Creating an Operating System for Video Camera Chips in an Intelligent Video Surveillance System

Alexey Subbotin[1](✉), Nataly Zhukova[2], and Elena Stankova[3]

[1] Saint-Petersburg State Electrotechnical University, Professor Popov Street 5, 197376 St. Petersburg, Russia
alesu1543@gmail.com
[2] St. Petersburg Institute of Informatics and Automation of the Federal Research Center at the Russian Academy of Sciences (FRC RAS), 14th Line V.O. 39, 199178 St. Petersburg, Russia
[3] St. Petersburg State University, 7-9, Universitetskaya Nab., 199034 St. Petersburg, Russia

Abstract. This article discusses a method for creating an operating system for devices with limited performance, specifically video camera chips used in intelligent video surveillance systems. Analogs are considered: general-purpose operating systems and special-purpose operating systems for devices with limited computing resources. The problem of fast launching of an operating system on process computers in devices is identified. Applications in an operating system can run faster if the operating system itself is designed for a single task, taking into account the architectural features of the embedded processor in the chips of a technical device, such as a video camera. Deep integration of the control program, file system, and scripting language in the operating system kernel is possible. The kernel of such a system runs faster for intelligent video surveillance systems. A solution is proposed, involving development of a custom operating system for video camera chips with limited computing resources. The new operating system runs and launches applications 67% faster in real time mode and boots twice faster than similar systems. The proposed method for increasing the speed of program execution in the operating system environment and for quickly launching the new operating system on video surveillance devices can also be applied in other devices, such as routers, tablets, and embedded computers for a wide range of applications.

Keywords: Operating systems · Technological computers · Video camera chips · Intelligent video surveillance systems

1 Introduction

An operating system is essential for any computer. Starting with the Multics operating system (https://www.multicians.org/), all operating systems provide a basic set of input/output operations and are an extension of the BIOS (Basic Input/Output System). A BIOS operates only with the machine, while an operating system is focused on a

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2027
O. Gervasi et al. (Eds.): ICCSA 2026 Workshops, LNCS 16763, pp. 511–523, 2027.
https://doi.org/10.1007/978-3-032-30530-5_32

specific computer architecture, particularly the central processing unit (CPU). Embedded operating systems (OS) for devices are quite common and not something specific. Most embedded OS do not have user-facing capabilities at the kernel level; they are resource managers and task schedulers. New devices are added to the operating system's address space. In addition to working with devices, the operating system divides RAM for shared use between applications. Each operating system contains a basic set of utilities and commands for working with devices and applications. Interrupts can be both hardware and software.

The main task of an operating system is to delimit memory (stack in CPU, heap in RAM, disk in hard drive) and assign access rights. It can be said that the operating system is a RAM manager and utilities [1]. Each OS executes application code with additional information contained in the header. Application headers can be of various formats: PAGEZERO for MacOS, MZ for Microsoft Windows, ELF for Linux. The bytecode of different applications runs concurrently in the OS and should not interfere with each other. It is possible to read the RAM of all applications, but in some cases (like *BSD), reading the memory of other applications is difficult or even impossible (with hardware memory encryption).

1.1 History of Operating Systems

The original function of all operating systems was to redirect input/output, while simultaneously performing arbitrary operations on the data. The concept of redirecting data streams from one device (the transmitter) to another (the receiver) has been preserved in all computers and operating systems (Fig. 1). The Multics operating system, with its graphical interface (a screen divided into four sections with a menu at the top), was designed for US missile defense. Multics also worked for AT&T, which provided telephone and telegraph services to the US population. Conversations were recorded on reels, and Multics commands helped US officers to extract necessary conversation fragments (files) into a spooler (a separate storage reel with audio interfaces on a GE-645 computer running Multics). The principle of file transfer and data redirection was inherited by the UNIX operating system, which source code is ten times less than the source code of Multics.

Despite the importance of a computer (or mainframe) operating system, creating your own isn't all that difficult. The popular Minix OS is a UNIX clone, the volume of the source code of which is about a half of a book volume (1,200 pages). However, Minix is full-fledged operating system. Linux was created as a Minix clone with added networking features.

1.2 Chips in Video Cameras

It's difficult to impress anyone with grain-sized chips these days, since everyone uses bank cards, and the chip on the card is a fully-fledged computer. However, a computer inside a device (for example, a video camera) can significantly outperform bank chips. It may have relatively large RAM and ROM, a good processor with integrated graphics functions [20, 21]. Video can be processed not only in the cloud or on a fog video

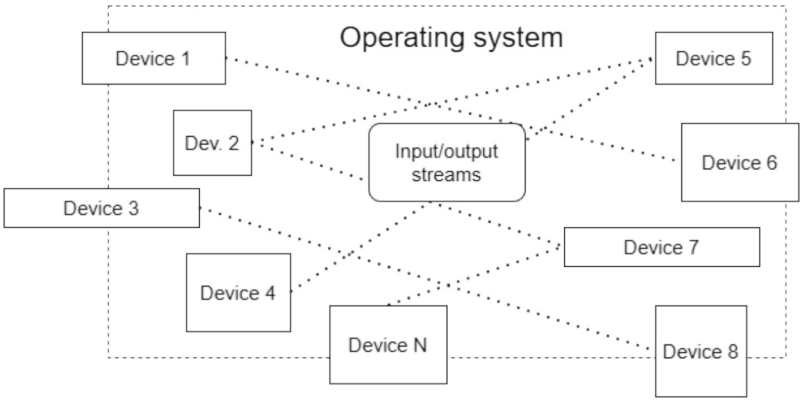

Fig. 1. The principle of input-output redirection in the OS.

server, but also directly on the camera's process computer using an integrated system-on-board (SoC) processor. ARTPEC processors for Axis Communications come in various series and modifications (chip generations), which differ in capabilities and performance (depending on the functions they can perform for video cameras). Built-in functions include color correction, haze removal, night mode, clarity, brightness and contrast adjustment, and many other features supported in explosion-proof and waterproof Axis video cameras.

2 Overview of General-Purpose Operating Systems

A popular operating system for embedded industrial computers is Linux (https://kernel.org/), which is a monolithic kernel with device drivers, utilities, and file systems. Various Linux-based OS builds are available, including Ubuntu, Astra, ALT, SUSE, and many others, differing primarily in their application sets. Linux supports basic UNIX commands (including those inherited from Multics), allowing it to run the simple X-Window windowing system (https://x.org/) and the more complex KDE (https://kde.org/) or Gnome (https://www.gnome.org/) on top, which contain a large number of additional applications and are more demanding on the graphics adapter (GPU). In the absence of a powerful graphics adapter, for example, in an industrial PC, it is recommended to use the basic X-Window windowing system or the command line. Linux supports various processor (CPU) architectures, a feature that no other OS can provide. At the time of writing, Linux supports the following processors: ARM, x86, PowerPC, RiscV, MIPS, and SuperH [18].

The most popular application operating system is Microsoft Windows (https://support.microsoft.com/ru-ru/windows), thanks to its simplicity and accessibility (it's affordable, and in some cases, Microsoft Windows comes pre-installed almost free). Windows has a robust API (Application Programming Interface) with high-quality documentation for developing applications. Windows supports threaded processes with priorities, allowing multiple programs from the steak and heap to run simultaneously. Priorities allow

for a real-time effect, where a program effectively takes over all processor resources and provides almost instantaneous response times, which is commonly considered real-time.

The macOS family from Apple (https://www.apple.com/os/macos/) accounts for a large amount of desktop and smartphone operating systems. Mac OS versions for PCs and iPhones were not quite stable, but after the FreeBSD kernel was implemented with permission from the developers, the macOS family of operating systems became extremely reliable. This OS line is very popular in Apple devices (smartphones, laptops, smartwatches, tablets, game consoles, professional video cameras, and televisions) due to Apple's licensing policy, which almost never sells macOS to non-Apple device makers [3, 4].

The *BSD OS line from the University of California, Berkeley (https://www.freebsd.org/) is considered the most stable OS. Various modifications exist: FreeBSD with support of Linux applications; NetBSD for routers; OpenBSD with strict licensing requirements for application development. Memory encryption makes it virtually impossible to modify other programs. A FreeBSD application runs only in its own memory, allocated by the OS. If the application crashes, the FreeBSD kernel restarts the application. This heap memory policy creates significant challenges for application developers, but the application will operate reliably and with high stability [15–17].

There are operating systems that are not widely used (Chrome OS, ReactOS, Haiku, OpenSolaris), but are designed for specific devices in the interests of individual corporations. The UNIX-like operating system OpenSolaris v.134 (http://www.opensolaris.com/) from Oracle has its own kernel and graphical interface (MilaX, marTux, Nexenta, etc.) developed for SQL and NoSQL data servers. OpenSolaris has its own DBMS, OracleDB (https://www.oracle.com/database/). OpenSolaris is also used on backend servers with programming languages: Java, Perl, PHP from the Web Stack package with MySQL, and Apache. Google's Chrome OS v.140.0.7339.201 (https://chromeos.google/) is used for tablets running the Chromium-based Google Chrome browser. Chrome OS supports ARM or x86 processors. Chrome OS is primarily installed on tablets such as the Google Pixel Slate, Asus Chromebook Tablet CT100, Acer Chromebook Tab 10, HP Chromebook X2, and Samsung Galaxy Chromebook 2.

Comparison of operating system capabilities is presented in Table 1. The systems are compared based on the following capabilities: p1 – operation under different processor types; p2 – reliable operation of the RAM manager (heap); p3 – applicability (or version availability) for technological embedded computers; p4 – reliable separation of user rights; p5 – configuration and editing of the kernel source code; p6 – implementation of process threads for the real processor operation mode (the response time of the first process is less than 4 ms); p7 – the ability to quickly start the OS without checking devices and loading drivers for each device in the PC; p8 – compilation and execution of binary code of applications from other OS; p9 – support for UNIX, Multics commands and the POSIX standard in general for executing the X-Window program; p10 – support for command line utilities inherited from Multics and UNIX; p11 – the ability to disable the multi-user mode of the OS and use the auto login function to speed up loading the desktop (graphical interface in the OS).

Table 1. Comparison of the features and limitations of the considered operating systems.

Name of the system	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	p11
Linux	+	-	+	+	+	+	-	+	+	+	+
Windows	-	-	+	+	-	+	+	-	-	-	+
macOS	-	-	+	+	-	-	+	+	-	+	+
FreeBSD	-	+	+	+	+	-	-	+	+	+	+
ReactOS	-	-	-	-	-	-	-	-	n/a	-	n/a
Haiku	-	-	-	-	-	-	-	-	n/a	-	-
OpenSolaris	-	-	-	-	-	-	-	-	+	+	+
QNX	-	+	+	+	-	+	-	+	+	+	+

According to the results of analyzes of existing systems (Linux, Windows, macOS, FreeBSD, ReactOS, Haiku, OpenSolaris, QNX) there is no system that supports all the required features.

3 Overview of Operating Systems for Devices

Embedded Linux (https://ubuntu.com/embedded) is a very popular operating system for chips, embedded devices, and industrial computers. It does not have the X-Window graphical shell and powerful KDE or Gnome window systems [19]. This version of Linux contains only three components: Bootloader (boot tools), Kernel (memory manager for applications, device drivers), and RootFS (a set of file systems and network utilities). Embedded Linux compilation and build tools (Toolchain, Yocto, Buildroot) can be installed separately [6].

QNX (http://qnx.com/) is designed for embedded systems and operates in real-time. This means that the first application launched in QNX responds within milliseconds, while other applications experience significant lag. QNX is used in train control systems and other systems that require guaranteed, near-real-time response. According to official documentation, real-time mode in QNX 6.3 is defined as a guaranteed application response time of up to 4 ms when using a processor with a frequency of 40 MHz or higher [10, 11].

Windows IoT or Windows Embedded OS (https://developer.microsoft.com/en-us/windows/iot/) is for embedded technological (specialized) devices: cash registers, self-service kiosks, ATMs, payment systems, and other technological and industrial devices. For example, for the ICOP Vortex 86DX computer, which is a SoC (system on a chip) [13].

NetBSD (https://netbsd.org/) has proven itself as an OS for routers and network devices. It is highly reliable and built on the BSD kernel, excluding large software modules that are unnecessary for technological computers based on chips for industrial devices [12]. MacOS (https://support.apple.com/en-us/macos) is used in IoT devices certified by Apple [14].

The presented operating systems are compared in Table 2. The operating systems are compared based on the following features: p1 – support of mobile devices and smartphones; p2 – support of Internet of Things technology; p3 – support of industrial devices and production control; p4 – support of X-Window for running applications in the KDE/Gnome graphical environment or browser; p5 – kernel customization for use in devices with limited CPU and GPU; p6 – applicability for intelligent video surveillance system video camera chips; p7 – availability of implemented protocols for working with video servers.

Table 2. Comparison of the features and limitations of the considered systems.

Name of the system	p1	p2	p3	p4	p5	p6	p7
Embedded Linux	+	n/a	+	+	+	+	+
QNX	+	-	+	+	-	-	-
Windows IoT	+	+	-	+	-	+	+
NetBSD	-	-	+	-	-	-	-
macOS	+	+	-	-	-	n/a	+

According to the results of analyzes of existing systems (Embedded Linux, QNX, Windows IoT, NetBSD, macOS) there is no system that supports all the required features.

4 Operating System Problems

After loading the file system driver, the OS checks the PC's devices. A timeout (device response time) is specified for each device type in the OS configuration files. The OS sends an initialization signal each time, obtains the series and model of the device (video adapter, network card, card reader, etc.), and checks for drivers (from the heap if it's Linux; from disk if it's QNX or Windows). If the required driver is not found, it displays an unknown device message [7]. After launching the application in some cases bytecode translation is required for a different processor (for example, Linux kernel was originally designed for the x86 processor architecture, so is requires bytecode translation) (Fig. 2).

It's possible to create a specialized OS for video camera chips that will run applications faster and boot up faster. The slow boot problem can be solved using Puppy Linux 9.5 (http://www.puppylinux.com/), which bypasses device checks and immediately opens the desktop with a window system. If a device fails, the OS user will find it out during the OS's operation.

Older motherboards and devices built before 2001 lacked the ability to continuously supply power. These devices used AT power supplies, not ATX (Advanced Technology Extended). The motherboard and the entire computer or device (camcorder or embedded computer) were not powered after a power outage. After powering on, each device

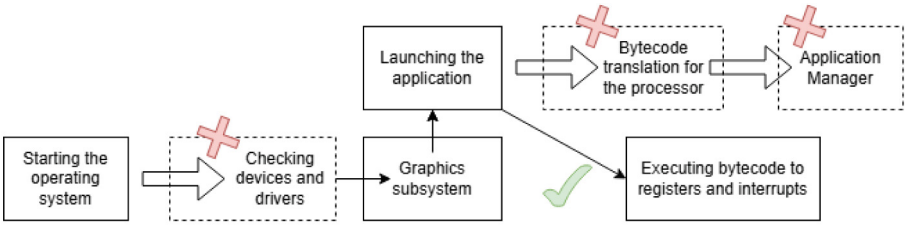

Fig. 2. Visual representation of OS problems.

received a [READY] command from the OS, which took time. Then hibernation mode was introduced, eliminating the need to poll devices. Now, there's no need to put the computer's devices into a ready state, since they were already polled when the power was turned on. If a device fails, a message will be displayed for the operating time of the computer or camcorder.

The problem of slow program execution in the OS can be solved by not converting applications in RAM, as in Linux, but executing them directly, as in FreeBSD. Therefore, binary emulation of Linux programs (ELFs) runs faster in FreeBSD than on the native 7.0.3 kernel (rsync://rsync.kernel.org/pub/), which both support the POSIX (Portable Operating System Interface) standard, which consists of basic utilities and commands. Many commands are inherited by the IEEE POSIX Certification standard (https://posix.opengroup.org/) from Multics and UNIX. In Linux, the commands are contained in the kernel itself with drivers. For example, in version 3.2.0, at startup, the 4.7 MB file "vmlinuz-3.2.0–53-generic" is read from the "/boot/" directory and written to a virtual disk in RAM. The directory also contains additional utilities with the prefixes: "abi-Ver.", "config-Ver.", "initrd.img-Ver.", "System.map-Ver." A 14 MB version 3.2.0 file, but the Linux kernel is still considered monolithic. The QNX4 operating system (https://blackberry.qnx.com/) contains a microkernel only 4 KB in size. The remaining OS functions in files: Fsys, Proc, Dev, Net, IPC, interrupt handling, and others are loaded by the microkernel when needed. There is no need to recompile the kernel, like in Linux. Even the superuser "root" cannot enter the "boot" directory and delete data, only "boot." QNX 6 Neutrino (https://qnx.software/) is a single-process (node) philosophy that blocks all other processes for itself (up to 4 ms), and other processes significantly slow down. Windows 98 operated on this principle, as it didn't have processor threads like Windows NT/XP. However, a single Windows 98 process didn't guarantee a response time of up to 4 ms, unlike QNX4. Starting with Windows NT, high-priority emulation and real-time mode are supported. Simply launch any program with the "start /high prog" or "start /realtime prog" commands. In FreeBSD, the kernel automatically compiles to an ELF file and runs. Possible make settings in the "/usr/src/sys/conf/kern.*.mk" config files during the initial FreeBSD boot from the "/red/herring" file using the "vi" and "ee" editors. The ELF file will then be compiled and launched. The kernel then loads additional services and even a terminal, if needed. Kernel parameters are stored in the "CONFIG_FILE" file. The kernel is launched with the "sh" command. Using heap fragmentation for a single task will speed up application execution, but this requires developing a custom OS.

If the OS is designed for a specific task [2, 5] (intelligent video surveillance for video camera chips), then such an OS will boot and run faster.

5 Solution Method

Despite its complexity (and its importance for PCs), creating an operating system is simple. The volume of the source code of the popular Minix 3.3.0 OS (https://minix3.org/) is merely as the volume of appendix in Andrew Tanenbaum's book "Creating an Operating System" [1]. Samsung monitors contain their own Tizen OS (in earlier versions: Bada and Orsay OS) with regular updates. Not all new OS are Linux clones, like Android Linux. Creating your own OS kernel is not that difficult.

A full-fledged OS can be considered as a program that provides:

1) reading the binary application format;
2) RAM manager for application collaboration;
3) memory management (heap, steak, disks) with access rights;
4) support of various devices with data stream redirection (event or interrupt pool for applications).

The kernel can be monolithic (as in Linux), where device drivers, memory manager, and utilities are contained in a single file and transferred to the RAM (heap) at OS startup. The kernel can also be a microkernel (as in QNX) and contain multiple files that are loaded separately.

The quality of an embedded technological operating system for video surveillance devices can be improved based on the following [9]:

- scan devices during OS operation, not at boot time;
- develop a RAM manager to allocate the heap to a single application;
- execute bytecode from an application without translation to a specific processor architecture;
- optimize bytecode for RAM clustering (heap) and the registers of a specialized processor in the device chip (video camera);
- eliminate permissions, access restrictions, and process threads.

To interact with the operating system, a command-line terminal capable of entering commands via familiar protocols is required, as connecting directly to the camera's pins and interfaces on the board is very inconvenient. This is especially true if the camera is located remotely (on the ceiling, at a height, etc.). A management tool and application development for remote connection are needed. The browser-based shell should be deeply integrated, such as Webmin 2.630 (https://webmin.com/). The tool should be deeply integrated into Subix with the ability to connect remotely via TCP/IP and combine the traditional Webmin push-button interface with the Command Line Interface (CLI) with four components (Prompt, Console, Terminal, Shell) in a simple and straightforward manner for a camera configuration specialist, not a system developer. This ensures easy daily use and configuration as needed for intelligent video surveillance systems.

In the WinSubs [8] graphics subsystem of the Subix operating system it is sufficient to create a simulated console with execution of all commands and viewing video fragments

from the camera's flash memory. A possible implementation option for the graphical subsystem (Fig. 3) for the Subix operating system is an area in the browser where the data is displayed and a graphical shell with a command line developed in the form of a website using HTML, CSS 3.2, and JavaScript as the frontend and PHP with Subscript [2] as the backend. The workspace is divided into two parts: in the upper part the Subix user is provided with real-time video viewing (left) and viewing individual fragments from past periods (right) with the ability to navigate using arrows that change the chronology depth (center). The second half (bottom) contains a console in which commands can be entered even without the WinSubs graphical subsystem (left). On the right is the area of control buttons: Menu for returning to the main website screen; Settings for switching to a separate video camera settings screen. The cross at the top of the Subix logo opens a page for rebooting the system.

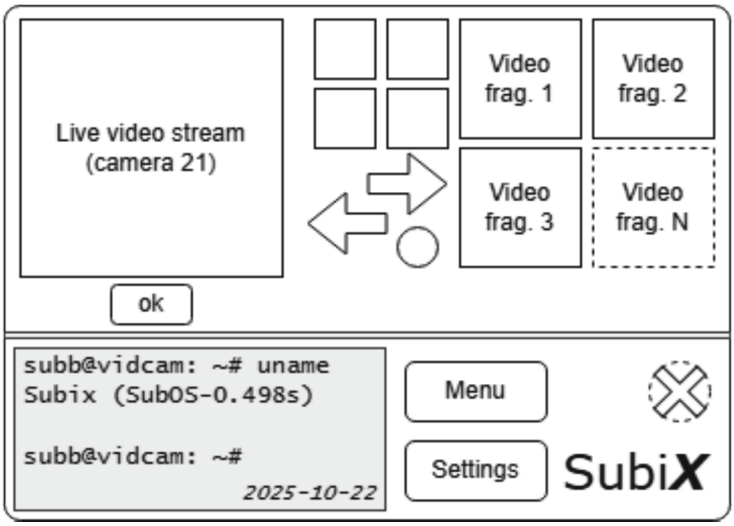

Fig. 3. Graphic subsystem for Subix OS.

In addition to interrupt remapping, executing instructions from the heap, maintaining access rights, drivers, and working with devices, any OS needs to boot from disk, which requires a file system (FS). For Axis P1365 MK II video cameras on chips with the ETRAX 100LX processor with RISC ISA instructions and 16 MB of RAM, the SubSysVideo FS [7] is suitable. It is adapted for intelligent video surveillance systems and is almost 50% faster than similar systems. WinSubs [8] is a suitable graphical shell; it visualizes events and is 20% faster than similar systems. For creating applications in the Subix operating system (SubOS-0.498s), the SubScript [2] scripting language is suitable; it is 6 times faster than similar systems. A description and comparison of the SubScript language [2] with a description of the language execution model (JIT, AOT, interpreter), memory management and comparative performance tests with Lua, Python, PHP and more than 17 languages (https://github.com/alex1543/17langs) on the same hardware are presented in detail in the work [2]. A new scripting language with

certain characteristics (the absence of configuration files, plug-in libraries and a complex, intricate scripting language core) and the use of the GCC 15.2 compiler (https://gcc.gnu.org/) can achieve an increase in the data exchange speed. The SubScript language [2] operates on the principle of macros and templates, for example, like rss2fido Script (https://rss2fido.sf.net/). The SubScript scripting language is created specifically for storing data for a single task and is executed as efficiently as possible. Performance measurements were carried out through JavaScript on the website (https://github.com/alex1543/EditTable), where you can select different languages for comparison (PHP, Perl, Python, Ruby, Go, Lua, Elixir, Dart, Node.js, etc.) for working on the backend for visualizing events and storing data in an intelligent video surveillance system [3, 4, 8].

6 Case Study

The authors of the study developed a remote connection application for the Subix OS (Fig. 4), which was used at the State Unitary Enterprise "Petersburg Subway" (https://metro.spb.ru/). Subix OS runs a webserver written in C for the GCC compiler (https://github.com/alex1543/practGCC). To connect to the graphical interface, simply enter "https://192.168.25.147/subix" in your browser, where "192.168.25.147" is the IP address, which can be changed in the settings. Connection is possible via either twisted pair (Ethernet) or USB.

After conducting the experiment (Table 3) within which the following parameters were measured: p1 - OS boot time; p2 - program execution time from RAM (heap); p3 - OS interrupt response time, it was found that Subix OS (SubOS-0.498s) works 2.3 (2.258) times faster than its analogs (Embedded Linux, QNX, Windows IoT, NetBSD, macOS). Data on the p2 and p3 parameters for QNX and Subix (SubOS-0.498s) could not be measured more accurately due to the timer limitation to 1000 ms. Launching and running applications in the new Subix operating system are 67% faster than QNX OS in real time. On other similar OSs, it was not possible to achieve real time mode, therefore the p2 and p3 parameters show the result of OS operation, that is on average slower by 978 (977.6) ms. For one application execution and launch.

To reproduce the experiment, compile the Subix OS source code (SubOS-0.498s) using a BASH script ("./comp_subb.sh"). Install all dependent applications, measure the time per day (24 or 25 h), open the logs in Apache OpenOffice (https://www.openoffice.org/), run JavaScript, and fill the resulting table. Thirty-one Axis P1365 IP video cameras (https://help.axis.com/en-us/axis-p1365-mk-ii) were used for testing. Statistics were processed on an Apple iMac M4 computer with the following technical specifications: 4 cores, IPS, 4.5K Ultra HD (4480x2520), 24 GB LPDDR5X, 1000 GB SSD, macOS 26.1 (T2/Apple Silicon 2025).

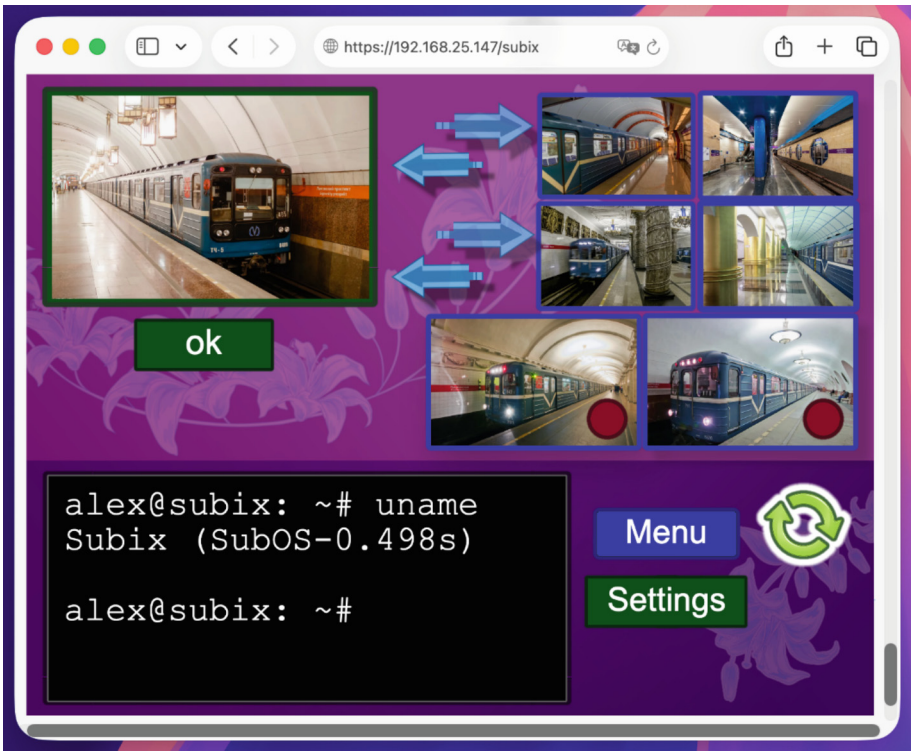

Fig. 4. Implementation of the graphical subsystem and command console in the Subix OS.

Table 3. Correlation of OS names and performance parameters.

OS Name	p1 (sec.)	p2 (ms.)	p3 (ms.)	Slower than Subix (times)
Embedded Linux[a]	157	2174	1279	2.63
QNX	124	2	1	2.03
Windows IoT	139	1492	873	2.32
NetBSD	142	1759	1028	2.37
macOS	117	893	275	1.94
Subix (SubOS-0.498s)	61	1	1	-
Subix is faster on average:				2.258

a) The default system in the subway (now).

7 Conclusion

The authors have created an operating system adapted for video camera chips for intelligent video surveillance systems.

The study found that the new Subix OS (SubOS-0.498s) boots twice as fast (2.22623) as its competitors; applications launch from disk 1000 times faster (1264) and execute in RAM 600 times faster (691.2). In real-time mode, the new operating system runs and launches applications 67% faster and in non-real-time is 600 times faster. According to the QNX documentation in real-time mode the first launched application must respond within 4 ms for a processor of 40 MHz or faster.

The proposed method for improving OS performance for embedded process computers can be applied not only in video cameras of video surveillance systems, but also in other devices: smart routers, industrial controllers, and process tablets at various stages of production and technical system maintenance.

Acknowledgments. This work was supported by the state budget, project No. FFZF-2025–0019.

References

1. Tanenbaum, A.S., Woodhull, A.S.: Operating systems: design and implementation, January 2006. ISBN 9780131429383
2. Subbotin, A., Delhibabu, R., Zhukova, N.: A script language creation for event visualization and data storage in an intelligent video surveillance system. Adv. Artif. Intell. Mach. Learn. **4**(4), 3161–3172 (2024)
3. Osipov, V., Zhukova, N., Subbotin, A., et al.: Intelligent escalator passenger safety management. Sci. Rep. **12**, 5506 (2022). https://doi.org/10.1038/s41598-022-09498-x
4. Tianxing, M., Vodyaho, A., Zhukova, N., Subbotin, A., et al.: Urban intelligent assistant on the example of the escalator passenger safety management at the subway stations. Sci. Rep. **13**, 15914 (2023). https://doi.org/10.1038/s41598-023-42535-x
5. Osipov, V., Nikiforov, V. Recurrent neural networks with controlled elements in restoring frame flows. Inf. Control Syst., 10–17 (2019). https://doi.org/10.31799/1684-8853-2019-5-10-17
6. Simmonds, C.: Mastering Embedded Linux Programming (2017)
7. Subbotin, A.N., Zhukova, N.A.: Development of a file system for storing data of an intelligent video surveillance system. Sci. Tech. J. Inf. Technol. Mech. Opt. **25**(2), 253–260 (2025). (in Russian). https://doi.org/10.17586/2226-1494-2025-25-2-253-260
8. Subbotin, A., Zhukova, N., Stankova, E.: Intelligent browser window system for visualizing events detected from video in distributed fog computing environment. In: Gervasi, O., et al. Computational Science and Its Applications – ICCSA 2025 Workshops. ICCSA 2025. Lecture Notes in Computer Science, vol. 15894, pp. 309–324. Springer, Cham (2026). https://doi.org/10.1007/978-3-031-97648-3_21
9. Titova, A.P., Karpov, N.S., Alexandrov, I.A.: Creation of an operating system that enables additive manufacturing. In: Proceedings of the 2024 Dynamics of Systems, Mechanisms and Machines (Dynamics), Omsk, Russian Federation, pp. 1–5 (2024). https://doi.org/10.1109/Dynamics64718.2024.10838674
10. Ilyushin, P., Volnyi, V., Suslov, K.: Review of approaches to creating control systems for solid-state transformers in hybrid distribution networks. Appl. Sci. **15**(20), 10970 (2025). https://doi.org/10.3390/app152010970
11. Berry, F., Kleihorst, R., Galàn, R.C.: Smart camera architecture. J. Syst. Architect. **59**(10), Part A, 817 (2013). ISSN 1383-7621, https://doi.org/10.1016/j.sysarc.2013.11.002

12. Gao, L., Zong, R.: Design of intelligent window automatic monitoring system based on microcontroller control. J. Electron. Res. Appl. **8**, 34–40 (2024). https://doi.org/10.26689/jera.v8i4.7922
13. Said, Y., Saidani, T., Atri, M.: Embedded architecture for real-time video processing. Imaging Rob. **15**(4) (2015). ISSN 2231-525X, https://doi.org/10.1109/DSD.2008.156
14. Lee, J.-H., Hyeon, B.-S., Jeon, O.-Y., Park, N.I.: Analysis of real-time operating systems' file systems: built-in cameras from vehicles. Forensic Sci. Int. Digit. Invest. **44**(Supplement), 301500 (2023). ISSN 2666-2817, https://doi.org/10.1016/j.fsidi.2023.301500
15. Zhou, Q.: Frequency-domain characterization of finite sample linear systems with uniform window inputs. Signals **6**, 1 (2025). https://doi.org/10.3390/signals6010001
16. Pietzsch, A., Woerd, J., Andrae, M., Gebbeken, N.: Analysis of the load-bearing behavior of catcher-cable systems for hazard prevention for windows under blast loads (2024)
17. Elavarasi, Anand, M., Sathya, S.: Design and implementation of reconfigurable virtual stage for real-time component using FreeRTOS. Int. J. Pure Appl. Math. **117**(16), 719–726 (2017). ISSN: 1314-3395, https://doi.org/10.12733/jica.2017.117.16
18. Sisi, C.: Empirical study on functional defects in multi-window Android systems. Appl. Comput. Eng. **73**, 288–294 (2024). https://doi.org/10.54254/2755-2721/73/20240413
19. Giatsintov, A., Mamrosenko, K., Bazhenov, P.: Architecture of the graphics system for embedded real-time operating systems. Tsinghua Sci. Technol. **28**(3), 541–551 (2023). https://doi.org/10.26599/TST.2022.9010028
20. Deng, L., Shi, Z., Wang, Y., Yu, X., Guan, Y., Xu, Z.: Hybrid window decoding for joint source channel anytime coding system. Entropy **26**, 940 (2024). https://doi.org/10.3390/e26110940
21. Wolf, W., Ozer, B., Lv, T.: Smart cameras as embedded systems. Computer **35**(9), 48–53 (2002). https://doi.org/10.1109/MC.2002.1033027