International Journal of Global Optimization and Its Application
Vol. 3, No. 2, June 2024, pp.75-80
e-ISSN: 2948-4030

DOI: 10.56225/ijgoia.v3i2.302

Original Article

Towards the Unraveling of Zombie Effect in the Linux Kernel

Shekh Abdullah-Al-Musa Ahmed [1,*], Md. Mahmudur Rahman [1], Shah Md. Baizid Habib [1], Ayesha Siddika Uzra [1] and Mabia Akonda Jemi [1]

[1] Department of Computing and Information System, Faculty of Science and Information Technology, Daffodil International University, Daffodil Smart City, Birulia 1216, Bangladesh; mahmud@daffodilvarsity.edu.bd (M.M.R.), shah@diu.edu.bd (S.M.B.H.), siddika15-14583@diu.edu.bd (A.S.U.), mabia@diu.edu.bd (M.A.J.)
* Correspondence: musaahmed.cis@diu.edu.bd (S.A.A.)

Citations: Ahmed, S.A., Rahman, Md.M., Habib, S.M.B., Uzra, A.S. & Jemi, M.A. (2024). Towards the Unraveling of Zombie Effect in the Linux Kernel. *International Journal of Global Optimization and Its Application, 3*(2), 75-80.

Received: 8 February 2024 Revised: 28 May 2024 Accepted: 6 June 2024 Published: 30 June 2024

Abstract: A zombie process is a type of process that has completed its execution but still has an entry in the process table. Zombie processes are typically created when a child process terminates, but its parent process fails to collect its termination status. Now that the fork () has created a new process, it duplicates the calling process. The new process is referred to as the child process. The calling process is referred to as the parent process. The child and parent processes are to run in separate memory spaces. Both memory spaces have the same content at the time of fork (). Whenever in Linux, a process is an instance of executing a program or command. While these processes exist, they will be in one of the five possible states, such as Running or Runnable (R), Uninterruptible Sleep (D), or Interruptable Sleep (S). Another process (the parent process) creates a child computing process. This technique pertains to multitasking operating systems and is sometimes called a subprocess or, traditionally, a subtask. Hence, there are five Linux process states. Such as running & runnable, interruptable sleep, uninterruptable sleep, stopped, and zombie. Each of these processes exists for a very well-defined reason. After all, a zombie process in Linux is a terminated child process that remains in the system's process table while waiting for its parent process to collect its exit status. A zombie process is a type of process that has completed its execution but still has an entry in the process table. As a result, if too many zombie processes are generated, the process table will be full. That is, the system will not be able to generate any new process, then the system will come to a standstill.

Keywords: Linux kernel; Process table; Zombie state; Kernel functions; Zombie processes; Memory management; Parent process; Child process.

1. Introduction

A zombie process in Linux is a terminated child process that remains in the system's process table while waiting for its parent process to collect its exit status. A zombie process is a type of process that has completed its execution but still has an entry in the process table. The OS uses the zombie state to make the parent check the child's process exit status and resource usage, such as CPU time, memory usage, IO cycles,

etc. Without the zombie state, the operating system would remove the information about the child's processes from the table the moment they finish (Feitelson, 2012). Even though under normal system operation, zombies are immediately waited on by their parent and then reaped by the system. Processes that stay zombies for a long time are usually an error and can cause a resource leak. Generally, the only kernel resource they occupy is the process table entry, their process ID (Wang et al., 2003). Whereas If the parent decides not to wait for the child's termination and executes its subsequent task, then at the termination of the child, the exit status is not read.

Hence, there remains an entry in the process table even after the termination of the child. This state of the child process is known as the Zombie state. Furthermore, once the zombie process is using the command "ps aux | grep Z" or "ps -eo pid, ppid, stat, com," then can kill it using the command "kill PID" where PID is the process ID of the zombies process. This command sends a signal to the process to terminate, and the process will be removed from the system. However, the zombies do not use any system resources. They use a PID, but the maximum number of PIDs is limited. So, if there are too many zombies, there is a risk of running out of PIDs, which is bad since the zombie's processes are terminated but have its entry in the process table to report to its parent process. Child processes are created during runtime using fork () by parent processes. An orphan process is formed when its parent dies while the process continues to execute, while a zombie process is a process that has terminated but its entry is in the system (Yang et al., 2011). After all, a zombie is a malicious program installed on a device that transforms it into a "zombie" that attacks other systems. A virus or Trojan first infects a computer or other device transformed by zombie malware.

2. Literature Review

The Linux kernel is the main component of a Linux operating system (OS) and is the core interface between a computer's hardware and its processes. It communicates with them and manages resources as efficiently as possible. Even though the major aim of kernel is to manage communication between software such as user-level applications and hardware (Reghenzani et al., 2019), CPU and disk memory (Nagpal et al., 2016). The objective of Kernel is to establish communication between user-level applications and hardware. As well as to decide state of incoming processes. It is as though the kernel is written in the C programming language (Saravanan et al.,2019). More precisely, the kernel is typically compiled with gcc under –std. Since the kernel is the most important part of the operating system. It is the primary interface between a computer's hardware and processes. The kernel connects these two to adjust resources as effectively as possible. Wherever the Kernel documentation, like the kernel itself, is very much a work in progress. So that is especially true when integrating many scattered documents into a coherent whole. So the Kernel functions are mathematical functions used in Support Vector Machines (SVMs) to transform input data into a higher-dimensional feature space (Zhang et al., 2015).

This transformation can make data points more linearly separable or capture complex relationships (Suresh et al.,2018). Hence, the Kernel functions have been very useful in data classification for identification and verification. Applying such mappings first and using some methods on the mapped data, such as principal component analysis (PCA). Furthermore, the most important benefit of the Kernel method is that it can work with non-linearly separable data and with multiple Kernel functions that depend on the type of data. Whereas, If the parent decides not to wait for the child's termination and executes its subsequent task, then at the termination of the child, the exit status is not read. Hence, there remains an entry in the process table even after the termination of the child. This state of the child process is known as the Zombie state. On the Unix and Unix-like computer operating systems, a zombie process or defunct process is a process that has completed execution via the exit system call but still has an entry in the process table.

So, it is a process in the terminated state. Furthermore, a zombie process is a process that has finished its execution but is still in the process table, waiting for the parent to read its exit status and take it out from the process table. After all, the English word "zombie" was first recorded in 1819 in the history of Brazil by the poet Robert Southey in the form of "zombi". Dictionaries trace the word's origin to African languages, relating to words connected to gods, ghosts and souls. Consequently, a zombie process is unaffected by the operating system command to end it forcefully (Nagpal et al., 2016). Long-running zombie processes result from unintentional mistakes and resource leaks, taking up much space in the process table. Hence, we need to avoid zombie build-up. Finally, proponents of zombie arguments generally accept that p-zombies are not physically possible, while opponents necessarily deny that they are metaphysically or, in some cases, even logically possible. The unifying idea of the zombie is that of a human completely lacking conscious experience.

3. Materials and Methods

Even if the parent decides not to wait for the child's termination and executes its subsequent task, then at the termination of the child, the exit status is not read. Hence, there remains an entry in the process table even after the termination of the child. This state of the child process is known as the Zombie state. Every time a process is removed from a Linux system, it informs its parent process about its execution. And until it has notified its parent, it stays in the process descriptor's memory (Hussain et al., 2003). This means that a dead process is not immediately removed and continues to hog the system's memory, hence becoming a zombie. Since this identified the zombie process using the command "ps aux | grep Z" or "ps -eo pid, ppid, stat, comm". So, it can kill it using the command "kill PID" where PID is the process ID of the zombies process. This command sends a signal to the process to terminate, and the process will be removed from the system. There is no need to eliminate zombie processes unless there is a large amount on the system. However a few zombies are harmless. Hence, there are a few ways to get rid of zombie processes. However, if the parent process is not programmed properly and is ignoring SIGCHLD signals, this won't help. After all zombie processes do not consume resources, for the most part. Each zombie process is still allocated a process ID number or PID. On 32-bit systems, the max number of PIDs available is 32767.

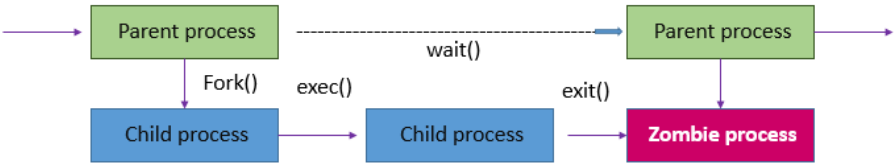

Figure 1. Zombie processes

Consequently, the Linux kernel manages the data and checks the necessary permissions for each process. For example, when opening a file, the kernel has to check the process ID against access permissions. If the kernel checks and sees that have permissions, it will open the file. Furthermore, a kernel process inherits its parent process's environment, which one calls the create kernel service to create, but with some exceptions. When initialized, the kernel process does not have a root or current directory. All uses of the file system functions must specify absolute path names (Jimenez et al., 2016; Hertel et al., 2003). Hence, the Linux kernel is the kernel used by Linux-based operating systems and is the interface between hardware and computer processes. A kernel is a special program responsible for managing the low-level functions of a computer (Merlo et al., 2002). Meanwhile, the Linux kernel consists of several important parts: process management, memory management, hardware device drivers, filesystem drivers, network management, and other bits and pieces (Raheja et al., 2016).

However, Linux is the kernel program in the system that allocates the machine's resources to the other programs. The kernel is an essential part of an operating system but useless by itself; it can only function in the context of a complete operating system. Whereas the UNIX kernel is the central core of the operating system. It provides an interface to the hardware devices as well as to process, memory, and I/O management. The kernel manages user requests via system calls that switch the process from user to kernel space. Since a kernel is the most important piece of software in an Operating System and is made up of many core components such as CPU management (Process & thread scheduling), Memory management (Paging, Segmentation algorithms), I/O Management, Process/Thread synchronization mechanisms (Mutexes, Spinlocks, Semaphores), etc. Whenever the smallest it can be is <1 MB, it does not become useful until adding enough device drivers and features to bring it up to 1 or 2 MB. Uncompressed, and with most of the modules statically linked, it could be as big as 15 MB (Khan et al., 2022).

After all, we can prevent that by calling the wait call immediately after a child process is executed so that it's cleaned up from the process table as soon as possible. Once a process becomes a zombie, it loses its memory pages, open file handles, semaphore locks, etc. As a result, to specifically identify zombie processes, one can search for the 'Z' state in the STAT column. For example, the output Z in the STAT column or the zombie or defunct> pattern from the ps command output can be used to identify zombies, as seen from the result.

4. Results and Discussion

The problem with the zombie is that it wastes system resources. The OS can not release the process identifier of the zombie process, such as pid in Linux. This is so because the identifier cannot be reallocated until officially released (Xu et al., 2015), even though zombie processes are usually harmless and do not impact the overall system performance. However, too many zombie processes can indicate a problem with the parent or system's process management. Now, if too many zombie processes are generated, the process table will be full. That is, the system will not be able to generate any new process, then the system will come to a standstill (see Figure 2).

Other	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Field Name	TICK	PID	PPD	USER ID	STATE	SWAP FLAG	NODE INDEX	INPUT BUFFER	MODE FLAG	USER AREA SWAP STATUS	USER AREA PAGE NUMBER	KERNEL STACK POINTER (KPTR)	USER STACK POINTER (UPTR)	PTBN	PTLR

Figure 2. Result of process table

Every time a process is removed from a Linux system, it informs its parent process about its execution. And until it has notified its parent, it stays in the process descriptor's memory (Supriya et al., 2016; Ike et al., 2023). This means that a dead process is not immediately removed and continues to hog the system's memory, hence becoming a zombie. While dangers of the Zombie Process occur, If their parent processes are no longer executing, zombie processes may signal an operating system issue. If only a few zombie processes exist, this is not a big deal, but it might cause problems for the system when under a lot of stress. After all, a zombie process is unaffected by the operating system's command to end it forcefully (Akkan et al., 2013). Long-running zombie processes result from unintentional mistakes and resource leaks, taking up much space in the process table. Hence, we need to avoid zombie build-up. Consequently, on Unix and Unix-like computer operating systems, a zombie or defunct process is a process that has completed execution via the exit system call but still has an entry in the process table that it is a process in the terminated state.

Furthermore, using SIGCHLD Signal. That can manually send the SIGCHLD signal to the parent of a zombie process. Consequently, it will intimate the parent to trigger the wait () system call, which will clean up the defunct child process from the process table. Hence, the zombie processes do not use up any system resources. Actually, each one uses a very tiny amount of system memory to store its process descriptor. However, each zombie process retains its process ID (PID). Linux systems have a finite number of process IDs — 32767 by default on 32-bit systems. The OS uses the zombie state to make the parent check the child's process exit status and resource usage, such as CPU time, memory usage, IO cycles, etc. Without the zombie state, the operating system would remove the information about the child's processes from the table the moment they finish, even though the process table is needed to store the state of a process that is currently suspended, either ready or blocked (Wang et al., 2003). It is not needed in a single-process system because the single process is never suspended. Now, a child's process becomes a zombie before being removed from the process table.

The parent process reads the child process's exit status, which reaps off the child process entry from the process table. Since a zombie is already dead, so you cannot kill it. To clean up a zombie, it must be waited on by its parent, so killing the parent should work to eliminate the zombie. After the parent dies, the zombie will be inherited by pid 1, which will wait on it and clear its entry in the process table. After all, every process becomes a zombie after its execution. As a result, processes marked as defunct are dead processes, so-called zombies, that remain because their parents have not destroyed them properly. These processes will be destroyed by init (8) if the parent process exits.

5. Conclusions

A zombie process is a terminated or completed process that remains in the process table. It will be there until its parent process clears it off. It calls wait () on its child and reads the exit value and other accounting information. Even if the parent decides not to wait for the child's termination and executes its subsequent task, then at the termination of the child, the exit status is not read. Hence, there remains an entry in the process table even after the termination of the child. This state of the child process is known as the zombie state. Since they are under normal system operation, zombies are immediately waited on by their parents

and then reaped by the system. Processes that stay zombies for a long time are usually errors and can cause a resource leak. Generally, the only kernel resource they occupy is the process table entry, their process ID (Abdulrahman et al., 2016). To prevent the zombie processes, use the wait () system call. After generating a child, the parent process will call wait () to wait for the child to complete and return its exit status. So, ignoring the SIGCHLD signal. The parent is sent a SIGCHLD signal when a child is terminated.

After all, potential security vulnerabilities, such as zombie processes, can introduce vulnerabilities in a Linux system. The existence of numerous zombie processes can confuse system administrators and monitoring tools, making it difficult to detect malicious activities. Consequently, zombie processes are not a problem as they take up very few resources, but there is one concern. Linux systems have a maximum amount of processes and thus process ID numbers. If a computer has enough zombie processes, the maximum amount is reached, and new processes can not be launched. Furthermore, we can prevent that by calling the wait call immediately after a child process is executed so that it is cleaned up from the process table as soon as possible. Once a process becomes a zombie, it loses its memory pages, open file handles, semaphore locks, etc. (Sivabalan et al., 2017). Hence, the first vulnerabilities are present in almost all parts of the kernel, including device drivers, kernel modules, and core kernel code. Solutions focusing on only one part of the kernel, such as kernel modules, are insufficient.

Meanwhile, the process table is a data structure maintained by the Linux kernel, containing information about each running process on the system. The process table in the Linux kernel is implemented as an array of process control blocks (PCBs) (Feitelson, 2012). Although sometimes referred to as a process in a terminated state. A Zombie process is usually cleaned from the memory system through a parent process. However, when the parent process is not notified of the change, the child process (zombie) does not get the signal to leave the memory. After all, there are multiple ways to check the state of a process in Linux. For example, command-line tools like ps and top can be used to check the state of processes. Alternatively, we can consult the pseudo-status file for a particular PID. As a result, the process table or process control block is the Process Table (PT) containing an entry for each process in the system. The entry is created when a Fork system call creates the process. Each entry contains several fields that store all the information about a single process.

Furthermore, the Kernel tables list frequently accessed information to help improve performance. Kernel tables improve performance because the database does not need to be checked to permit or deny resolving events listed in the kernel tables. Hence, the kernel handles memory management, including allocating and deallocating memory for processes and managing the virtual memory system. It provides mechanisms for memory protection, virtual memory mapping, and swapping data between physical memory and disk storage (Jimenez et al., 2016) (Sivabalan et al.,2017). A zombie process is a terminated or completed process that remains in the process table. It will be there until its parent process clears it off. It calls wait () on its child and reads the exit value and other accounting information. Finally, to avoid zombie processes, developers should ensure that their programs properly handle the termination status of child processes using system calls like wait () or waitpid().

Author Contributions: Conceptualization, S.A.A. and M.M.R.; methodology, S.A.A.; software, S.A.A.; validation, M.M.R., S.M.B.H., A.S.U. and M.A.J.; formal analysis, S.A.A. and M.M.R.; investigation, S.A.A. and M.M.R.; resources, S.A.A.; data curation, M.M.R., S.M.B.H., A.S.U. and M.A.J.; writing—original draft preparation, S.A.A., M.M.R., S.M.B.H., A.S.U. and M.A.J.; writing—review and editing, S.A.A., M.M.R., S.M.B.H., A.S.U. and M.A.J.; visualization, S.A.A.; supervision, S.A.A., A.S.U. and M.A.J.; project administration, S.A.A.; funding acquisition, S.A.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: We are grateful to Daffodil International University, including all the Computing and Information System Department faculties and all students in Operating System in Batch 16 and 17. Also, thank you to all the helping hands who support and help us in every step of our study.

Conflicts of Interest: The authors declare no conflict of interest.

References

Abdulrahman, A. A. K. (2016). Multi-Level Windows Exploitation Using Linux Operating System. *Asian Journal of Natural & Applied Sciences*, *5*(2),324-333.

Akkan, H., Lang, M., & Liebrock, L. (2013). Understanding and isolating the noise in the Linux kernel. *The International journal of high-performance computing applications*, *27*(2), 136-146.

Feitelson, D. G. (2012). Perpetual development: a model of the Linux kernel life cycle. *Journal of Systems and Software*, *85*(4), 859-875.

Hertel, G., Niedner, S., & Herrmann, S. (2003). Motivation of software developers in Open Source projects: an Internet-based survey of contributors to the Linux kernel. *Research policy*, *32*(7), 1159-1177.

Hussain, A., Heidemann, J., & Papadopoulos, C. (2003, August). A framework for classifying denial of service attacks. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications* (pp. 99-110).

Ike, U. K., Ononiwu, C., & Onumajuru, J. S. Analysis of Linux Kernel Iptables for Mitigating DDOS Attacks; A Component-Based Approach. *International Journal of Computer Science and Mathematical Theory, 9*(4),122-131.

Jimenez, M., Papadakis, M., & Le Traon, Y. (2016, October). Vulnerability prediction models: A case study on the linux kernel. In *2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM)* (pp. 1-10). IEEE.

Khan, A. W., Zaib, S., Khan, F., Tarimer, I., Seo, J. T., & Shin, J. (2022). Analyzing and evaluating critical cyber security challenges faced by vendor organizations in software development: SLR based approach. *IEEE access*, *10*, 65044-65054.

Merlo, E., Dagenais, M., Bachand, P., Sormani, J. S., Gradara, S., & Antoniol, G. (2002, August). Investigating large software system evolution: the linux kernel. In *Proceedings 26th Annual International Computer Software and Applications* (pp. 421-426). IEEE.

Nagpal, D., & Sharma, D. (2016). Overview of Threats and Attacks in Cloud Infrastructure. *International Journal of Research in IT, Management and Engineering, 6*(6), 16-23.

Raheja, S., Munjal, G., and Shagun, S. (2016). Analysis of Linux Kernel Vulnerabilities. *Indian Journal of Science and Technology, 9*(48), 39-48.

Ramirez, R., & Choucri, N. (2016). Improving interdisciplinary communication with standardized cyber security terminology: a literature review. *IEEE Access*, *4*, 2216-2243.

Reghenzani, F., Massari, G., & Fornaciari, W. (2019). The real-time linux kernel: A survey on preempt_rt. *ACM Computing Surveys (CSUR)*, *52*(1), 1-36.

Saravanan, A., Bama, S. S., Kadry, S., & Ramasamy, L. K. (2019). A new framework to alleviate DDoS vulnerabilities in cloud computing. *International Journal of Electrical & Computer Engineering (2088-8708)*, *9*(5),4163-4175.

Sivabalan, S., & Radcliffe, P. J. (2017, November). Detecting IoT zombie attacks on web servers. In *2017 27th International Telecommunication Networks and Applications Conference (ITNAC)* (pp. 1-3). IEEE.

Suresh, R., & Langes, C. M. Detection of Zombie Apps Using Zapdroid. International Journal of Latest Trends in Engineering and Technology,5(2),057-062.

Wang, X., & Reiter, M. K. (2003, May). Defending against denial-of-service attacks with puzzle auctions. In *2003 Symposium on Security and Privacy, 2003.* (pp. 78-92). IEEE.

Yang, Y., Littler, T., Sezer, S., McLaughlin, K., & Wang, H. F. (2011, December). Impact of cyber-security issues on smart grid. In *2011 2nd IEEE PES International Conference and Exhibition on Innovative Smart Grid Technologies* (pp. 1-7). IEEE.

Zhang, X., Knockel, J., & Crandall, J. R. (2015, April). Original SYN: Finding machines hidden behind firewalls. In *2015 IEEE Conference on Computer Communications (INFOCOM)* (pp. 720-728). IEEE.