УДК 004.94

РЕАЛИЗАЦИЯ ПЕРЕДАЧИ СООБЩЕНИЙ МЕЖДУ ПРОЦЕССАМИ В МНОГОПРОЦЕССОРНОЙ СИСТЕМЕ

Мартышкин Алексей Иванович, кандидат технических наук, доцент,
доцент кафедры «Вычислительные машины и системы»
Пензенский государственный технологический университет
(440039, Россия, г. Пенза, проезд Байдукова/ул. Гагарина, 1а/11, e-mail: alexey314@yandex.ru)
Воронцов Александр Анатольевич, кандидат технических наук,
доцент кафедры «Вычислительные машины и системы»
Пензенский государственный технологический университет
(440039, Россия, г. Пенза, проезд Байдукова/ул. Гагарина, 1а/11, e-mail: aleksander.vorontsov@gmail.com)

Аннотация. В статье описываются способы реализации передачи сообщений между процессами в многопроцессорной системе. Целью проводимого исследования является исследование методов управления взаимодействующими процессами на основе классических задач «читатели-писатели» и «производители-потребители». Представлены возможные решения указанных задач с применением синхронизирующих механизмов монитора и семафора. Кроме того, в статье рассмотрены некоторые методы межпроцессного обмена на примере именованных и неименованных каналов и общей памяти. Во второй половине работы затронуты вопросы, касающиеся исследования программных средств измерения производительности каналов межпроцессного обмена. Рассмотренные в статье методы межпроцессного обмена обладают разной производительностью, для измерения которой воспользовались программным пакетом lmbench. Полученные основные результаты проведенных тестов отражены на графиках. В завершении работы было проведено сравнение каналов и общей памяти по двум параметрам: полоса пропускания и время задержки. В заключение были сделаны основные выводы по проделанной работе.

Ключевые слова: операционная система, взаимодействие процессов, база данных, транзакция, разделяемая память, процесс-писатель, процесс-читатель, семафор, монитор, общий ресурс.

THE IMPLEMENTATION OF MESSAGE PASSING BETWEEN PROCESSES IN A MULTIPROCESSOR SYSTEM

Martyshkin Alexey Ivanovich, candidate of technical sciences, docent,
associate Professor of sub-department «Computers and systems»
Penza State Technological University
(440039, Russia, Penza, Baydukov Proyezd / Gagarin St., 1a/11, e-mail: alexey314@yandex.ru*)*
Vorontsov Aleksandr Anatolievich, candidate of technical sciences,
associate Professor of sub-department «Computers and systems»
Penza State Technological University
(440039, Russia, Penza, Baydukov Proyezd / Gagarin St., 1a/11, e-mail: aleksander.vorontsov@gmail.com)

Abstract. The article describes how to implement the transfer of messages between processes in a multiprocessor system. The purpose of the research is to study the methods of managing the interacting processes on the basis of the classic tasks of “reader-writers” and “producer-consumers”. The possible solutions to these problems using the monitor and semaphore synchronization mechanisms are presented. In addition, the article discusses some methods of interprocess exchange on the example of named and unnamed channels and shared memory. In the second half of the work, issues related to the study of software tools for measuring the performance of inter-process exchange channels were raised. The interprocess exchange methods considered in the article have different performance, for the measurement of which they used the lmbench software package. The obtained main results of the tests are reflected in the graphs. At the end of the work, a comparison was made between channels and shared memory using two parameters: bandwidth and delay time. In conclusion, the main conclusions were made on the work done.

Keywords: operating system, process interaction, database, transaction, shared memory, writer process, reader process, semaphore, monitor, shared resource.

Введение. В связи с постоянным увеличением объемов обрабатываемой информации и увеличением параллелизма алгоритмов обработки требуется увеличивать производительность систем и повышать скорость обмена между параллельными ветвями алгоритмов. Стандартные средства обмена – IPC, являются программными средствами, а поэтому обладают недостаточными скоростями. Решением проблемы может служить аппаратная реализация средств межпроцессного обмена. Аппаратная реализация различных алгоритмов часто встречается в вычислительной технике и позволяет сильно повысить производительно, а часто и снизить ее себестоимость.

Исследование методов управления взаимодействующими процессами на основе классической задач «читатели-писатели», «производители-потребители» с применением синхронизирующего механизма монитора. В статье рассмотрены методы управления взаимодействия процессов, базирующиеся на принципах классических задач: «читатели-писатели», «производители-потребители» [1–4].

Задача «читатели-писатели». Базу данных делят между собой два типа процессов – читатели, которые выполняют транзакции, просматривающие записи базы данных, и писатели, транзакции которых могут просматривать и изменять записи. Чтобы исключить взаимное влияние транзакций друг на друга процесс-писатель должен иметь первоочередной доступ к базе данных, к которой, в случае отсутствия обращения процессов-писателей, могут обратиться для выполнения транзакций сколько угодно процессов-читателей.

Приведенное выше определение касается разделяемой (общей для всех) базы данных, но ею может быть также и файл, связанный список, таблица и т.д. В простейшем случае представленная задача решается с использованием двух семафоров. Один закрывает доступ в базе данных, второй к переменной, хранящей число активных читателей [5,11].

В проведенных экспериментах параметр nr нужен для подсчета числа активных читателей. В протоколе входа для процессов-читателей сначала значение переменной nr увеличивается на 1, затем проверяется, равно ли оно 1.

В данном решение имеется существенный недостаток. Выражается он в том, что процессы читатели имеют приоритет относительно процессов писателей. Предположим, один из читателей на данный момент использует базу данных. В любой момент может появиться еще один читатель. Из принципов задачи о читателях и писателях одновременная работа двух читателей не запрещается и допустима, второй читатель может обратиться к базе данных. Как только будут появляться другие процессы-читатели, они также могут быть допущены к базе данных без каких-либо ограничений.

Теперь рассмотрим решение этой же задачи, с использование высокоуровневого синхронизационного примитива, называемого монитор [2,6,7]. Есть два типа процессов и по два вида действий на процесс, поэтому получаем четыре процедуры монитора: request_read, release_read, request_write, release_write. Для синхронизации доступа к базе данных необходимо вести учет числа записывающих и читающих процессов.

В ходе эксперимента, соответствующий этой спецификации, определено, что в начале процедуры request_read процесс-читатель должен приостановиться, пока nw не станет равной 0; эта задержка происходит на условной переменной oktoread. Аналогично процесс-писатель вначале процедуры request_write до обнуления переменных nr и nw должен приостановиться на условной переменной oktowrite. В процедуре release_read для процесса-писателя вырабатывается сигнал, когда значение nr равно нулю. Поскольку писатели выполняют перепроверку условия своей задержки, данное решение является правильным, даже если процессы-писатели всегда получают сигнал. Однако это решение будет менее эффективным, поскольку получивший сигнал процесс-писатель при нулевом значении nr должен снова приостановиться. С другой стороны, в конце процедуры release_write точно известно, что значения обеих переменных nr и nw равны нулю. Следовательно, может продолжить работу любой приостановленный процесс. Представленное здесь решение не устанавливает порядок чередования процессов-читателей и процессов-писателей.

Задача «Производители и потребители». Имеется огромное число вариантов постановки и решения такой задачи в рамках конкретных операционных систем (ОС) [1,8–10].

Рассмотрим решение данной задачи с использование описанного выше монитора. Процесс-производитель и процесс-потребитель взаимодействуют с помощью разделяемого буфера, состоящего из n ячеек. Для представления очереди сообщений использованы массив buf и две целочисленные переменные front и rear, которые указывают соответственно на первую заполненную и первую пустую ячейку. В целочисленной переменной count хранится количество сообщений в буфере. Операции с буфером deposit (поместить в буфер) и fetch (извлечь из буфера) становятся процедурами монитора. Взаимное исключение неявно, поэтому семафорам не нужно защищать критические секции. Условная синхронизация реализована с помощью двух условных переменных [11].

Обрабатывая операцию signal, процесс просто сообщает, что теперь некоторое условие истинно. Поскольку процесс-сигнализатор и, возможно, другие процессы могут выполняться в мониторе до возобновления процесса, запущенного операцией signal, в момент начала его работы условие запуска может уже не выполняться. Например, процесс-производитель был приостановлен в ожидании свободной ячейки, затем процесс-потребитель извлек сообщение и запустил приостановленный процесс. Однако до того, как этому производителю пришла очередь выполняться, другой процесс-производитель мог уже войти в процедуру deposit и занять пустую ячейку. Аналогичная ситуация может возникнуть и с потребителями. Таким образом, условие приостановки необходимо перепроверять.

Операторы signal в процедурах deposit и fetch выполняются безусловно, поскольку в момент их выполнения условие, о котором они сигнализируют, является истинным. В действительности операторы wait находятся в циклах, поэтому операторы signal

могут выполняться в любой момент времени, поскольку они просто дают подсказку приостановленным процессам. Однако программа выполняется более эффективно, когда signal выполняется, только если известно наверняка (или хотя бы с большой вероятностью), что некоторый приостановленный процесс может быть продолжен.

Если данные производятся и потребляются примерно с одинаковой частотой, то процессу не приходится долго ждать доступа к буферу. Однако обычно потребитель и производитель работают неравномерно. Например, производитель может быстро создать сразу несколько элементов, а затем долго вычислять до следующей серии элементов. В таких случаях увеличение емкости буфера может существенно повысить производительность программы, уменьшая число блокировок процессов. (Это пример классического противоречия между временем вычислений и объемом памяти) [5,12–16].

Здесь решается так называемая задача о кольцевом буфере, который используется в качестве многоэлементного коммуникационного буфера. Представим буфер массивом buf(n), где n > 1. Пусть переменная front является индексом первого сообщения очереди, a rear — индексом первой пустой ячейки после сообщения в конце очереди. Вначале переменные front и rear имеют одинаковые значения, скажем, 0.

При таком представлении буфера производитель помещает в него сообщение со значением data, выполнив следующие действия:

buf(rear) = data; rear = (rear+1) % n;

Аналогично потребитель извлекает сообщение в свою локальную переменную result, выполняя действия:

result = buf(front); front = (front+1) % n;

Оператор взятия остатка (%) используется для того, чтобы значения переменных front и rear всегда были в пределах от 0 до n-1. Очередь буферизованных сообщений хранится в ячейках от buf[front] до buf[rear] (не включительно). Переменная buf интерпретируется как кольцевой массив, в котором за buf(n-1) следует buf(0). Вот пример конфигурации массива buf (рисунок 1).

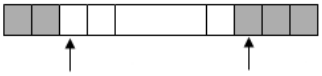

Рисунок 1 – Общее представление кольцевого буфера

Затемненные ячейки заполнены, белые – пусты. Таким образом, можно получить условия пустоты и заполненности буфера: если rear = front, то буфер пуст; если front – 1 = rear, то буфер полон.

Исследование методов межпроцессного обмена в многопроцессорных системах. Межпроцессное взаимодействие (IPC) – набор способов обмена данными между множеством потоков в одном или более процессах, которые могут быть активированы на одном или нескольких компьютерах, объединенных в сеть. В рамках работы рассмотрены три метода межпроцессного обмена: программные каналы, очереди FIFO и разделяемая память.

Именованные (программные) каналы присутствуют во всех известных версиях Unix-систем. Канал создается системным вызовом pipe и предоставляет возможность однонаправленной передачи данных. В этом случае функция возвращает два файловых дескриптора: fd[0] и fd[l], причем первый открыт для чтения, а второй, в свою очередь, открыт для записи.

На рисунке 2 показан канал при занятии его единственным процессом. Хотя канал создается и поддерживается всего одним процессом, он редко используется лишь этим конкретным процессом. Как правило, каналы применяются для связи между родительским и дочерним процессами. В общем, это происходит следующим образом: процесс создает канал, а затем вызывает функцию fork, создавая свою копию – дочерний процесс (рисунок 3). После этого процесс-родитель закрывает открытый для чтения конец канала, а дочерний процесс закрывает открытый на запись конец канала. Это дает возможность обеспечивать одностороннюю передачу данных между процессами, как показано на рисунке 4 [17].

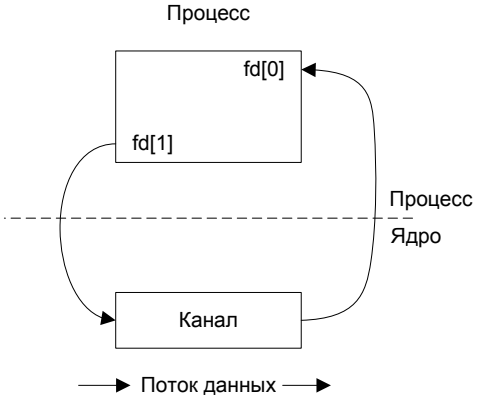

Рисунок 2 – Канал при занятии его единственным процессом

Именованные каналы (FIFO).

FIFO создается функцией mkfifo:

#include <sys/types.h>

#include <sys/stat.h>

int mkfifo(const char **pathname.* mode_t *mode):*

/* Возвращает 0 при успешном выполнении. -1 - при возникновении ошибок */

Здесь *pathname* – обычное для Unix полное имя файла, которое и будет именем FIFO.

Аргумент *mode* указывает битовую маску разрешений доступа к файлу, аналогично второму аргументу команды open.

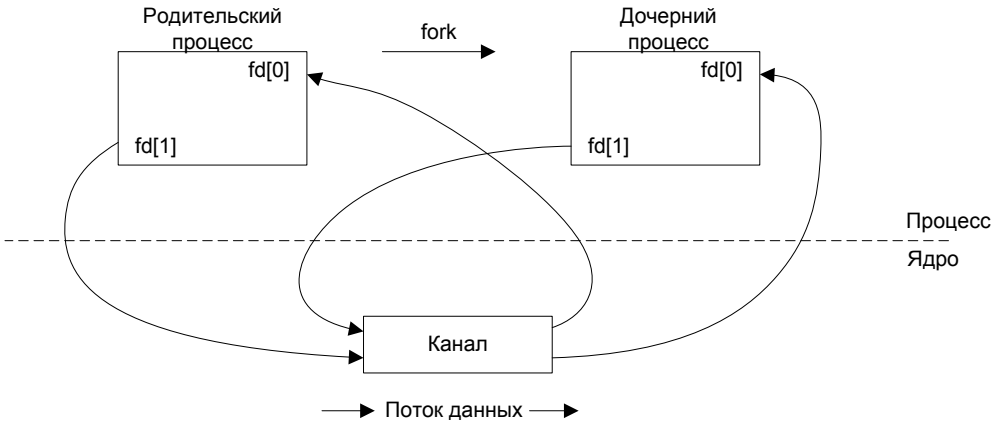

Рисунок 3 – Канал после активизации системного вызова fork

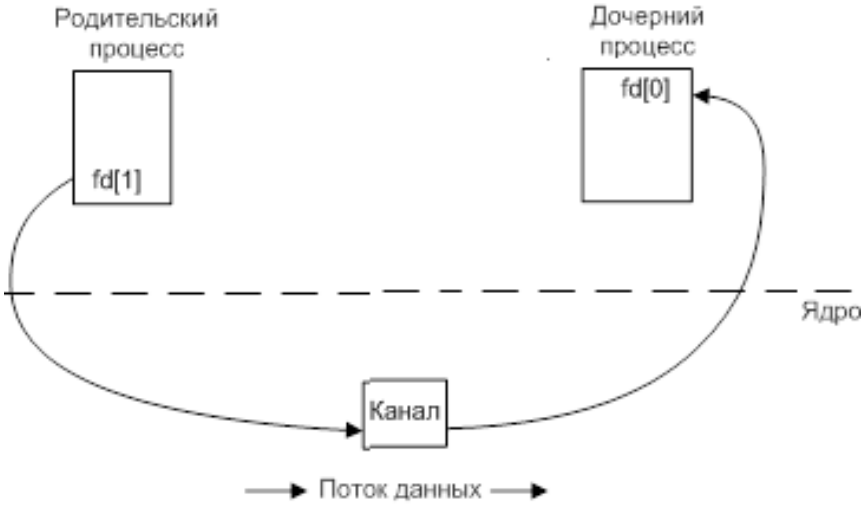

Рисунок 4 – Канал, разделяемый двумя процессами

Разделяемая (общая) память. Является наиболее быстрым средством межпроцессного взаимодействия.

Основной недостаток именованных и неименованных каналов состоит в том, что для передачи между процессами информация обязательно должна пройти через ядро (рисунок 5) [18,19].

Общая память обеспечивает возможность двум процессам обмениваться данными через общий участок памяти (рисунок 6).

На рисунке отражен тот факт, что данные копируются дважды: из исходного входного файла в общую память и наоборот, из памяти в результирующий выходной файл.

Для выделения участка общей памяти возможно применять POSIX функцию shm_open():

#include <sys/mman.h>

int shm_open(const char *name, int oflag, mode_t mode);

Общую память можно создать с помощью вызова функции mmap, основное отличие от shm_open() состоит в том факте, что память будет всё время оставаться выделенной до момента удаления или перезагрузки ЭВМ.

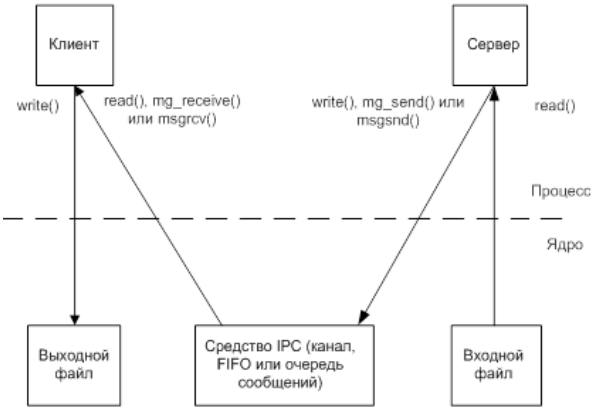

Рисунок 5 – Передача содержимого файла от сервера клиенту

Исследование программных средств измерения производительности каналов межпроцессного обмена. Рассмотренные ранее в статье методы межпроцессного обмена обладают разной производительностью, для измерения которой воспользовались программным пакетом lmbench, который весьма универсален и может работать на подавляющем большинстве ОС семейства Unix [20,21].

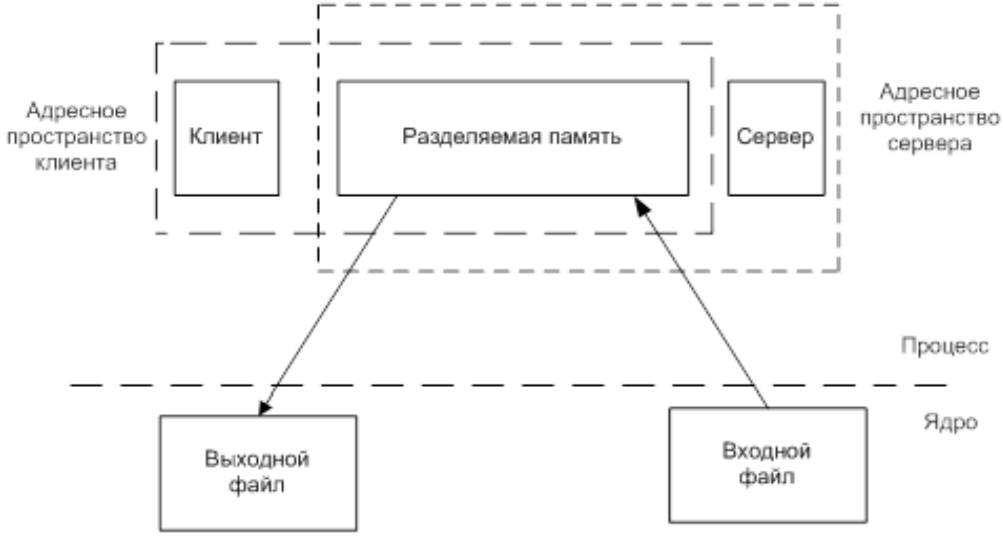

Рисунок 6 – Процесс копирования файла через общую память

Этот программный пакет предоставляет довольно богатые возможности по измерению различных низкоуровневых аспектов производительности ядра ОС [18].

Его тесты можно разбить на две большие группы: тесты пропускной способности и латентности.

В рамках работы прежде всего будут интересовать результаты следующих тестов: измерение полосы пропускания общей памяти при чтении; измерение полосы пропускания программных каналов; измерение времени задержки общей памяти; измерение времени задержки программных каналов.

Результаты тестирования. Тестирование проводилось на машине под управлением операционной системы FreeBSD 8.0.

Полоса пропускания общей памяти при чтении. Результаты измерения полосы пропускания представлены на рисунке 7.

Рисунок 7 – Результаты измерения полосы пропускания общей памяти

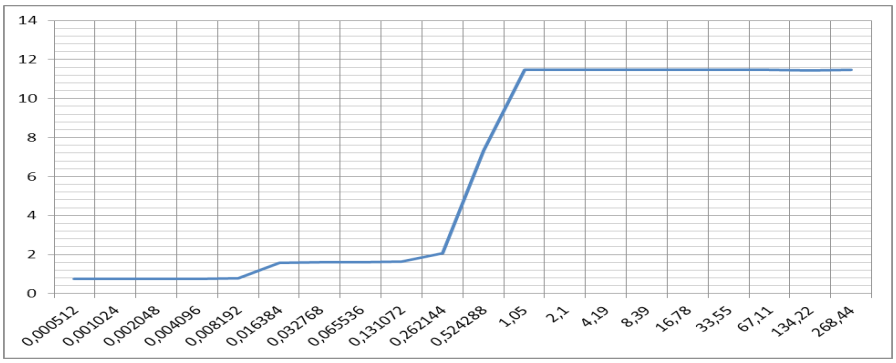

Рисунок 8 – Результат измерения времени задержки при работе общей памяти

По оси X отложен объем передаваемой информации в мегабайтах. По оси Y полоса пропускания в мегабайтах в секунду. Максимальное значение составляет 10433,46 МБ/сек. Среднее значение равно 5064,3 МБ/сек.

Полоса пропускания программных каналов. В отличие от тестов, измеряющих характеристики общей памяти, для программных каналов измеряется только среднее значение полосы пропускания. Оно равняется 1967,03 МБ/сек.

Временя задержки общей памяти. Результат измерения времени задержки при работе общей памяти приведен на рисунке 8. По оси X отложен объем передаваемой информации в мегабайтах. По оси Y время задержки в микросекундах. Минимальное значение составляет 0,761 микросекунды. Среднее значение 6,14445 микросекунды.

Временя задержки программных каналов. Так же, как и при измерении полосы пропускания вычисляется только среднее значение. Оно равняется 12,2546 микросекунд.

По итогам проведенного тестирования можно сделать вывод, что общая память является более быстрым способом передачи данных между процессами, при этом обладающий меньшими задержками.

Заключение. В работе были рассмотрены следующие классические задачи управления взаимодействующими процессами: «писатели-читатели», «производители-потребители». А также были представлены возможные их решения с использованием механизма мониторов. Также рассмотрены некоторые методы межпроцессного обмена на примере именованных и неименованных каналов и общей памяти.

В завершении работы было проведено сравнение каналов и общей памяти по двум параметрам: полоса пропускания и время задержки.

СПИСОК ЛИТЕРАТУРЫ:

1. Таненбаум Э., Бос Х. Современные операционные системы // 4-е издание, СПБ.: Питер, 2015. – 1120 с.

2. Вашкевич Н.П. Формализация алгоритма управления взаимодействующими параллельными процессами в задаче «производители-потребители» с использованием механизма мониторов / Вашкевич Н.П., Волчихин В.И., Бикташев Р.А. // Вопросы радиоэлектроники, серия ЭВТ. – 2010. – С.3–15.

3. Вашкевич Н.П., Бикташев Р.А. Достоинство использования механизма монитора и языка логики событийных недетерминированных автоматов на примере формализации алгоритма управления взаимодействующими процессами в задаче «Писатели-читатели» // Новые информационные технологии и системы: сборник научных статей XIII Международной научно-технической конференции. – 2016. – С. 78–80.

4. Вашкевич Н.П., Волчихин В.И., Бикташев Р.А. Автоматное представление алгоритмов управления параллельными процессами в задаче «Писатели – читатели» на основе концепции недетерминизма и механизма монитора // Известия высших учебных заведений. Поволжский регион. Технические науки. – 2015. – № 3 (35). – С. 65–76.

5. Стивенс У. UNIX: взаимодействие процессов. Пер. с англ. – СПб.: Питер, 2003. – 576 с.

6. Бикташев Р.А., Петкилев А.А., Балаев К.А. Разработка устройства синхронизации процессов для многопроцессорных систем // В сборнике: Современные технологии в науке и образовании – СТНО-2018 Сборник трудов международного научно-технического форума: в 11 томах. Под общ. ред. О.В. Миловзорова. – 2018. – С. 80–85.

7. Вашкевич Н.П., Бикташев Р.А., Синев М.П. Формализация алгоритмов управления многопоточным доступом к разделяемым ресурсам на основе использования событийных недетерминированных автоматов // Интеллект. Инновации. Инвестиции. – 2014. – № 1. – С. 128–133.

8. Мартышкин А.И. Математическое моделирование и оценка производительности средств синхронизации процессов в многопроцессорных системах // Наукові записки Міжнародного гуманітарного університету: статті учасників другої міжнародної мультидисциплінарної конференції. – 2016. – С. 151–157.

9. Мартышкин А.И., Карасева Е.А. Анализ процесса управления общими ресурсами в многопроцессорных системах // Информационные технологии. Радиоэлектроника. Телекоммуникации. – 2017. – № 7. – С. 368–371.

10. Мартышкин А.И., Карасева Е.А. Разработка и исследование математических моделей управления общими ресурсами в многопроцессорных системах // Модели, системы, сети в экономике, технике, природе и обществе. – 2017. – № 1 (21). – С. 150–158.

11. Мартышкин А.И. Математические модели семафоров для координации доступа к общим ресурсам многопроцессорных систем // Colloquium-journal. – 2018. – № 8-1 (19). – С. 36–39.

12. Петкилев А.А., Балаев К.А., Бикташев Р.А. Устройство управления ресурсами в многопроцессорной системе // Информационные технологии в науке и образовании. Проблемы и перспективы: Сборник научных статей Всероссийской межвузовской научно-практической конференции. Под редакцией Л.Р. Фионовой. – 2018. – С. 12–14.

13. Вашкевич Н.П., Бикташев Р.А. Структурный синтез устройства управления ресурсами в многопроцессорной системе // Оптико-электронные приборы и устройства в системах распознавания образов, обработки изображений и символьной информации. Распознавание – 2017: Сборник материалов XIII Международной научно-технической конференции. – 2017. – С. 101–103.

14. Бикташев Р.А. Модель управления процессами в многопроцессорных системах на основе стохастических сетей массового обслуживания // Новые

информационные технологии и системы: сборник научных статей XIV Международной научно-технической конференции, посвященной 70-летию кафедры «Вычислительная техника» и 30-летию кафедры «Системы автоматизированного проектирования». – 2017. – С. 17–20.

15. Бикташев Р.А., Вашкевич Н.П. Модели событийных недетерминированных автоматов для формального представления основных свойств систем управления параллельными процессами и ресурсами // Инфокоммуникационные технологии. – 2013. – Т. 11. – № 3. – С. 95–98.

16. Волчихин В.И., Вашкевич Н.П., Бикташев Р.А. Модели событийных недетерминированных автоматов представления алгоритмов управления взаимодействующими процессами в многопроцессорных вычислительных системах на основе использования механизма монитора // Известия высших учебных заведений. Поволжский регион. Технические науки. – 2013. – № 2 (26). – С. 5–14.

17. Мартышкин А.И., Мартенс-Атюшев Д.С., Полетаев Д.А. Методы планирования и модели для оценки производительности средств синхронизации взаимодействующих процессов в параллельных вычислительных системах // Современные социально-экономические процессы: проблемы, закономерности, перспективы: Сборник статей III Международной научно-практической конференции: в 2 частях. – 2017. – С. 37–40.

18. Стивенс У.Р. UNIX: разработка сетевых приложений. – СПб.: Питер, 2007. – 1039 с.

19. Родригес К.З., Фишер Г., Смолски С. Linux. Азбука ядра. – М.: Кудиц-пресс. – 2007. – 584 с.

20. Документация к программному комплексу lmbench // lmbench man pages URL: http://lmbench.sourceforge.net/man/index.html (дата обращения: 25.06.2019).

21. Эндрюс Г.Р. Основы многопоточного, параллельного и распределенного программирования: Пер. с англ. – М.: Издательский дом «Вильямс». – 2003. – 512 с.

Статья публикуется при поддержке стипендии Президента РФ молодым ученым и аспирантам на 2018-2020 гг. (СП-68.2018.5).

Статья поступила в редакцию 01.07.2019
Статья принята к публикации 20.09.2019