Hello everyone, my name is Betty, and today I'm excited to present my Year 3 project: ChronOS, a teaching operating system written in Rust.

This project including two main parts. Firstly, it covers fundamental concepts of operating systems, such as those found in CS241 Operating Systems and Computer Networks, as well as CS257: Advanced Computer Architecture. Additionally, it serves as a comprehensive learning resource for the Rust programming language. Even those who have never encountered Rust before can start from scratch and learn the language through this project.

ChronOS consists of two main components: the Labsheets and the corresponding code. The Labsheets provide step-by-step guidance, explaining each steps and the reasons behind it. Additionally, each task includes corresponding code examples, allowing students to run them in advance to see the results or compare their own implementations with mine. There is also test case provided for them to check the results.

Why create this operating system project? There are many excellent OS, with many third-year students choose to make this kind of OS as their projects. To explain why I set this project , let me first introduce them.

First, there are the labs provided with OS course at this moment, which every Warwick student would get in touch. It's an excellent set of labs, but it focuses more on the C language and how to use it. However, it doesn't build a runnable operating system kernel, which creates a bit of a gap between students and their real world experience of an operating system – one that's independent of the host machine.

Next, there's the Birmingham OS guide, which teaches you to write a kernel from scratch. It uses C and some assembly language, starting from the ground up. However, this book spends too much time on very low-level aspects, like the bootloader section, which can be too challenging for university-level CS students and general enthusiasts due to its steep learning curve (you can see this from the table of contents), and this book is also unfinished. While it provided some insights, it's not the most suitable tutorial.

PunchOS is a project from my seniors, also presented as a third-year project. I received help and inspiration from them while writing my project. It's also written in C and doesn't require students to build the entire system from scratch but provides a skeleton for students to implement several functionalities. This reduces the initial workload, striking a balance between difficulty and making an operating system.

However, I still want to help beginners create their own operating systems from scratch while introducing a brand new programming language. Most operating system projects are written in C, and I want to offer a new option. Rust, being superior to C in memory safety and other aspects, makes it easier and more convenient for everyone to write code. Additionally, there's no need to use assembly language, avoiding complications for those who haven't learned it. My lab starts from zero, introducing Rust installation, and received positive feedback on evaluation day. Even those who hadn't encountered Rust in their life, found it very simple and easy to understand.

In this project, I primarily use the RustRover IDE, which offers code highlighting, auto-completion, and testing features for development. Additionally, Git aids in version control, enabling multiple branches for flexibility, while GitHub serves as a backup and distribution platform to prevent data loss and facilitate updates.

And another tool is QEMU. It is a widely-used emulator for operating system development due to its versatility and platform support. It allows developers to run their code on different architectures without the need for physical hardware.

So at the technical challenge part, I would face 5 main challenges.

Firstly, since Rust is a new language for me, I spent a lot of time learning its concepts. Writing a project like this in an unfamiliar language posed a considerable challenge. Secondly, while most OS projects are written in C, there's limited support for using Rust, but I still wanted to be innovated. Additionally, there's often a gap between course content and practical application, with some topics not covered in the curriculum. What’s more, not like other software development, I develop something without the support of standard library, that’s very difference from normal software developement.

Lastly, the most challenging aspect is how to explain these concepts and principles in a clear and concise manner to beginners. As this is an educational OS project, I aim to provide a smooth learning curve, teaching both Rust and OS concepts. Explaining new concepts to others, rather than just learning and implementing them to myself. So these presents a fresh challenges.

Next, I'll introduce my system, which is divided into two main parts, as I mentioned earlier. Firstly, there are the Lab Sheets, currently consisting of five labs. Lab 0 provides an introduction to Rust language along with extensive learning resources. It is inspired by Lab 1 of CS241 Operating Systems and Computer Networks. But instead of providing some new sample code and then explaining the concepts related to that code, this lab will use these examples directly to explain new content when encountered in the remaining lab 1 through 4. Students using this lab don't need to complete Lab 0 beforehand; they can jump to relevant sections as new content is introduced. I've included inline explanations for every line of code, clarifying the purpose of each variable and aiding understanding of the system's design decisions.

Lab1 to Lab4 (possibly a fifth, depending on the content) each cover a specific aspect of the operating system. For example, Lab2 deals with VGA output, while Lab3 focuses on paging. Each lab is broken down into smaller tasks, where we introduce what each task aims to achieve, outline the specific implementation steps, and conclude with screenshots of the output.

Additionally, there's an "Extra Knowledge" section where students can delve into new concepts. My goal is to ensure that students can complete the entire project using only these labs, without the need for additional searches or learning. During the evaluation day, this labsheet received positive feedback from fellow students, many of whom had no prior experience with Rust. They appreciated the additional knowledge and the ability to navigate to Rust tutorials, which greatly facilitated their learning.

These screenshots demonstrate the navigation and extra knowledge features, and I'll showcase more details during the demo.

Next, let's talk about the code aspect. I've prepared corresponding code for each task in every lab. Students can run this code before implementing to generate output or compare their implementation with mine. Additionally, it can serve as a foundation for those who prefer not to start from scratch. As an educational operating system, I aim to make it as beginner-friendly as possible, lowering the barrier to OS development.

Meanwhile, in the code section, I also provide a test suite. After completing a lab, students can run the test cases I provide to check if each feature meets expectations. I'll also output error messages to indicate which part went wrong, or "ok" if everything is fine. This feature has received positive feedback too. As clear error messages help students in their development process.

Entering the project management part, I encountered several challenges throughout the project's progression. Underestimating the workload of other courses, as long as unexpected module failures, let me unable to work and setbacks in project progress. However, by employing the MOSCOW method to prioritize tasks and using incremental software development method. I ensured the I can display a functional, runnable system.

Regular tutor meetings and supervision by tutor were crucial for project success. We have meeting every 2 week and with meetings increasing to weekly sessions during critical stages.

So in conclusion, this project is succuessfully delivered.

I also have higher expectations for this project, and perhaps I can implement them in the near future. Firstly, let the system can receive input. Currently, the operating system cannot read keyboard inputs and only performs output based on pre-written content. This would involve interrupt-related content. Secondly, implementing processes and multitasking. Presently, the system cannot handle processes or their switching. With the implementation of interrupts and heap, I hope to accomplish this. Lastly, creating video tutorials for the labs. Similar to those provided in embedded courses, many students, including myself, prefer following along with step-by-step lab implementations demonstrated by video. Having videos demonstrating the content of each lab, I believe, would enhance the teaching effectiveness.