DEPÓSITO LEGAL ppi 201502ZU4666
Esta publicación científica en formato digital es continuidad de la revista impresa
ISSN 0041-8811
DEPÓSITO LEGAL pp 76-654

Revista de la Universidad del Zulia

Fundada en 1947 por el Dr. Jesús Enrique Lossada

Ciencias Exactas Naturales y de la Salud

Año 10 N° 27
Mayo - Agosto 2019
Tercera Época
Maracaibo-Venezuela

Software complex for measuring operating systems' main functions performance

Alexey I. Martyshkin *

ABSTRACT

This paper describes the software package created by the author for measuring the performance of operating system functions. This work aims to synthesize a software package designed to analyse the execution time of software functions of uniprocessor and multiprocessor operating systems. The package analyses a number of functions of operating systems: mutexes, semaphores, read-write locks, FIFO and PIPE channels, TCP and UDP sockets, context switching, system calls. Unlike analogues, this package is represented by a convenient graphical user interface; the Qt and Qwt libraries are used for its implementation, they have a rich set of widgets, i.e. programs that facilitate access to information. The libraries used are cross-platform, which allows us to make a simpler procedure for transferring applications to different operating systems; smoothing the test results curves is used. In conclusion, the results of test measurements are presented.

KEYWORDS: Linux, performance measurement, operating system functions, QT and QWT libraries, C ++ language.

Paquete de software para medir el rendimiento de las funciones principales de los sistemas operativos

RESUMEN

Este documento describe el paquete de software creado por el autor para medir el rendimiento de las funciones del sistema operativo. Este trabajo tiene como objetivo sintetizar un paquete de software diseñado para analizar el tiempo de ejecución de las funciones de software de los sistemas operativos uniprocesador y multiprocesador. El paquete analiza una serie de funciones de los sistemas operativos: mutexes, semáforos, bloqueos de lectura y escritura, canales FIFO y PIPE, zócalos TCP y UDP, cambio de contexto, llamadas al sistema. A diferencia de los análogos, este paquete está representado por una conveniente interfaz gráfica de usuario; las bibliotecas Qt y Qwt se utilizan para su implementación, tienen un amplio conjunto de widgets, es decir, programas que facilitan el acceso a la información. Las bibliotecas utilizadas son multiplataforma, lo que nos permite hacer un procedimiento más simple para transferir aplicaciones a diferentes sistemas operativos; se utiliza el suavizado de las curvas de resultados de la prueba. En conclusión, se presentan los resultados de las mediciones de prueba.

PALABRAS CLAVE: Linux, medición de rendimiento, funciones del sistema operativo, bibliotecas QT y QWT, lenguaje C ++.

*Candidate of Technical Sciences, Associate Professor, Computational Machines and Systems Department, Penza State Technological University (440039, Russia, Penza, 1/11 Baydukova passage / Gagarina st., 1/11 e-mail: alexey314@yandex.ru)

Recibido: 30/07/2019 Aceptado: 11/09/2019

Introduction

The goal was set to develop a software package designed to analyse the execution times for software functions of uniprocessor and multiprocessor operating systems. The performance measurement for various functions of an operating system is necessary not only for the operating system developers to evaluate the effectiveness of the implementation of a particular function and to compare the performance of its implementation in other operating systems, but also for application programmers (Biktashev and Martyshkin, 2013), who need to measure the performance of the operating system functions to select the most suitable implementation for their projects (especially for large software systems projects) (Ivanov et al., 2017; Skvortsov and Pyurova, 2016). Implementations of the same operating system functions can significantly differ in different operating systems both in performance and in the ways of working with them (Duplenko, 2013 a,b). For this reason, a developer can select the most preferred tools for development in a specific operating system, or vice versa - choose an operating system in which the implementation of these functions most optimally meets the requirements of the designed software.

1. Formulation of the problem

The software package should perform testing of operating system functions, such as:

- Means for synchronizing processes (threads): mutual exceptions (mutexes), semaphores, read-write locks;

- Means of interprocess exchange: named FIFO channels, unnamed PIPE channels, TCP and UDP sockets;

- Context switching;

- System calls (read, write, open, close).

Currently, the software package Lmbench is widely used, which allows testing nix-compatible systems, incorporating a wide range of tests for various operating system subsystems (Stevens, 2003, 2007). The Lmbench composition includes tests for determining the following characteristics:

a) interprocess exchange means; b) means for synchronizing processes (threads); c) system calls; d) context switching.

This package allows its user to get a fairly holistic view of the tested operating system, to identify the strengths and weaknesses of the implementation of specific functions. But, on the other hand, the Lmbench package does not have a graphical interface and each type of testing is presented by a separate program that must be launched from the command line with some arguments passed.

From the foregoing, it is clear that the LMbench package has a number of drawbacks that make working with it inconvenient, requiring repeated repetition of the same actions and additional calculations to obtain results. Repeatedly invoking the testing program also makes the testing procedure with Lmbench a cumbersome, time-consuming procedure.
Thus, the software package developed for testing the operating systems' functions makes the testing work more convenient, more productive, and affordable than the LMbench software package.

For the development of a software package, tasks that are supposed to be solved with its help are specified: 1) Selection of the tested functions; 2) Collection of test results; 3) Processing the results; 4) Displaying the received statistics in the form of graphs; 5) Displaying the results in a separate window; 6) Colour settings for displaying data; 7) Smoothing charts. Analysing the set tasks, we conclude about the input and output data of the developed software package. Input data are: a) name of the tested function; b) a colour palette for displaying results; Output data are: a) Graphs of the results obtained; b) Display of results that do not require presentation in the form of graphs.

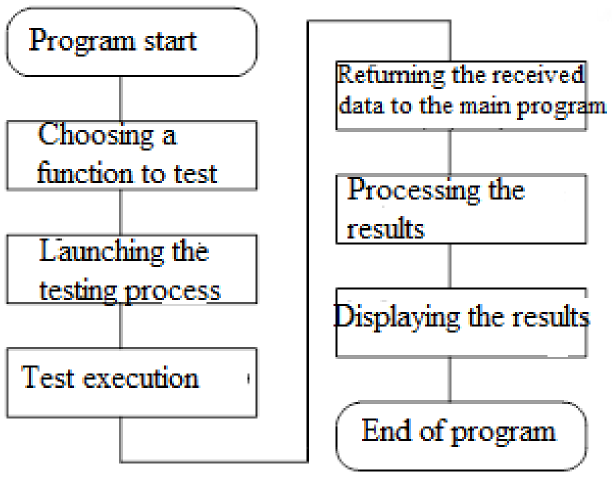

Program start
Choosing a function to test
Launching the testing process
Test execution
Returning the received data to the main program
Processing the results
Displaying the results
End of program

Fig. 1. The operation algorithm of the developed software package

The objectives of the study are:

- to determine the composition of U-functions (the software function of the operating system

kernel associated with any system call) that implement the network and transport levels;

- to measure the execution time of each U-function in the process of transmitting data blocks;

- to determine the probabilistic characteristics of the call and processing of U-functions;

- to determine the dependences of the probability-time characteristics that determine the computation system performance depending on the traffic parameters and the characteristics of network devices.

Network protocols are implemented programmatically in the operating system kernel as a set of U-functions.

Since these protocols are implemented in software, the performance of the network and transport layers of the OSI network model can be measured using software tools. To do this, we can use software monitors (Martyshkin, 2016a; Martyshkin et al, 2018a).

Monitors of the tracer type (tracers) are programs that record the specified parameters of the computing process at specific points. Each such point corresponds to a specific event in the work of the computation system, and when it passes, a transmission is organized to the measurement monitor programs that collect, accumulate and display measurement information.

2. Mathematical model of work with semaphores

Semaphores are used to coordinate the use of single or fixed sets of resources by several computational processes. The semaphore performance problem is that when processes interact, there are requirements for access to shared resources that lead to a collision of transactions, as they conflict with each other. Conflicts lead to loss of operating system performance (Martyshkin, 2018, 2016b). This is most characteristic in parallel and multiprogram systems when interacting processes are implemented in independent processors which may require a common resource at the same time. If the resource is required by too many processes, then they are queued. At the same time, requests are satisfied on the basis of: "first in, first out" principle (FIFO) (Martyshkin et al, 2018b; Tanenbaum and Bos, 2015).

We assume that a computer system comprises a common resource available to the set of processes executed in the nodes and n –processor nodes protected by a semaphore S. An analytical model of an n-processor system with a single shared resource for evaluating performance losses due to conflicts over access to the semaphore, using such scheduling concept as FIFO, is shown in Fig. 2 a. The model is presented as an open stochastic queuing network consisting of n (S_1,. .., S_n) queuing systems simulating processor nodes and a single-channel queuing system (S_n +1) which simulates a semaphore. A thread of requests for the execution of processes with intensity $\lambda_0 = 1/T$ arrives at the input of the n-processor system where T is the average duration of the interval between incoming requests (Martyshkin, et al., 2018a).

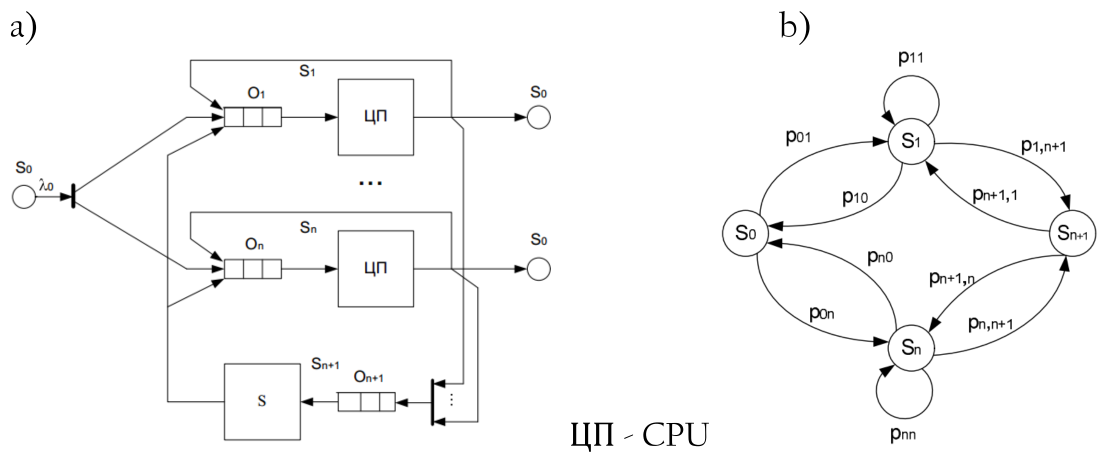


Fig. 2. Diagram of the analytical model of the n-processor system (a) and its flow graph (b)

The request flow is distributed by the preliminary scheduler to the processor nodes with probabilities p_{01},. .., p_{0n}, presented in the form of a probability flow graph for the stochastic network shown in Fig. 2, b. We suppose that the flows of requests for the execution of processes at the input of a multiprocessor system are distributed equally between processor nodes, i.e. $p_{01} = \ldots = p_{0n} = 1/n$. Requests that have been served in the semaphore are an equally likely return to the processor nodes for continued service, therefore, $p_{n+1,1} = \ldots = p_{n+1,n} = 1/n$.

The waiting time for a request in the network is estimated by

$$T_w = \alpha_1 t_{w1} + \alpha_2 t_{w2} + \ldots + \alpha_n t_{wn} + \alpha_{n+1} t_{wn+1}, \quad (1)$$

Where $\alpha_i = \lambda_i / \lambda_0$ is the network transmission coefficient (i = 1,. .., n +1); t_{wi} is the waiting time in the i-th queuing system. The intensity of request flows is determined by the system of equations: $\lambda_i = \sum_{i=0}^{n} p_{ij} \lambda_j$, where p_{ij} is the probability of transmission from the queuing system S_i in the queuing system S_j; i, j = 0,1,. .., n + 1

Priority-based scheduling yields almost twice advantage as to waiting time in a queue for a semaphore than using a FIFO- based strategy. The obtained models allow us to make quantitative estimates of the waiting time of processes accessing a shared resource through a semaphore. The models can be used in the designing of parallel operating systems, where the execution time of processes is critical (Martyshkin, 2018).

3. Architecture of the software and hardware package

Work with software package was carried out on a computer running an operating system of the Linux family. To ensure the operability of the program, a nix-compatible operating system with installed Qt and Qwt libraries is required. Having analysed the data on the advantages and disadvantages of modern programming languages, we can conclude that it is most advisable to use the C ++ language to solve the problems posed for a number of the following reasons: support for the paradigm of object-oriented programming, high performance of final programs, etc.

The data structure of the developed software package is shown in Fig. 3. The scheme of the main program algorithm is shown in Fig.4.

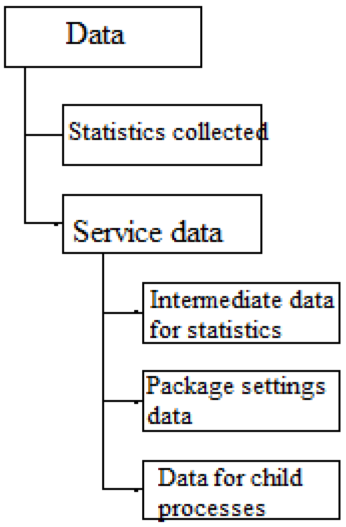


Fig. 3. Data structure of the *software package* under development

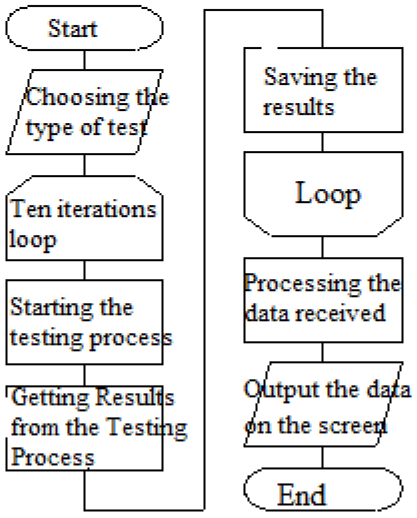

Start
Choosing the type of test
Ten iterations loop
Starting the testing process
Getting Results from the Testing Process
Saving the results
Loop
Processing the data received
Output the data on the screen
End

Fig. 4. The scheme of the main program algorithm

4. Program development

In accordance with RAD technology, we outline three stages in the development of the software package:

1) development of a graphical interface for user interaction with the program; 2) the
2) development of functional modules that solve the tasks specified in the Development Task;

3) binding the implementation of the functional part to the graphical interface.

The basis of the future user interface is a multi-window interface. Separate windows are created for various problems to be solved, providing an unloaded interface that is necessary for their solution (Rodríguez et al., 2007; Williams, 2014; Hughes and Hughes, 2004).

In the developed software package, the curve smoothing (filtering) is used. Smoothing is a technology used to eliminate the "jagged" effect that occurs at the edges of a multitude of 2D or 3D images that are displayed separately from one another at the same time. Almost all normal graphics are offered in two forms: raw and smooth. A graph can be built in a straightforward manner from message to message, why it will have angular outlines. But if smoothing is selected, then the graph will have a smoother outline. The smoothing level and algorithm are the same for all graphs. Smoothing is performed using several consecutive data, and their number is usually selected experimentally.

Figs. 5-6 show the graphs of test results (semaphore and context switching).

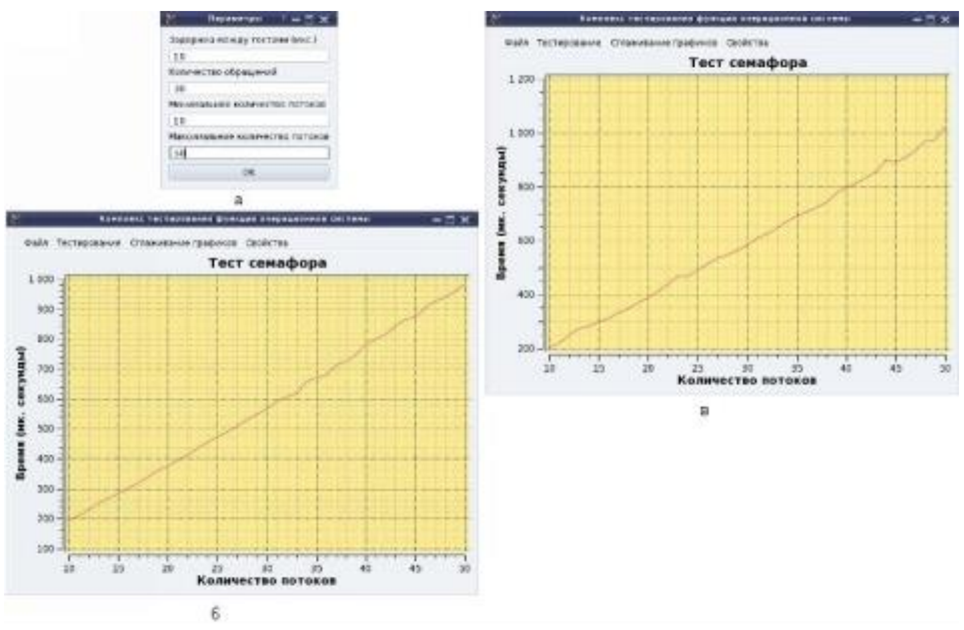
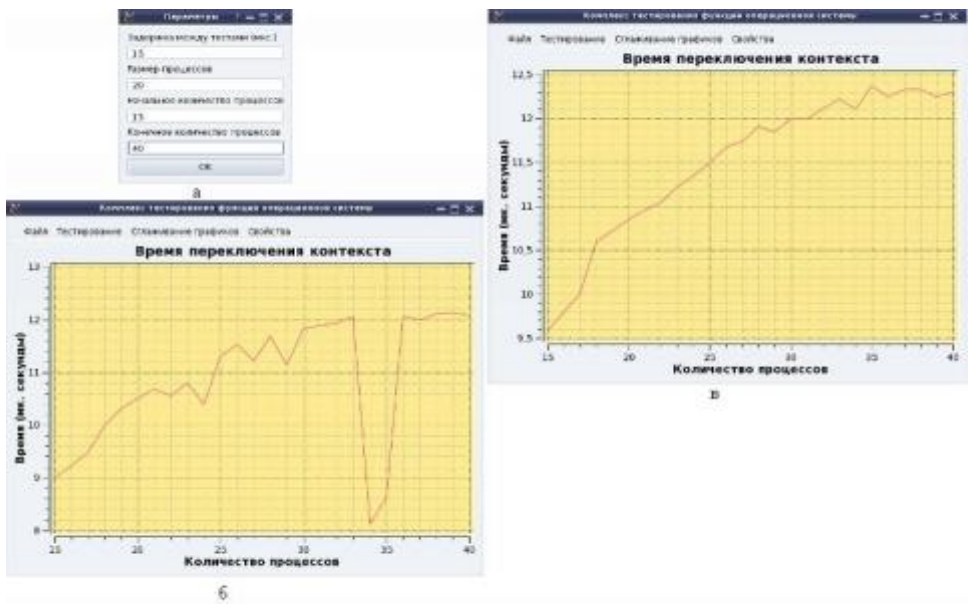

Параметры
Задержка между тестами (мкс)
15
Количество обращений
30
Минимальное количество потоков
10
Максимальное количество потоков
50
OK
а
Комплекс тестирования функций операционной системы
Файл Тестирование Сглаживание графиков Свойства
Тест семафора
Время (мк. секунды)
Количество потоков
б
в

Fig. 5. Semaphore test: setting parameters (a), without smoothing (b), and using the most probable smoothing (c). Fig. 6. Context switching time test: setting parameters (a) without smoothing (b) and using the most probable smoothing (c)


Параметры
Задержка между тестами (мкс)
15
Размер процессов
20
Начальное количество процессов
15
Конечное количество процессов
40
OK
а
Комплекс тестирования функций операционной системы
Файл Тестирование Сглаживание графиков Свойства
Время переключения контекста
Время (мк. секунды)
Количество процессов
б
в

The software package developed is registered with the Federal Service for Intellectual Property.

Conclusions

The software package was developed to test the operating system functions: mutexes, semaphores, read-write locks, FIFO and PIPE channels, TCP and UDP sockets, context switching, system calls (read, write, open, close). After selecting the type of testing, the software package allows us to automatically collect the necessary statistics, process it and display the processing results in a user-friendly form.

Unlike analogues, this package is represented by a convenient graphical user interface, for the implementation of which the Qt and Qwt libraries are used. These libraries have a rich set of widgets, i.e. programs that facilitate access to information, with which we can create a complex graphical interface. The libraries used are cross-platform, which makes it easier to transfer requests to different operating systems.

Acknowledgements

The paper is published with the support of the scholarship granted by the President of the Russian Federation to young scientists and graduate students for 2018-2020. (SP-68.2018.5).

References

Biktashev R.A., Martyshkin A.I. (2013). A set of programs for measuring the performance of operating system functions // XXI Century: Results of the past and problems of the present plus. 2013. No. 10 (14). Pp. 190–197.

Duplenko A.G. (2013a). Comparative analysis of data sorting algorithms in arrays // Young scientist. 2013. No. 8. Pp. 50-53.

Duplenko A.G. (2013b). Evolution of methods and algorithms for sorting data in arrays // Young Scientist. 2013. No. 9. Pp. 17-19.

Hughes C., Hughes T. (2004). Parallel and distributed programming using C ++. 2004.672 p.

Ivanov K.K., Razdobudko S.A., Kovalev R.I. (2017). Parallel sorting methods // Young scientist. 2017. Number 7. Pp. 15-16.

Martyshkin A. I. (2016a). Modern methods of measuring the performance of multicore computing systems // New Information Technologies and Systems: Collection of scientific papers of the XIII International Scientific and Technical Conference. 2016. Pp. 128 - 131.

Martyshkin A. I. (2016b). Semaphore performance problems in parallel systems // Collection of papers of the International scientific-practical conference: in 3 parts. 2016. Pp. 81-83.

Martyshkin A. I. (2018). Mathematical models of semaphores for coordinating access to shared resources of multiprocessor systems // Colloquium-journal.No. 8 (19). part 1. 2018. Pp. 36-39.

Martyshkin A.I., Markin E.I., Tereshkin D.O., Razdobudov S.A. (2018a). Analytical models for evaluating the performance of semaphores of multiprocessor systems // Collection of papers of the XII International Scientific and Practical Conference: in 2 parts. 2018. Pp. 78-81.

Martyshkin A.I., Markin E.I., Zotkina A.A., Razdobudov S.A. (2018b). Modern methods of measuring the performance of multi-core systems with HT technology // collection of papers of the XXV International scientific and practical conference.2018. Pp. 50-52.

Razdobudov S. A., Salnikov I. I. (2018). Research and analysis of the possibility of multithreading when developing software in C ++ // International Student Scientific Herald.2018. No. 3-2. Pp. 334-337.

Rodriguez C.Z., Fisher G., Smolsky S. (2007). Linux. ABC core. - Kudits Press, Moscow, 2007.584 p.

Skvortsov S.V., Pyurova T.A. (2016). Parallel data sorting algorithms and their implementation on the CUDA platform // Vestnik RGRTU. 2016. No. 58. Pp.42-48.

Stevens U.R. (2007). UNIX : developing network requests. Peter, 2007.1039 p .

Stevens U.R. (2003). UNIX : process interconnection. Peter, 2003.576p.

Tanenbaum E., Bos H. (2015). Modern Operating Systems. 4th ed. SPb. : Peter, 2015.1120 p. : ill. (Series "Classic computer science").

Williams E. (2014). Parallel programming in C ++ in action. The practice of developing multi-threaded programs. DMK Press, 2014, 672 p.