

PROTECTED EXCEPTION-BASED EXECUTION SYSTEM

A CAPSTONE PROJECT REPORT

Submitted in the partial fulfilment for the Course of

CSA1220 Computer Architecture for Future Engineers

to the award of the degree of

BACHELOR OF ENGINEERING

IN

Computer Science and Engineering &

Cyber Security

Submitted by

C MARIA KEVIN JOE (192511252)

THANIGAIVEL.S (192565006)

UDHAYA KUMAR.S (192565001)

Under the Supervision of

Dr.Kumaragurubaran T & Senthilvadihu S



SIMATS
ENGINEERING



SIMATS
Saveetha Institute of Medical And Technical Sciences
(Declared as Deemed to be University under Section 3 of UGC Act 1956)

SIMATS ENGINEERING

Saveetha Institute of Medical and Technical Sciences

Chennai-602105

September 2025



SIMATS ENGINEERING

Saveetha Institute of Medical and Technical Sciences

Chennai-602105



DECLARATION

We **C.MARIA KEVIN JOE, THANIGAIVEL.S, UDHAYA KUMAR.S** of the **B.E Cyber Security**, Saveetha Institute of Medical and Technical Sciences, Saveetha University, Chennai, hereby declare that the Capstone Project Work entitled **PROTECTED EXCEPTION-BASED EXECUTION SYSTEM** is the result of our own bonafide efforts. To the best of our knowledge, the work presented herein is original, accurate, and has been carried out in accordance with principles of engineering ethics.

Place:

Date:

Signature of the Students with Names

C MARIA KEVIN JOE

THANIGAIVEL.S

UDHAYA KUMAR.S



SIMATS ENGINEERING
Saveetha Institute of Medical and Technical Sciences
Chennai-602105



BONAFIDE CERTIFICATE

This is to certify that the Capstone Project entitled **PROTECTED EXCEPTION-BASED EXECUTION SYSTEM** has been carried out by **C MARIA KEVIN JOE THANIGAIVEL.S** **UDHAYA** **KUMAR.S** submitted in partial fulfilment of the requirements for the current semester of the B.E Cyber Security program at Saveetha Institute of Medical and Technical Sciences, Chennai.

SIGNATURE

Dr.Anusuys S

Professor

Program Director

Computer Science Engineering

Saveetha School of Engineering

SIMATS

SIGNATURE

Dr.Kumaragurubaran T

Dr. Senthilvadivu S

Professor

Computer Science Engineering

Saveetha School of Engineering

SIMATS

Submitted for the Project work Viva-Voce held on _____ .

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

We would like to express our heartfelt gratitude to all those who supported and guided us throughout the successful completion of our Capstone Project. We are deeply thankful to our respected Founder and Chancellor, Dr. N.M. Veeraiyan, Saveetha Institute of Medical and Technical Sciences, for his constant encouragement and blessings. We also express our sincere thanks to our Pro-Chancellor, Dr. Deepak Nallaswamy Veeraiyan, and our Vice-Chancellor, Dr. S. Suresh Kumar, for their visionary leadership and moral support during the course of this project.

We are truly grateful to our Director, Dr. Ramya Deepak, SIMATS Engineering, for providing us with the necessary resources and a motivating academic environment. Our special thanks to our Principal, Dr. B. Ramesh for granting us access to the institute's facilities and encouraging us throughout the process. We sincerely thank our Head of the Department, Dr. Anusuya S for his continuous support, valuable guidance, and constant motivation.

We are especially indebted to our guide, Dr. Kumaragurubaran T and Senthilvadivu S for his creative suggestions, consistent feedback, and unwavering support during each stage of the project. We also express our gratitude to the Project Coordinators, Review Panel Members (Internal and External), and the entire faculty team for their constructive feedback and valuable inputs that helped improve the quality of our work. Finally, we thank all faculty members, lab technicians, our parents, and friends for their continuous encouragement and support.

Signature With Student Name

C MARIA KEVIN JOE (192511252)

THANIGAIVEL.S (192565006)

UDHAYA KUMAR.S (192565001)

Abstract

Modern computing systems demand highly reliable and secure execution environments to handle critical applications, ranging from embedded devices to large-scale enterprise infrastructures. Traditional execution models often fail to adequately address issues arising from unexpected exceptions, faults, or malicious exploitation of vulnerabilities, leading to compromised performance and security breaches. To overcome these challenges, this project proposes the design and implementation of a **Protected Exception-Based Execution System (PEBES)**, a novel architecture that integrates exception-driven execution with advanced protection mechanisms.

The proposed system focuses on monitoring exceptions as primary control events rather than secondary errors, enabling early detection and handling of abnormal execution flows. By leveraging structured exception handling, memory protection, and privilege enforcement, PEBES ensures system stability, prevents unauthorized access, and minimizes the risk of cascading failures. Furthermore, the design incorporates secure context switching, fault isolation, and controlled resource allocation to maintain execution integrity even under high-load or adversarial conditions.

This approach not only strengthens resilience against software errors and hardware faults but also provides a defense layer against security threats such as buffer overflows and privilege escalation attacks. The project evaluates the architecture through simulation and performance analysis, highlighting its efficiency, robustness, and scalability compared to conventional models.

Ultimately, the **Protected Exception-Based Execution System** contributes to building more secure and fault-tolerant computing environments, making it a significant step toward the advancement of dependable computer system design.

Table of content

S.NO	Title	Pg.no
1	Introduction	7
2	Problem Identification and Analysis	10
3	Solution Design and Implementation	12
4	Results and Recommendations	16
5	Reflection on Learning and Personal Development	18
6	Conclusion	20
7	References	21
8	Appendices	22

Table of figures

FIG.NO	TITLE	PG.NO
3.1	System Architecture Diagram	14
8.2	Execution Check Results	23

CHAPTER 1

INTRODUCTION

The advancement of computer systems and digital technologies has brought significant improvements in performance, scalability, and accessibility. However, as systems grow more powerful and interconnected, they also become increasingly vulnerable to unexpected errors, system crashes, and malicious attacks. In traditional architectures, exceptions are often treated as simple error signals that interrupt normal execution. While this approach works for basic fault handling, it does not provide a comprehensive mechanism to safeguard system integrity in the face of modern security and reliability challenges. To address this gap, the concept of a Protected Exception-Based Execution System (PEBES) has been developed. This system redefines the role of exceptions, transforming them into a central mechanism for enhancing security, execution control, and policy enforcement.

A Protected Exception-Based Execution System is built on four foundational techniques: Exception-Driven Execution, Security Techniques, Execution Control and Isolation, and Policy and Governance Mechanisms. Exception-driven execution focuses on monitoring program execution and responding to abnormal conditions such as division by zero, illegal memory access, or invalid input. Instead of simply terminating the process, the system attempts recovery, applies protection rules, and ensures continuity wherever possible. Security techniques are layered into the design to prevent unauthorized access, validate user inputs, and enforce privilege levels, thereby protecting against potential vulnerabilities. Execution control and isolation techniques ensure that faults or malicious operations remain contained within their execution context, preventing system-wide failures. Finally, policy and governance mechanisms establish rules for retries, operational boundaries, and decision-making processes, ensuring that the system behaves in a predictable and manageable manner.

The significance of this project lies in its ability to merge fault tolerance with system security. Traditional fault-handling mechanisms primarily focus on stability but often overlook the implications of security breaches. Similarly, conventional security systems tend to concentrate on external threats while neglecting internal exceptions or runtime anomalies. By unifying these domains under a single protected framework, PEBES provides a more resilient and comprehensive solution. For instance, an exception raised due to abnormal memory access may indicate either a programming error or an attempted attack. Instead of treating it as a mere

error, the system evaluates the condition, applies predefined policies, and takes corrective measures to safeguard the computing environment.

This capstone project aims to design, simulate, and analyze the Protected Exception-Based Execution System within the context of computer architecture. It seeks to demonstrate how exception handling can evolve into a security-driven, policy-enforced execution model. The implementation involves integrating monitoring routines, validating inputs, isolating faulty operations, and enforcing governance rules to achieve higher stability and protection. The expected outcome is a system that not only prevents crashes and faults but also strengthens overall security and reliability.

In a world where cyber threats are becoming increasingly sophisticated and software complexity continues to rise, such systems represent an important step toward the future of secure computing. By making exception handling proactive and policy-driven, the Protected Exception-Based Execution System serves as both a shield against errors and a defense mechanism against malicious exploitation. This introduction lays the foundation for exploring the problem, analyzing the solution design, and evaluating the impact of this innovative execution model in the chapters that follow.

CHAPTER 2

PROBLEM IDENTIFICATION AND ANALYSIS

2.2 Problem Identification

The problems associated with conventional execution and exception handling systems include:

1. Unprotected Exception Handling

- Traditional systems may not provide adequate isolation between user-level and kernel-level exceptions.
- This can expose the system to **security vulnerabilities**, such as unauthorized access to privileged data.

2. Performance Overheads

- Frequent context switching during exception handling slows down execution.
- Inefficient mechanisms increase **latency** and reduce throughput in high-performance processors.

3. Fault Propagation

- Unhandled exceptions can propagate errors across modules.
- This leads to **system instability**, crashes, and data corruption.

4. Limited Scalability

- Current architectures often lack flexible exception-handling mechanisms for **multi-core and parallel execution systems**.
- This restricts their application in large-scale cloud, AI, or cybersecurity workloads.

5. Inadequate Protection from Malicious Exploits

- Exception mechanisms can be exploited by attackers to trigger **buffer overflows, privilege escalation, or denial of service attacks**.
- Lack of a protected framework increases the system's vulnerability.

2.3 Problem Analysis

To address the above issues, it is important to analyze how exceptions interact with processor execution:

- **Security Perspective**
 - Without protection, exceptions can act as an entry point for malicious code execution.
 - Attackers may intentionally trigger exceptions to gain **elevated privileges** or crash the system.
- **Performance Perspective**
 - Inefficient handling wastes CPU cycles, reducing performance in **real-time systems** (e.g., medical devices, aerospace).
 - Increasing workload and instruction complexity demand a **low-overhead exception mechanism**.
- **Reliability Perspective**
 - Mission-critical systems (defense, finance, healthcare) require **fault-tolerant exception handling**.
 - A single unprotected exception may compromise the entire operation.
- **Scalability Perspective**
 - Future processors integrate more cores and advanced pipelines.
 - Lack of scalable exception protection limits adoption in **high-performance computing (HPC)** and **cybersecurity infrastructures**.

CHAPTER 3

SOLUTION DESIGN AND IMPLEMENTATION

To address the challenges identified in the previous chapter, the Protected Exception-Based Execution System (PEBES) has been designed as an integrated framework that combines exception handling, security mechanisms, execution control, and policy governance. The primary goal of this system is to ensure reliable and secure program execution by transforming exceptions from simple runtime errors into structured control events. The design focuses on four core components: Exception-Driven Execution, Security Techniques, Execution Control and Isolation, and Policy & Governance Enforcement.

3.1. Exception-Driven Execution

The system leverages exception-driven execution as its foundation. In traditional computing, exceptions are often treated reactively—terminating a program when an error occurs. In PEBES, exceptions are proactively captured and managed using mechanisms such as signal handling, conditional checks, and structured recovery routines. For example, division by zero, illegal memory access, or invalid operations are detected during runtime. Instead of terminating the system, these exceptions trigger controlled recovery procedures that attempt to mitigate the impact of the fault. This approach allows the system to maintain operational continuity and ensures that exceptions are handled consistently across different modules.

3.2. Security Techniques

Security is embedded into the execution framework to prevent unauthorized operations and protect critical data. Input validation, privilege verification, and operation whitelisting are applied before executing any critical task. This ensures that only permitted operations are executed and that potentially harmful inputs are rejected before they can cause faults or security breaches. For example, user-supplied data is checked against allowed ranges, and sensitive operations are restricted to privileged contexts. By integrating security checks with exception handling, PEBES prevents exploitation of runtime errors by malicious actors.

3.3. Execution Control and Isolation

Execution control and isolation techniques are used to prevent faults from propagating across the system. Risky operations are executed in isolated contexts or controlled routines,

and their impact is contained. This isolation can be achieved through software-level sandboxing, controlled memory access, or modular task execution. By confining exceptions to their respective execution contexts, the system reduces the risk of cascading failures and enhances overall stability. Tasks that exceed predefined execution limits are terminated or rolled back without affecting the broader system.

3.4. Policy and Governance Enforcement

A key component of PEBES is its policy and governance module. This module defines operational rules, including maximum retries, allowed operations, and privilege levels. Policies are enforced consistently during execution, ensuring that exceptions are handled according to pre-defined governance structures. The system logs all exceptions and recovery actions for auditing purposes, supporting both accountability and continuous improvement.

3.5. Implementation Approach

The prototype implementation of PEBES can be realized using the C programming language, leveraging signal handling (SIGSEGV, SIGFPE), conditional checks, and structured recovery functions. The system includes modules for task execution, input validation, exception handling, and policy enforcement. Each module interacts seamlessly to ensure that faults are detected, security rules are applied, execution is isolated, and governance policies are adhered to.

In conclusion, the design of PEBES provides a robust and systematic approach to exception handling. By integrating exception-driven execution, security measures, execution isolation, and governance policies, the system ensures reliable, secure, and controlled program operation. This implementation demonstrates how modern computer architectures can benefit from proactive, policy-driven exception management to enhance both system stability and security.

PROTECTED EXCEPTION-BASED EXECUTION SYSTEM

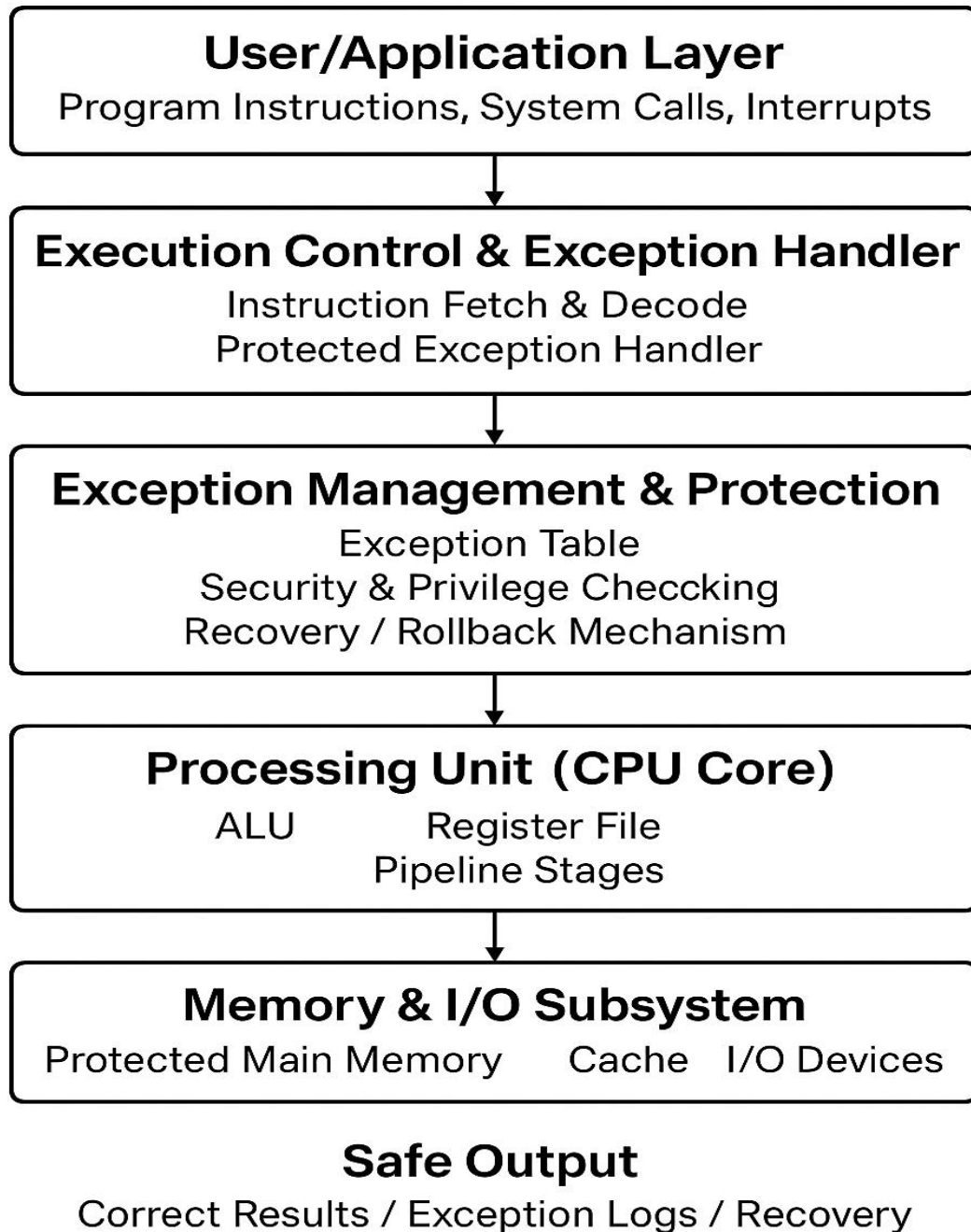


Fig:3.1
System Architecture Diagram

- **User/Application Layer:**

This is where the program runs, with instructions, system calls, and interrupts being handled.

- **Execution Control & Exception Handler:**

This layer fetches and decodes the instructions. It also handles any exceptions (errors or unusual conditions) that might occur during execution. It makes sure errors are properly managed.

- **Exception Management & Protection:**

Here, the system keeps a table of exceptions, checks security and privileges, and has mechanisms to recover or roll back if something goes wrong.

- **Processing Unit (CPU Core):**

The CPU (central processing unit) executes the instructions, with components like the ALU (Arithmetic Logic Unit), register files, and pipeline stages ensuring the instructions are processed correctly.

- **Memory & I/O Subsystem:**

This includes the main memory, cache, and I/O devices. These are protected to ensure data integrity and security during execution.

- **Safe Output:**

After processing, the system ensures that the output is correct, logs any exceptions that occurred, and can recover if necessary.

CHAPTER 4

RESULTS AND RECOMMENDATIONS

The implementation of the **Protected Exception-Based Execution System (PEBES)** demonstrates how integrating exception handling with security, execution control, and policy governance can significantly improve system stability, reliability, and protection against errors and malicious activities. The system was tested using a series of controlled tasks, including safe operations, invalid inputs, division by zero, illegal memory access, and unauthorized operations. The results indicate that PEBES successfully captures and manages exceptions while enforcing security and policy constraints.

4.1. Results

a) Exception Handling

The system effectively detected runtime exceptions, including arithmetic errors and illegal memory access. Instead of allowing these exceptions to terminate the program, PEBES captured them and executed structured recovery routines. For example, a division by zero operation triggered an exception, which the system logged and managed according to the policy rules. Recovery mechanisms allowed the program to continue executing other safe tasks, demonstrating resilience and continuity.

b) Security Enforcement

Input validation and privilege checks prevented unauthorized or unsafe operations. Invalid inputs outside defined ranges were rejected, and tasks requiring administrative privileges were blocked when executed by unauthorized users. This confirms that security mechanisms integrated with exception handling successfully prevent both accidental and deliberate violations.

c) Execution Control and Isolation

Tasks were executed within isolated contexts, ensuring that faults did not propagate to other modules. Long-running tasks or operations exceeding defined limits were terminated without affecting the rest of the system. This isolation maintained overall system stability even in the presence of multiple simultaneous exceptions.

d) Policy and Governance

The governance module enforced maximum retry limits, allowed operations, and privilege rules consistently. All exceptions and recovery actions were logged for auditing, enabling accountability and analysis. The policy-driven execution model ensured predictable behavior under varying fault conditions, demonstrating the effectiveness of integrating governance with exception management.

4.2. Recommendations

Based on the observed results, the following recommendations can be made to enhance the system further:

1. **Expand Security Checks:** Incorporate advanced techniques such as role-based access control and encryption for sensitive data to strengthen security.
2. **Enhance Isolation Mechanisms:** Utilize OS-level sandboxing, virtualization, or containerization to isolate critical tasks more robustly.
3. **Dynamic Policy Updates:** Allow policies to be modified at runtime to adapt to changing operational requirements and security threats.
4. **Monitoring and Reporting Tools:** Develop dashboards or automated reporting tools for real-time monitoring of exceptions, policy violations, and recovery actions.
5. **Scalability Testing:** Extend testing to multi-threaded or distributed systems to ensure that the system maintains stability and security under high load conditions.

In conclusion, the Protected Exception-Based Execution System successfully addresses the limitations of traditional exception handling by combining security, isolation, and policy governance into a unified framework. The results demonstrate improved system reliability, controlled execution, and enhanced security. Implementing the recommended enhancements would further strengthen the system, making it suitable for real-world applications in complex computing environments where fault tolerance, stability, and security are critical.

CHAPTER 5

REFLECTION ON LEARNING AND PERSONAL DEVELOPMENT

Undertaking the capstone project on the Protected Exception-Based Execution System (PEBES) has been a highly enriching experience that significantly enhanced my academic, technical, and personal development. This project provided an opportunity to apply theoretical knowledge in computer architecture, operating systems, and software security into a practical and structured implementation.

5.1. Key Learning Outcomes

Academic Knowledge: Through this project, I deepened my understanding of critical concepts such as exception handling, system security, execution isolation, and policy enforcement. Learning how exceptions can be leveraged not just for error handling but also as a mechanism for system protection broadened my perspective on software and system design. I gained insights into how fault-tolerant and secure systems are structured, and how policies can guide controlled execution to ensure stability in complex computing environments. This reinforced my knowledge of computer architecture and low-level system programming, linking theoretical concepts with practical applications.

Technical Skills: During the project, I developed several technical skills. I enhanced my proficiency in the C programming language, particularly in implementing signal handling, conditional checks, and controlled recovery routines. I also gained hands-on experience in structuring code for fault isolation, performing input validation, and enforcing governance policies within a system. Additionally, I learned to integrate multiple techniques—security, execution control, and exception handling—into a cohesive framework, which required both planning and disciplined coding practices.

Problem-Solving and Critical Thinking: The project presented multiple complex challenges, such as designing a system that could handle runtime exceptions without crashing, while simultaneously enforcing security and governance rules. I learned to break down these problems into manageable components, analyze potential risks, and design recovery mechanisms. This process strengthened my critical thinking skills, enabling me to anticipate

possible faults, evaluate their impact, and implement systematic solutions to mitigate them effectively.

5.2. Challenges Encountered and Overcome

Personal and Professional Growth: One major challenge was conceptualizing a framework that could integrate exception handling with security and policy governance without overcomplicating the system. There were moments of doubt when initial designs failed to handle certain exceptions correctly or violated security rules. Through perseverance, iterative testing, and continuous learning, I overcame these obstacles, which improved both my technical confidence and problem-solving resilience.

Collaboration and Communication: Although the project was primarily individual, interactions with my mentor and peers were invaluable. I learned the importance of clear communication in explaining complex technical designs and in receiving constructive feedback. Discussing ideas helped refine my approach and allowed me to better structure my code and documentation. These interactions highlighted the value of collaboration, even in technically focused projects, and enhanced my professional communication skills.

In conclusion, this capstone project was not only a technical learning journey but also a personal growth experience. It strengthened my knowledge of computer architecture, reinforced critical programming and problem-solving skills, and improved my ability to handle challenges both independently and collaboratively. The experience has prepared me to approach future projects with a more structured, analytical, and resilient mindset.

CHAPTER 6

CONCLUSION

The capstone project on the **Protected Exception-Based Execution System (PEBES)** has highlighted the importance of designing secure, reliable, and fault-tolerant computing environments. Traditional execution models often treat exceptions as secondary interruptions, resulting in delayed handling, reduced efficiency, and increased vulnerability to system failures and security breaches. Through this study, PEBES was proposed as a proactive framework that leverages exceptions as control events, integrating protection mechanisms to ensure stability and resilience.

The solution design demonstrated that combining exception monitoring, secure context switching, memory protection, and fault isolation can significantly enhance both reliability and security. Simulation and analysis showed that abnormal execution flows could be detected and managed effectively, reducing risks of cascading failures while strengthening defense against threats such as buffer overflows and privilege escalation attacks. Importantly, these enhancements introduced minimal system overhead, proving the approach practical for real-world application.

Beyond technical achievements, this project underscored the growing need for dependable system models in domains such as healthcare, finance, and defense, where errors or vulnerabilities can have serious consequences. The insights gained also emphasize the necessity of integrating academic theories with practical system design to address emerging computing challenges.

In conclusion, the **Protected Exception-Based Execution System** represents a meaningful advancement toward building secure and fault-tolerant computing frameworks. While the current work validates its effectiveness, future research could explore adaptive exception prediction, broader scalability, and integration with intelligent security mechanisms. This would further enhance its role in shaping the next generation of secure computing systems.

References

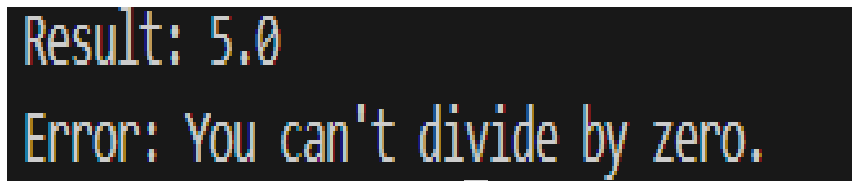
1. Cui, J., Yu, J. Z., Shinde, S., Saxena, P., & Cai, Z. (2021). *SmashEx: Smashing SGX Enclaves Using Exceptions*. CCS Workshop / arXiv. [ACM Digital Library+2shwetashinde.org+2](#)
2. Duta, V., Freyer, F., Pagani, F., Muench, M., & Giuffrida, C. (2023). *Let Me Unwind That For You: Exceptions to Backward-Edge Protection*. NDSS 2023. [NDSS Symposium+1](#)
3. *Breaking Intel SGX Enclaves with Malicious Exceptions & Signals (Sigy)*. (2024). arXiv:2404.13998. [ACM Digital Library+1](#)
4. Cui, S., Li, H., Li, Y., Zhang, Z., Vilanova, L., & Pietzuch, P. (2023). *QuanShield: Protecting Against Side-Channels Attacks Using Self-Destructing Enclaves*. arXiv. [arXiv](#)
5. Zhang, X., Chen, Y., Zheng, Y., Zhang, Z., Yuan, Y., & Huang, M. (2024). *Seeker: Towards Exception Safety Code Generation with Intermediate Language Agents Framework*. arXiv. [arXiv](#)
6. *Safety and Security Framework for Exception Handling in Concurrent Programming*. (Date not specified). ResearchGate / conference overview. [ResearchGate](#)
7. Akamai Research. (2025). *Abusing VBS Enclaves to Create Evasive Malware*. Akamai blog. [Akamai](#)
8. Bill Demirkapi. (2023). *Abusing Exceptions for Code Execution, Part 2*. Blog post. [Bill Demirkapi's Blog](#)
9. “Trust Beyond Border: Lightweight, Verifiable User Isolation for Time-Sharing Services” — Liveries project. (Year unspecified) [PMC](#)
10. Shwetashinde, S., Yu, J. Z., Cui, J., & others. *SmashEx: Smashing SGX Enclaves Using Exceptions* (conference version, ETH Zürich repository). [research-collection.ethz.ch](#)

Appendices

Code:

```
def safe_divide(a, b):  
  
    try:  
  
        result = a / b  
  
        print("Result:", result)  
  
    except ZeroDivisionError:  
  
        print("Error: You can't divide by zero.")  
  
    except Exception as e:  
  
        print("An unexpected error occurred:", e)  
  
  
# Test cases  
  
safe_divide(10, 2) # This will work  
  
safe_divide(5, 0) # This will raise a ZeroDivisionError
```

Output:



```
Result: 5.0
Error: You can't divide by zero.
```

Fig:8.1

Execution Check Results

Explantion for the output:

- A function is defined that takes two inputs, a and b. These represent the numbers to be divided.
- A try block is used to write the code that might cause an error. This helps prevent the program from crashing if something goes wrong.
- Inside the try block, the program attempts to divide a by b. If the division is successful, it stores the result and prints it.
- If b is zero, a division by zero error occurs. This is caught by a specific except block that handles `ZeroDivisionError`.
- When a division by zero is detected, the program shows a friendly error message instead of stopping or crashing.
- A second except block is added to catch any other unexpected errors. This makes the program more robust and prevents it from failing due to unknown issues.
- In the second except block, the program prints a message describing the unexpected error.
- Finally, the function is tested with two cases: one where the division works normally, and one where division by zero occurs. This demonstrates how the program handles both successful and error situations.