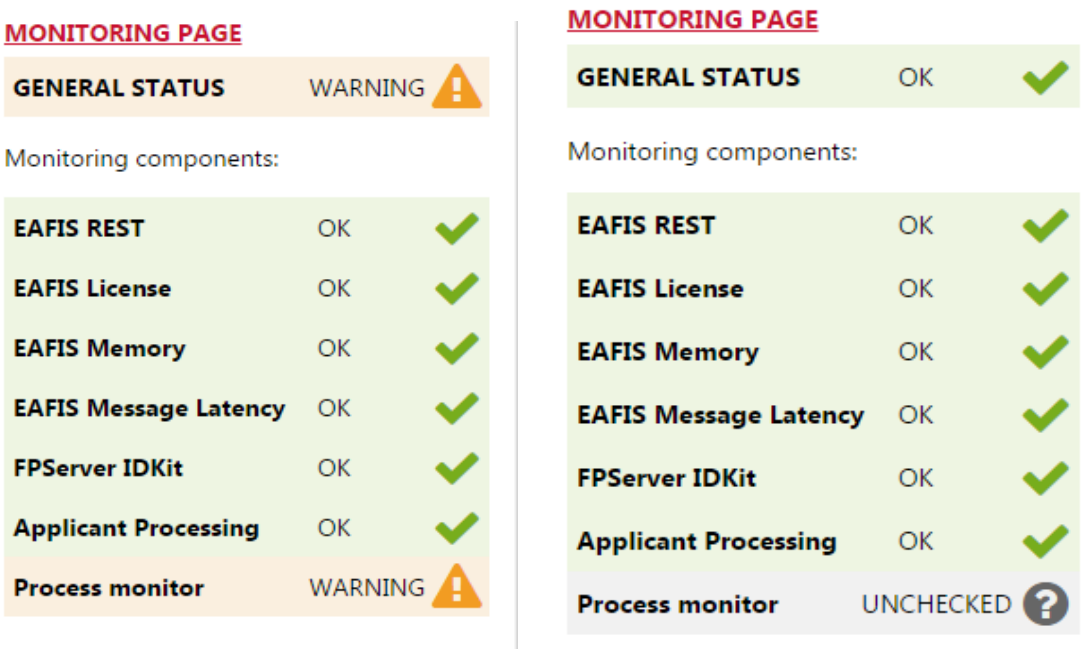
Informace o souběhu nepovolených procesů je zobrazena na dashboardu FP serveru a je také poskytována zpráva ve formátu JSON pro centrální monitoring.

Na hlavním stránce dashboardu je zobrazen jeden ze čtyř stavů:

- OK – nedošlo k souběhu
- WARNING – ve sledovaném období k souběhu došlu, ale již netrvá
- CRITICAL – souběh procesů stále trvá
- UNCHEKED – nepovedlo se získat data



Obrázek 10 FP server dashboard



3.6.4 Rizika špatné funkce nástroje

Z implementace vyplívá několik možných rizik pro správný běh nástroje. Tou hlavní je synchronizace času na jednotlivých uzlech clusteru. Ke sběru dat o vzniku a zániku procesů dochází lokálně na uzlech, kde každý má svůj systémový čas. Pokud by došlo ke značným rozdílům v synchronizaci času, vedlo by to k častým falešným poplachům nebo v nejhorším případě k nemožnosti využití tohoto nástroje.

Dalším rizikem je časté nestandartní situace na uzlech (například častý restart FP serveru). Nástroj se umí dobře vypořádat s procesy, které vznikly/zanikly v době jeho běhu. Naopak problematické určení času startu i konce procesu, který se udál mimo běh nástroje může zhoršovat kvalitu poskytnutých informací.

4 Testování

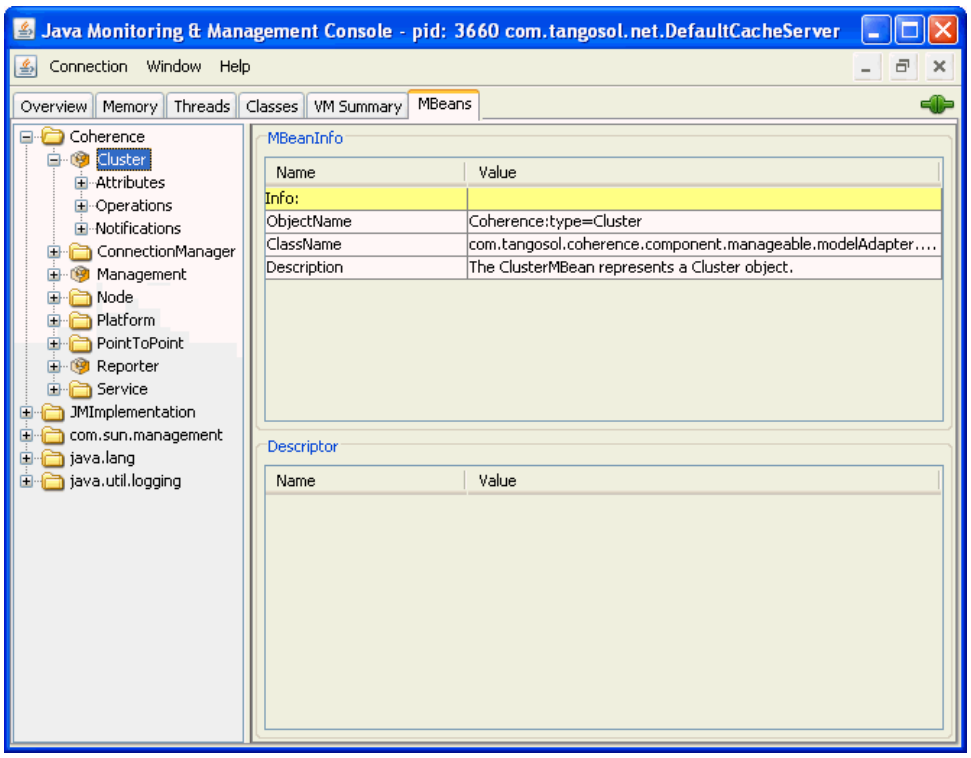
Nástroj je napsán pro operační systém Linux. Platformě závislé věci byly testovány na distribuci Ubuntu, konkrétně verzi 14.04.3 LTS a tak na 32, tak 64 bitové variantě. Testování probíhalo postupně během jeho vývoje. První implementovanou částí bylo přijímání netlink zpráv od jádra operačního systému. K tomu bylo využito výpisu obsahu zpráv v konzoli operačního systému, zatím bez implantace obsahu jednotlivých zpráv. Dále byl vyvinut scanner proc filesystemu a filtrování netlink zpráv podle tytu, výsledky byly ukládány do databáze a bylo možné si definovat proces/y k jehož detekci mělo docházet. K tomuto účelu je možné použít libovolný program, který lze spustit ve více instancích (využito okna webového prohlížeče). Pro toto testovaní jsem použil několik základních scénářů:

- v době startu neběží žádný proces, k spuštění procesu dojde za běhu – testuje zachycení netlink zpráv a jejich správné parsování
    - výsledkem je jeden záznam v databázi s vyplněným *end_time* podle toho, zda došlu k ukončení procesu za běhu nástroje
    - testuje zachytávání netlink zpráv operačního systému a jejich parsování
- sledovaný proces běží při startu nástroje
    - výsledkem je jeden záznam v databázi
    - testu funkčnost scanneru procfs
- kombinace předchozích bodů, tedy proces běžící před startem nástroje a spuštění druhého procesu v době běhu
    - výsledkem jsou dva záznamy v databázi

Poslední větší částí k otestování byla správná interpretace souběhu procesů. Pro tyto účely jsem vytvořil několik sad testovacích dat, které pokrývaly různé situace, ke kterým mohlo dojít. Tedy k souběhu došlo, nedošlo, dále jestli k problematické situaci došlo k minulosti nebo zda stále trvá.

4.1 Další rozvoj

Pro další možnosti rozvoje nástroje se nabízí několik variant. Nyní je nástroj implantován jako součást FP serveru z toho plyne možnost implementace jako samostatně běžícího nástroje. Další z možností poskytnutí vlastního dashboardu/gui pro zobrazení výsledků. Toho může být docíleno poměrně jednoduše poskytnutím JMX rozhraní a poté využít například nástroje JConsole, které je součástí běhového prostředí Java. Popřípadě implementace vlastního řešení pro zobrazení.



Obrázek 11 JConsole

Ze závislosti implementace na operačním systému linux plyne další možnost budoucího rozvoje a to je vytvoření platformě nezávislého nástroje, tedy především implementace umožňující provozování nástroje pro operační systémy Windows.

5 Závěr

V teoretické části práce byla probrána teorie týkající procesů jádra linuxového operačního systéme a technologie, které byly využity k implementaci monitorovacího nástroje a to především technologie Java Native Access, Java Management Extensions a Netlink socket.

V dalších částech práce je podrobně popsaná implementace nástroje, problémy se kterými se bylo potřeba vypořádat a také rizika, která by mohla vést ke zhoršení výsledků nástroje nebo dokonce k úplnému zamezení jeho využití.

Je také důležité říct, proč byl pro implementaci nástroje vybrán jazyk Java, přestože výsledný nástroj nesdílí jednu z hlavních předností této technologie, tedy platformní nezávislost.

Důvodem je implementace ve společnosti Home Credit International a.s., kde je hlavní technologií pro vývoj systémů právě Java a z toho plynoucí požadavky na možnost snazšího začlenění do infrastruktury a taky možnost údržby kódu.

Literatura

[1] M. KERRISK. The Linux Programming Interface: A Linux and UNIX System Programming Handbook. No Starch Press, 2010, ISBN 1-59327-220-0

[2] A. SILBERSCHATZ, P.B.GALVIN a G.GAGNE. Operating systém concepts, 7th edition, ISBN 0-471-69466-5

[3] JSR 3: JavaTM Management Extensions (JMXTM)Specification. Sun Microsystems, 2002. Dostupné z http://jcp.org/en/jsr/summary?id=3, [online] [cit 2016-04-26]

[4] JSR 255: JavaTM Management Extensions (JMXTM) Specifikation 2.0. Sun Microsystems, 2009. Dostupné z: http://jcp.org/en/jsr/detail?id=255. [online] [cit 2016-04-26]

[5] JSR 160: JavaTM Management Extensions (JMX) Remote API. Sun Microsystems, 2003. Dostupné z http://jcp.org/en/jsr/detail?id=160 [online][cit 2016-04-26]

[6] Dokumentace virtuálního filesystemu procfs. Dostupné z http://man7.org/linux/man-pages/man5/proc.5.html , [online][cit 2017-05-10]

[7] Derby Reference Manual, version 10.10. The Apache Software Foundation, 2014

[8] Derby Developer's Guide, version 10.10. The Apache Software Foundation, 2014

[9] Quartz documentation. Dostupné z http://www.quartz-scheduler.org/documentation, [online][cit 2017-05-15]

[10] J. SALIM, H. KHOSRAVI, A. KLEEN, A. KUZNETSOV. Linux Netlink as an IP Services Protocol. RFC 3549. 2003.

[11] A. N. AYUSO, R. M. GASCA, L. LEFEVRE. Communicating between the kernel and user-space in Linux using Netlink sockets. Software: Practise and Experience, 40(9):797-810, 2010

Seznam obrázků

Seznam příloh

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work

(an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution

notices within Derivative Works that You distribute, alongside
or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly

negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS