***Синхронний механізм зв'язку традиційної віртуальної машини на асинхронному сигналі призводить до зростання часу затримки та зниження продуктивності. В роботі застосовано механізм зв'язку, так званий ком-сокет, що використовує міжпроцесорне переривання (МПП) для синхронізації і усунення деяких некорисних перевірок пакетів. Залучено спільне використання пам'яті для скорочення часу копіювання даних. Порівняно з UNIX IPC, ком-сокет володіє меншим часом затримки та більшою продуктивністю***

***Ключові слова: ефективність комунікації, ком-сокети, синхронізація, перевірка пакетів, копіювання пам'яті, час затримки***

***Синхронный механизм связи традиционной виртуальной машины на асинхронном сигнале приводит к росту времени задержки и снижению производительности. В работе применен механизм святи, так называемый ком-сокет, использующий межпроцессорные прерывания (МПП) для синхронизации и устранения некоторых неполезных проверок пакетов. Задействовано совместное использование памяти для сокращения времени копирования данных. По сравнению с UNIX IPC, ком-сокет обладает меньшим временем задержки и большей производительностью***

***Ключевые слова: эффективность коммуникации, ком-сокеты, синхронизация, проверка пакетов, копирование памяти, время задержки***

УДК 004.002
DOI: 10.15587/1729-4061.2016.60629

# ПОБУДОВА ТА ВИКОРИСТАННЯ МІЖДОМЕННОГО МЕХАНІЗМУ ЗВ'ЯЗКУ ДЛЯ ВИСОКОПРОДУКТИВНОЇ ОБРОБКИ ДАНИХ

**В. М. Мельник**
Кандидат фізико-математичних наук, доцент*
E-mail: melnyk_v_m@yahoo.com
**П. А. Пех**
Кандидат технічних наук, доцент, завідувач кафедри*
E-mail: pekh_petro@mail.ru
**К. В. Мельник**
Кандидат технічних наук, доцент*
E-mail: ekaterinamelnik@gmail.com
**Н. В. Багнюк**
Кандидат технічних наук, доцент*
E-mail: bagnjuk_n@rambler.ru
**О. К. Жигаревич**
Асистент*
E-mail: oz_lutsk@mail.ru
*Кафедра комп'ютерної інженерії
Луцький національний технічний університет
вул. Львівська, 75, м. Луцьк, Україна, 43018

## 1. Вступ

Завдяки зростаючій продуктивності комп'ютерного обладнання вже вперше введені технології віртуалізації ставали ефективним способом підвищення коефіцієнта використання апаратних ресурсів. Так, завдяки запуску багатоканальної операційної системи на одній фізичній машині (комп'ютері) ставало очевидним підвищення не тільки використання ресурсів, але й скорочення витрат на управління і отримання більш високої продуктивності і надійності, в порівнянні із роботою одноканальної операційної системи. На сьогодні традиційні розподілені додатки можуть бути вільно розгорнуті на подібній платформі, працюючи легко і прозоро за рахунок високопродуктивних сокетів [1, 2]. Однак при розгортанні деяких мережевих додатків інтенсивного використання, наприклад, таких як інтернет-серверів, файлових мережевих систем, ефективність комунікації між доменами і сьогодні викликає серйозні проблеми.

Хоча більшість МВМ-машин, наприклад, такі як Xen [3], підтримують в своїй роботі деякі методи для забезпечення вимог зв'язку між доменами, з забезпеченням двостороннього зв'язку між доменами. Один із цих зв'язків заснований на протоколі TCP/IP, а інший є новим протоколом, що називається Xen-сокетом [4], який і використовується для отримання високої пропускної здатності. Всі вони потребують МВМ для перехоплювати комунікації, що і призводить до зниження продуктивності і підвищення часу затримки. Таким чином, протокол між доменами зв'язку стає основним вузьким місцем системи при розгортанні роботи деяких додатків з інтенсивною обробкою даних на МВМ-машинах. Наприклад, пропускна здатність Xen-сокетів становить тільки 130 Мбіт, а це набагато нижче, ніж ємність гігабітних мереж Ethernet. Ця ситуація, ускладнюється в основному через низьку продуктивність, обумовлену роботою стека TCP/IP і частотою посторінкової передачі інформації під час сеансу зв'язку.

## 2. Аналіз літературних даних та постановка проблеми

Існують деякі публікації, які розглядають усунення непотрібних перевірок пакетів з метою отримання пропускної здатності передачі даних до 3310 Мбіт/с [5]. І хоча в цій роботі описано удосконалення Xen-сокета з метою отримати пропускну здатність 9295 Мбіт/с, однак *Xen*-сокет являється одностороннім тунельним протоколом, тобто, в той же час, можна сказати, в комунікації є тільки один домен для відправки повідомлення і один домен для його отримання. І коли виникає потреба з двох сторін відправити пакет, то мережеві додатки повинні створювати інший сокет. Це призводить до зростання складності використання *Xen*-сокета. В ряді робіт [6, 7] (з Xen-петлею), [8] (X-шляхом), які в своїх дослідженнях взяли за основу аналогічні споріднені побудови з застосуванням *Xen*-сокета, досягли удосконалення двохсторонньої сумісності через діючі канали зв'язку до мережевого стека. Одже, всі описані вище підходи володіють меншою пропускною здатністю в порівнянні з UNIX IPC і не досягають сподіваних очікувань.

Для того, щоб передавати інформацію на довгому шляху передачі через складне мережеве середовище, Роберт Елліот Кан розробив і втілив у використання TCP/IP протокол. Інформація повинна спочатку помістится в буфер сокета, який повинен бути виділений для протоколу TCP/IP. Далі вона повинна бути перебудована в пакет даних, перед тим як користувач відправить повідомлення. Тільки тоді TCP/IP обчислить контрольну суму створеного пакета даних. Після цього, NIC направляє пакет даних до мережі за допомогою DMA. Якщо користувач хоче отримати повідомлення з мережі, TCP/IP повинен розпакувати пакет даних і перевірити контрольну суму, щоб переконатися, що повідомлення було прийняте правильно. Якщо виявлено, що трапилося що-небудь не так під час цього процесу, відповідне повідомлення буде відправлено знову. Описаний експеримент доводить, що не існує, наприклад, окремого біта даних для випадку, коли допущено необхідністю передати повідомлення на відстань 90 кілометрів через ускладнену мережу.

Вся надійність передачі даних опирається на строго точну і складну перевірку відісланого пакета. На недостаток, в той час як дана технологія забезпечує високу надійність, цей же процес зумовлює додаткові витрати і знижає показник високої продуктивності. Згідно з аналізом, проробленим в роботі [9], процедура процесу TCP/IP займає до 15 % процесорного часу CPU, а копіювання даних і здійснення перевірки пакета буде забирати навіть до 34 % процесорного часу CPU. В роботі [10] було проаналізовано ефект впливу TCP/IP на операційну систему. Система змушена жертвувати 14 % власної продуктивності для управління буфером даних, а відповідні драйвери займатимуть 20 % процесорного часу CPU, в той час як процедура процесу TCP/IP займає лише 26 % процесорного часу CPU.

В роботі [9] також здійснюється аналіз завантаження пам'яті, обумовлене TCP/IP. У цій роботі *ops*-пам'ять може бути розділена згідно призначення на три типи: читання даних з простору користувача в простір ядра, яка обумовлює 0,1 % накладних витрат; запис даних в буфер сокета, який зумовлює 1.1 % накладних витрат; і читання та надсилання даних від NIC, які разом обумовлюють до 2,5 % накладних витрат.

Роблячи підсумок цих експериментів, можна зробити висновок, що причиною спаду продуктивності роботи TCP/IP є ускладнення управління буфером, перевірки даних та необгрунтовані рамки керування, однак не часто *ops*-пам'ять. Процедура копіювання даних у фізичній машині є досить надійною і можна отримати високу продуктивність передачі даних за рахунок усунення процедури перевірки даних і скорочення шляху проходу даних.

## 3. Мета і завдання дослідження

*Метою* даної роботи ставиться розробка і можливість застосування високопродуктивного механізму міждоменного комунікаційного зв'язку, названого *ком-сокетом*, який базується на механізмі комунікації зі спільною (розділювальною) пам'яттю.

Використання МПП замість асинхронного повідомлення в процесі доставки має сприяти зниженню часу затримки ком-сокета. Передбачається, що механізм використання спільної пам'яті не тільки має скорочувати час копії пам'яті, але і дозволяти уникати накладних витрат, обумовлених посторінковою пропускною здатністю. Перспективним є те, що МВМ не буде потрібно перехоплювати в комунікації між доменами, що також дозволить уникнути накладних витрат, спричинених МВМ-викликом і спричинить підвищенню продуктивності.

З іншої сторони актуальним є те, що ком-сокет відповідає стандарту сокетів Берклі. Це говорить про те, що програміст може легко його використовувати. Прототип ком-сокета реалізований на базі процесорів x86 МВМ.

Завданнями дослідження, вирішення яких представляється доцільним для досягнення цієї мети, обрано:

– створити ком-сокет з використанням спільної пам'яті;

– запрограмувати необхідні програмні структури для забезпечення функціонування ком-сокета;

– здійснити реалізацію ком-сокета;

– провести порівняльну характеристику його роботи з іншими видами зв'язку, враховуючи час затримки та продуктивність.

## 4. Розробка та спосіб реалізації ком-сокета

Стандартний сокетий інтерфейс забезпечується ком-сокетом, яким користувач може так же само легко використовуватися, як і при використанні IPC UNIX. То ж зараз буде описано детальну інформацію про принципи проектування ком-сокета та його впровадження, включаючи рамки роботи, управління буфером, оповіщення про повідомлення і інше.

Створення TCP/IP включає в себе умову забезпечення правильності та надійності передачі даних на ускладненій мережі. Таким чином, в протоколі TCP/IP є деякі ускладнені і часомісткі операції, які було проаналізовано в попередньому розділі. Можна припустити, що ніякої помилки не відбувається при копіюванні

даних у фізичній машині, оскільки весь цей процес відбувається в стабільному середовищі і високій надійності перевірки ECC пам'яті. Однак, без будь-яких модифікацій це призведе до низької продуктивності в умовах використання TCP/IP протоколів у міждоменному зв'язку. Таким чином, для того, щоб отримати високу продуктивність, ком-сокети повинні усунути непотрібні витрати часу і перевірку пакетів даних та спростити управління стеком.

Традиційна міждоменна комунікація, яка базується на MBM, повинна і часто здійснювати MBM-виклик. Проте добре відомо, що MBM-виклик є причиною частих операцій TLB-виключень, які забруднюють L2-кеш, переповнюючи *pipe*-лінію і так далі. Все це призведе до прямих і непрямих затрат які надалі негативно впливають на продуктивність роботи додатків як в режимі виконання користувача, так і в режимі виконання ядра. Таким чином, здійснення MBM-виклику повинно бути як можна меншим, і навіть з уникненням його під час передачі даних.

Однак, як відомо, досить важко виділити кілька послідовних сторінок в пам'яті, якщо операційна система працює протягом тривалого часу, тому що це викликатиме часті *swap*-операції і навіть може призводити до фрагментації пам'яті. То ж ще один важливий принцип дизайну ком-сокета направлений на зменшення частоти виділення пам'яті, наскільки це можливо реалізувати, щоб зменшити вплив на систему пам'яті в цілому. У той же час, інтерфейс ком-сокета повинен слідувати *Berkeley*-стандарт сокета, щоб зробити його зручним для використання програмістами-розробниками.

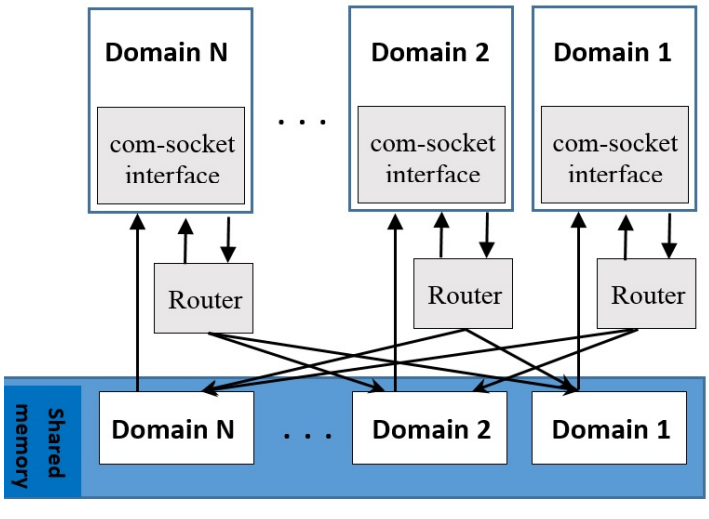


Рис. 1. Архітектура ком-сокета

Рис. 1 показує архітектуру ком-сокета. Програміст може використовувати інтерфейс ком-сокета для взаємодії з іншими доменами. В кожному домені є маршрутизатор, який визначає шлях і домен-одержувач для відправки повідомлення. Якщо домен відправляє повідомлення самому собі, то маршрутизатор передаватиме повідомлення до відповідного сокета, і якщо цільовими є інші домени, то маршрутизатор скопіює повідомлення в буфер, що відповідає відповідному домену і побудує відповідну структуру даних для опису повідомлення, поклавши її в чергу цільового домена-приймача. Якщо цільового домена не існує, маршрутизатор буде відкидати повідомлення і повертає код помилки для додатка. Для забезпечення умови ізоляції та закритості доступу між доменами, кожен домен може читати тільки свій доступний йому буфер і записувати в буфери інших доменів.

В кожному домені є власний *daemon* для сканування черги приймача, виявлення і отримання інформації про опис повідомлення, з якого він і отримує адресу повідомлення в пам'яті. В опис структури повідомлення подається інформація про відправника повідомлення, приймача і все інше, яка докладно буде описана нижче в цьому розділі публікації.

Відповідний буфер спільної пам'яті використовується ком-сокетом для передачі даних. Він резервується MBM безпосередньо при старті операційної системи, і вона фіксує фізичну адресу цього буфера в адресний простір через MBM-виклик. Кожен домен має власний буфер який є доступним іншим доменам, в який вони можуть здійснювати тільки запис, і який може тільки зчитуватися власним доменом.

Як показано на рис. 2, кожен такий відповідний домену буфер ділиться на блоки по 8 КБ, які помічені біт-картою відображення. Коли домен посилає деяке повідомлення, він повинен перш за все запросити блок для того, щоб розмістити його в нього і маскувати відповідний біт в біт-карті відображення, щоб вказати, що блок використовується (зайнятий), а потім розмістити інформацію опису повідомлення в черзі прийому для відповідного домена і, при необхідності, повідомити про це домен-приймач. Приймаючий *daemon* повинен сканувати власну чергу прийому повідомлень з метою отримати інформацію опису повідомлення, за допомогою якої він повинен отримати розмір повідомлення, адресу пам'яті та іншу корисну інформацію про повідомлення прийому. Далі він повинен побудувати відповідну структуру і розмістити його у відповідний список отримувача сокета, з якого додаток і отримає повідомлення. Після того, як повідомлення буде прочитане, ком-сокет очистить відповідний біт повідомлення в біт-карті відображення для звільнення місця пам'яті.

Через вищенаведений опис можна зрозуміти, що немає MBM-викликів під час передачі даних, а тільки потрібно здійснити всього два рази копіювання даних: перший раз – з простору користувача-відправника до відповідної спільної пам'яті, а другий – з відповідної пам'яті у простір користувача-приймача.

Для того щоб зменшити складність програмування і підтримки програмного стилю, ком-сокет використовує ту ж структуру адреси, що і IPv4-формат. Якщо адресою з'єднання є 192.168.0.0, то вона є закритою адресою, а IP-адреса кожного домена містить власний ідентифікатор домена збільшений на 2. Наприклад, якщо ідентифікатор домена рівний 3, то відповідна IP-адреса для нього буде 192.168.0.5. По аналогії як і IPv4, адреса 192.168.0.255 використовується для мовлення. Номер порта використовується операційною системою для того, щоб відокремити різні додатки з метою забезпечення неповторності відправки та отримання повідомлень.

Оповіщення про надходження повідомлення здійснюється в наступному порядку. Є дві схеми, які мо-

жуть бути використані в якості механізму оповіщення повідомлення: перша схема використовує таймер для сканування буфера з метою перевірити наявність повідомлень, що очікують обробки; інша – використовує дозвіл на виконання завдання перевірки буфера, коли здійснюється кожне переривання. Всі вони або спричиняють великий час затримки, або призводять до високого завантаження CPU. Зі сторони цих причин ком-сокет використовує міжпроцесорні-переривання (МПП) для того, щоб повідомити цільовий домен про надходження повідомлення для нього.

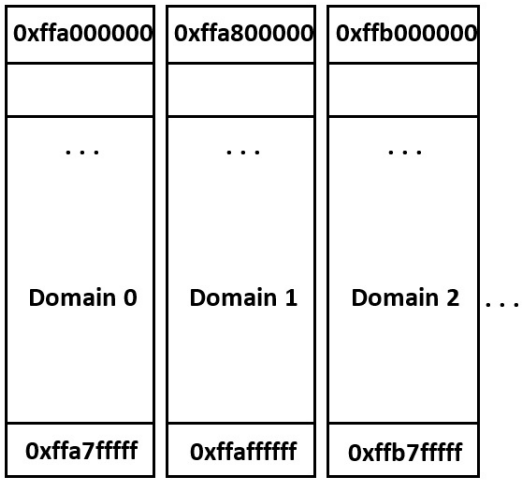


Рис. 2. Архітектура пам'яті у відповідності до домена

МПП відправляється з відправкою повідомлення і цільовий домен буде спланувати *daemon* прийому для аналізу буфера повідомлень і отримання відісланого повідомлення. В зв'язку з високою значимістю МПП не могло бути згенерованим після відправки повідомлення. Ком-сокет використовує тег, щоб відкривати/закривати через прапор статусу прийомний *daemon*. Якщо *daemon* в цю мить обробляє повідомлення, то нове повідомлення буде оброблено автоматично. У цій ситуації, немає необхідності в МПП. В іншому випадку, потрібно відправити МПП до домена прийому з метою повідомити приймач про явне прибуття нового повідомлення.

Однією з важливих структур даних ком-сокета є його сокетна структура під назвою *com_sock*. Саме її та пов'язані з нею структури наведено нижче.

```
struct com_sock
{
 struct sock sk;
 struct com_addr dest;
 struct com_addr sour;
 struct com_skb_head skb_head;
 spinlock_t lock;
};
struct com_skb
{
 struct com_skb *next;
 struct com_buf *msg;
 int port;
 int offset;
 int len;
};
struct com_rcv_msg
{
 int msg_len;
 struct com_addr sour;
 int dest_port;
 int flags;
 char type;
 struct com_buff msg_data;
 int msg_num;
};
struct domain_buff_control
{
 int start;
 int msg_index[TAB_ENTRYS];
 spinlock_t buf_lock;
 long empty[TAB_ENTRYS/64];
 spinlock_t recv_lock;
 int end;
 struct com_recv_msg msg [TAB_ENTRYS];
};
```

В структуру *com_sock* включена також і стандартна сокетна структура *sock*, а також поля цільової адреси одержувача, адреси відправника та список повідомлень. Список повідомлень будується за зразком структури *com_skb*. Для того щоб забезпечити синхронізацію, структура *com_sock* використовує спін-блокування (параметр типу *spinlock_t*). Тип повідомлення, довжина, зсув, адреса і деяка інша корисна інформація, – все вложено в структуру *com_skb*.

Проста діаграма потоків даних подана на рис. 3. Коли користувач реалізує системний виклик *send()/write()* з попаданням в ядро, відправник повинен насамперед просканувати бітову карту *bit-map* цільового домена, щоб отримати вільний блок. Щоб скопіювати дані безпосередньо в спільний буфер пам'яті цільового домена використовується функція *copy_from_iovc()*. Якщо цільовий домен не обробляє повідомлення, відправнику також необхідно відправити МПП до нього, щоб повідомити його про явне надходження повідомлення. Приймаючий *daemon* здійснює сканування списку повідомлень для здійснення обробки повідомлення, що надійшло, і помістить його у *skb*-список відповідного цільового сокета. Коли користувач застосовує системний виклик *receive()/read()*з попаданням в ядро, операційна система буде використовувати функцію *get_msg_from_skb()*, щоб отримати повідомлення із *skb*-списку, а потім за допомогою функції *copy_to_iovc()* скопіює дані безпосередньо в простір користувача. Після цього операційна система очистить відповідний біт в бітовій карті *bit-map*, щоб звільнити блок.

Можна зробити висновок з усього прокоментованого вище процесу, що є тільки два рази копіювання даних з одного домену в інший, і немає необхідності у МВМ-виклику протягом усього цього процесу.

```
SERVER
sockfd=socket(AF_COM, ...);
...
bind(sockfd, ...);
listen(sockfd, ...);
...
recvsock=accept(sockfd, ...);
```

```
. . .
read(recvsock, ...);
. . .
write(recvsockfd, ...);
. . .
close(recvsock);
CLIENT
sockfd=socket(AF_COM, ...);
. . .
. . .
connect(sockfd, ...);
. . .
write(sockfd, ...);
. . .
. . .
. . .
read(sockfd, ...);
. . .
close(sockfd, ...);
```

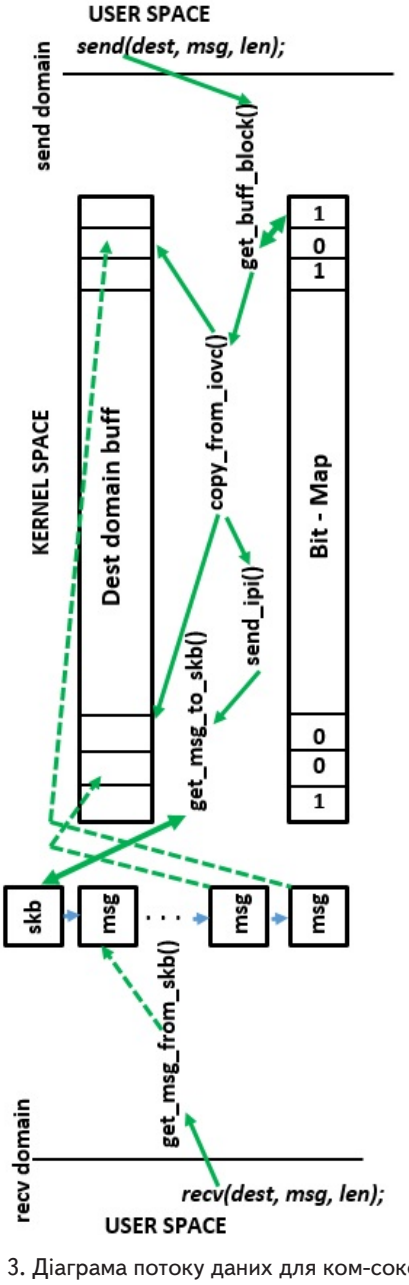


Рис. 3. Діаграма потоку даних для ком-сокета

## 4. Обговорення результатів досліджень ком-сокета

Далі можна попробувати реалізувати прототип ком-сокета на базі системи x86 МВМ яка може запускати кілька примірників операційної системи в один і той же час без додаткових продуктивних затрат. Ядро виділяє ресурси для операційної системи і захищає її від несанкціонованого доступу інших користувачів. Таке мікроядро відтворює хорошу продуктивність і з ним одночасно можуть працювати багато інших серверів, що базуються на ядерній основі, проявляючи, як і традиційні МВМ, гнучкість і широкомасштабність роботи.

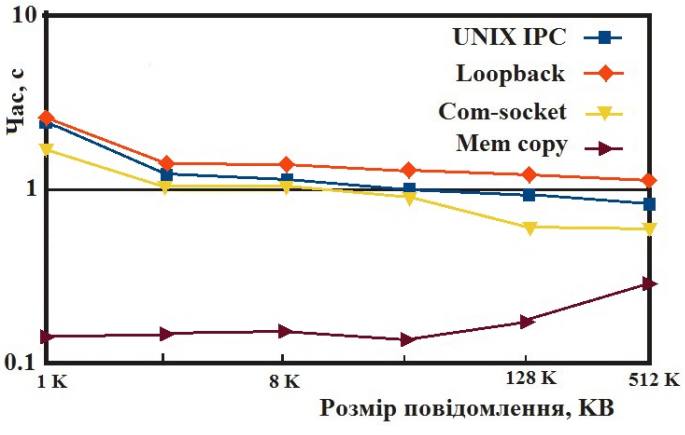


Рис. 4. Ком-сокет, UNIX IPC та порівняння продуктивності копіювання пам'яті

Ця ідея досить добре підходить для cloud-комп'ютерної обробки даних. У цьому дослідженні для тестування продуктивності ком-сокета використано двохпроцесорний AMD Opteron сервер DELL T605 з 16 ГБ DRAM. Виявлено, що ком-сокет має більш високу продуктивність у порівнянні з UNIX IPC і досягає очікуваних результатів. Рис. 4 показує результати досліджень ком-сокета, UNIX IPC та порівняння продуктивності копіювання пам'яті. В даній роботі не порівнюються результати досліджень ком-сокета з результатами Xen-сокета, тому що Xen-сокет і X-шлях мають більш низьку продуктивність згідно наведених літературних даних у порівнянні з UNIX IPC [4, 8]. То ж, оскільки немає МВМ-перехоплення при обробці комунікації, ком-сокет також отримаєте високу продуктивність в іншому МВМі.

По-перше, порівняємо пропускну здатність кожного протоколу зв'язку, включаючи, UNIX IPC зв'язок між доменами, Loopback і ком-сокет. Щоб продемонструвати високий рівень комунікації, можна також перевірити швидкість копіювання пам'яті. У цьому експерименті відправляєтьсч 1GB даних на інший домен для розміру повідомлення в діапазоні від 1 до 512 Кб. Чим менше часу (рис. 4) використовується для кожного протоколу, тим більш високу пропускну здатність він виявляє в роботі. Як видно з рисунка, смуга пропускання ком-сокета значно вища, ніж в інших протоколах у випадках, коли блок повідомлень є об'ємно малим. Це проявляється тому, що ком-сокет спрощує управління буфером. У той же час, висока продуктивність виділеного буфера використовується ком-сокетом, щоб уникнути виклик *kmalloc()* і щоб звільнитися від зайвої операції, яка займає відповідний проміжок часу. Використання МПП дозволяє уникнути МВМ-виклику при відправленні

повідомлення, який все-таки використовується Xen-сокетом [5], кожен раз. Це робить ком-сокет більш простим і ефективним.

За допомогою літературних даних [7, 8] та експериментальних вимірювань здійснено порівняння часу затримки при встановленні операції з'єднання для кожного виду протоколу. Ком-сокет володіє меншим часом затримки, (50 с.) ніж інший протокол (Xen [8] – 210 с., Loopback – 71 с.), за винятком UNIX IPC (21 с.), і є більш ефективним, ніж інші протоколи. Це проявляється тому, що, при встановленні з'єднання, ком-сокет повинен відправити переривання МПП на сервер і сервер також має відправити МПП клієнту. Оскільки операція МПП вимагає відповідних часових затрат, то ком-сокет і володіє дещо вищим часом затримки в порівнянні з UNIX IPC.

## 5. Висновки

В роботі створено і застосовано механізм зв'язку, ком-сокет, що використовує міжпроцесорне переривання (МПП) для синхронізації і усунення деяких некорисних перевірок пакетів при передачі даних. Запрограмовано робочі структури для функціонування *ком*-сокета зі спільним використанням пам'яті (shared memory) для скорочення часу копіювання даних. Здійснено також порівняльну характеристику параметрів часу затримки та продуктивності. Виявлено, що ком-сокет володіє меншим часом затримки та більшою продуктивністю порівняно з іншими видами зв'язку, в тому числі і з UNIX IPC.

Хоча ком-сокет демонструє високу продуктивність, однак він не підтримує повної бінарної сумісності, тобто додаток не може використовувати ком-сокет безпосередньо, без певної модифікації. З іншого боку, ком-сокет не підтримує асинхронний механізм, а потребує здійснювати сканування списку повідомлень з метою виявлення в ньому нового, що надходить у випадку, коли користувач хоче отримати нове повідомлення. Таким чином, виходячи з даної роботи було б корисно відтворити ком-сокет з підтримкою бінарної сумісності та асинхронного механізму.

Література


1. Мельник, В. М. Вплив високопродуктивних сокетів на інтенсивність обробки даних [Текст] / В. М. Мельник, Н. В. Багнюк, К. В. Мельник // ScienceRise. – 2015. – Т. 6, № 2 (11). – С. 38–48. doi: 10.15587/2313-8416.2015.44380
2. Melnyk, V. M. High production of java sockets for health clouds in science [Текст] / V. M. Melnyk, O. K. Zhygarevich, K. V. Melnyk // Engineering Software. – 2014. – Vol. 19, Issue 3. – P. 36–40.
3. Barham, P. Xen and the art of virtualization [Text] / P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield // In Proceedings of the nineteenth ACM symposium on Operating systems principles. ACM, 2003. – P. 164–177.
4. Zhang, X. Xen-socket: A high-throughput interdomain transport for virtual machines [Text] / X. Zhang, S. McIntosh, P. Rohatgi, J. Griffin // Lecture Notes in Computer Science, 2007. – P. 184–203. doi: 10.1007/978-3-540-76778-7_10
5. Menon, A. Optimizing network virtualization in Xen [Text] / A. Menon, A. L. Cox, W. Zwaenepoel // In Proceedings of the annual conference on USENIX'06 Annual Technical Conference: USENIX Association, 2006. – P. 2.
6. Burtsev, A. Fido: Fast intervirtual-machine communication for enterprise appliances [Text] / A. Burtsev, K. Srinivasan, P. Radhakrishnan, L. N. Bairavasundaram, K. Voruganti, G. R. Goodson // In Proceedings of the 2009 conference on USENIX Annual technical conference: USENIX Association, 2009. – P. 25–25.
7. Wang, J. Xen-loop: a transparent high performance inter-vm network loopback [Text] / J. Wang, K. L. Wright, K. Gopalan // In Proceedings of the 17-th international symposium on High performance distributed computing: ACM, 2008. – P. 109–118. doi: 10.1145/1383422.1383437
8. Kim, K. Interdomain socket communications supporting high performance and full binary compatibility on Xen [Text] / K. Kim, C. Kim, S. I. Jung, H. S. Shin, J. S. Kim // In Proceedings of the fourth ACM SIGPLAN / SIGOPS international conference on Virtual execution environments, 2008. – P. 11–20. doi: 10.1145/1346256.1346259
9. Foong, A. P. TCP performance revisited [Text] / A. P. Foong, T. R. Huff, H. H. Hum, J. R. Patwardhan, G. J. Regnier // IEEE International Symposium on Performance Analysis of Systems and Software. ISPASS 2003, 2003. – P. 70–79. doi: 10.1109/ispass.2003.1190234
10. Minturn, D. Addressing TCP/IP processing challenges using the IA and IXP processors [Text] / D. Minturn, G. Regnier, J. Krueger, R. Iyer, S. Makineni. // Intel Technology Journal. – 2003. – Vol. 7, Issue 4. – P. 39–50.