

操作系统实验1

邓人嘉 21301032

一、实验步骤

1.1 创建一个Rust项目

- 进入mnt目录，执行"cargo new os --bin"

```
[root@dcea8e5aca3c mnt]# cargo new os --bin
Created binary (application) `os` package
[root@dcea8e5aca3c mnt]#
```

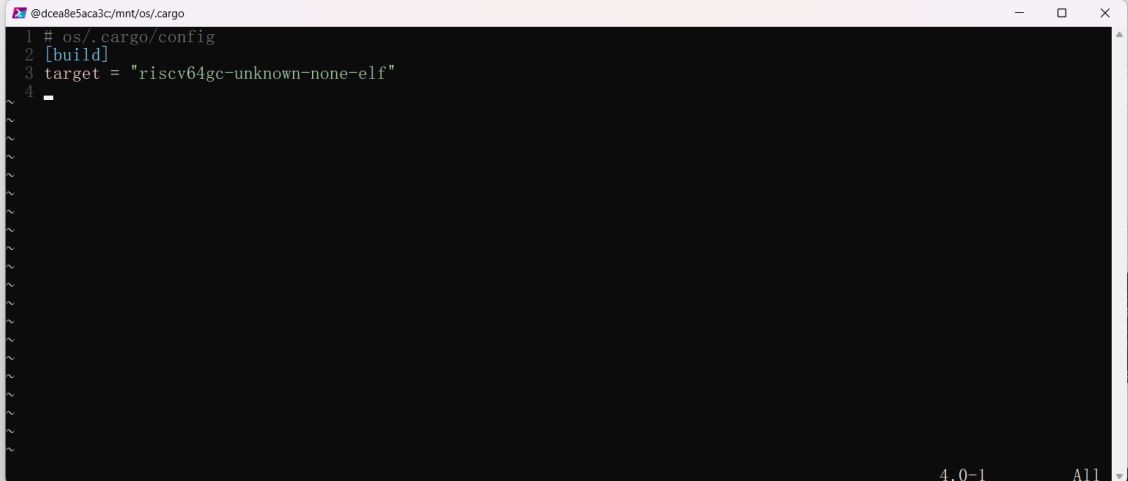
- cd os
- cargo build
- cargo run

```
Created binary (application) `os` package
[root@dcea8e5aca3c mnt]# cd os
[root@dcea8e5aca3c os]# ls
Cargo.toml  src
[root@dcea8e5aca3c os]# cargo build
Compiling os v0.1.0 (/mnt/os)
Finished dev [unoptimized + debuginfo] target(s) in 2.06s
[root@dcea8e5aca3c os]# cargo run
Finished dev [unoptimized + debuginfo] target(s) in 0.08s
Running `target/debug/os`
Hello, world!
[root@dcea8e5aca3c os]#
```

1.2 移除标准库依赖

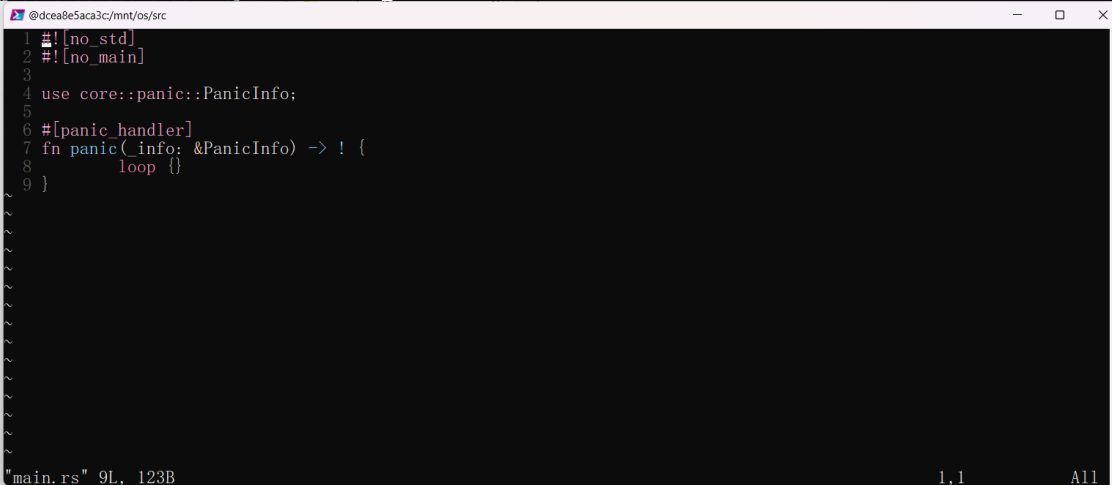
- 修改target为riscv64

```
[root@dcea8e5aca3c os]# mkdir .cargo
[root@dcea8e5aca3c os]# cd .cargo
[root@dcea8e5aca3c .cargo]# touch config
```



```
@dcea8e5aca3c/mnt/os/.cargo
1 # os/.cargo/config
2 [build]
3 target = "riscv64gc-unknown-none-elf"
4
```

```
[root@dcea8e5aca3c os]# ls
Cargo.lock Cargo.toml src target
[root@dcea8e5aca3c os]# cd src
[root@dcea8e5aca3c src]# ls
main.rs
[root@dcea8e5aca3c src]# vim main.rs_
```



```
1 #![no_std]
2 #![no_main]
3
4 use core::panic::PanicInfo;
5
6 #[panic_handler]
7 fn panic(info: &PanicInfo) -> ! {
8     loop {}
9 }
```

"main.rs" 9L, 123B 1,1 All

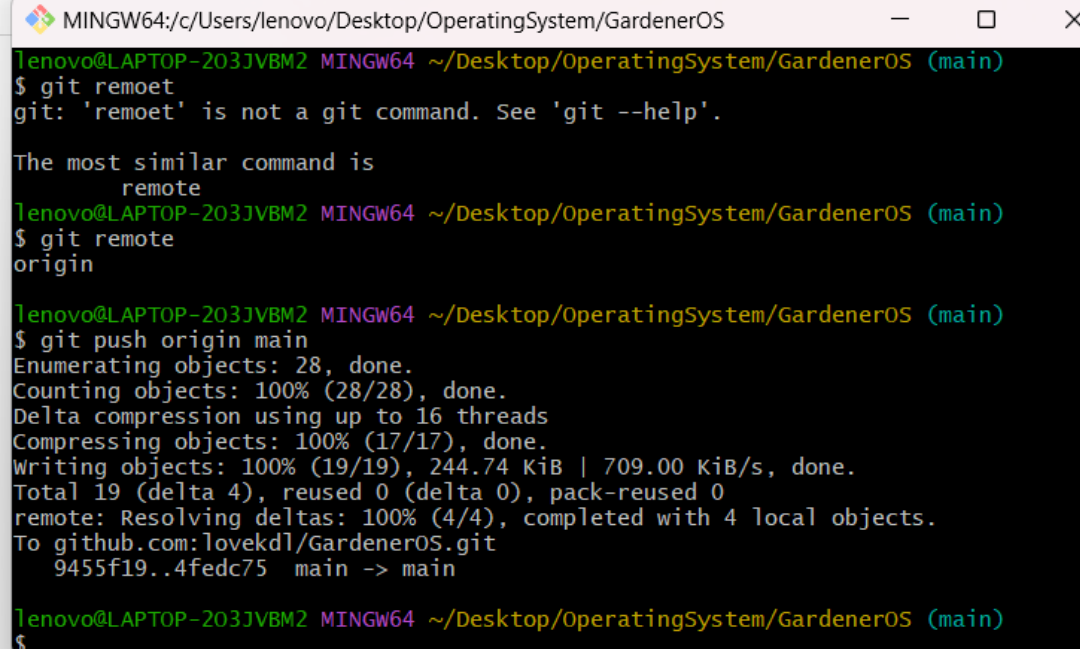
- 安装相关软件包

```
[root@dcea8e5aca3c os]# rustup target add riscv64gc-unknown-none-elf
info: component 'rust-std' for target 'riscv64gc-unknown-none-elf' is up to date
[root@dcea8e5aca3c os]# cargo install cargo-binutils
Updating 'rustc' index
Ignored package 'cargo-binutils v0.3.6' is already installed, use --force to override
[root@dcea8e5aca3c os]# rustup component add llvm-tools-preview
info: component 'llvm-tools' for target 'x86_64-unknown-linux-gnu' is up to date
[root@dcea8e5aca3c os]# rustup component add rust-src
info: component 'rust-src' is up to date
[root@dcea8e5aca3c os]#
```

- [root@dcea8e5aca3c os]# cargo build
Finished dev [unoptimized + debuginfo] target(s) in 0.11s
[root@dcea8e5aca3c os]#

- 这部分的git提交

```
lenovo@LAPTOP-203JVB2 MINGW64 ~/Desktop/OperatingSystem/GardenerOS (main)
$ git commit -m "21301032drj"
On branch main
nothing to commit, working tree clean
```



```
MINGW64:/c/Users/lenovo/Desktop/OperatingSystem/GardenerOS
lenovo@LAPTOP-203JVB2 MINGW64 ~/Desktop/OperatingSystem/GardenerOS (main)
$ git remoet
git: 'remoet' is not a git command. See 'git --help'.

The most similar command is
    remote
lenovo@LAPTOP-203JVB2 MINGW64 ~/Desktop/OperatingSystem/GardenerOS (main)
$ git remote
origin

lenovo@LAPTOP-203JVB2 MINGW64 ~/Desktop/OperatingSystem/GardenerOS (main)
$ git push origin main
Enumerating objects: 28, done.
Counting objects: 100% (28/28), done.
Delta compression using up to 16 threads
Compressing objects: 100% (17/17), done.
Writing objects: 100% (19/19), 244.74 KiB | 709.00 KiB/s, done.
Total 19 (delta 4), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (4/4), completed with 4 local objects.
To github.com:lovekd1/GardenerOS.git
9455f19..4fedc75 main -> main

lenovo@LAPTOP-203JVB2 MINGW64 ~/Desktop/OperatingSystem/GardenerOS (main)
$
```

- 执行命令分析移除标准库后的独立可执行程序。

```
@dcea8e5aca3c/mnt/os
[root@dcea8e5aca3c os]#
[root@dcea8e5aca3c os]#
[root@dcea8e5aca3c os]#
[root@dcea8e5aca3c os]#
[root@dcea8e5aca3c os]# file target/riscv64gc-unknown-none-elf/debug/os
target/riscv64gc-unknown-none-elf/debug/os: ELF 64-bit LSB executable, UCB RISC-V, RVC, double-float ABI, version 1
(SYSV), statically linked, with debug_info, not stripped
[root@dcea8e5aca3c os]# rust-readobj -h target/riscv64gc-unknown-none-elf/debug/os

File: target/riscv64gc-unknown-none-elf/debug/os
Format: elf64-littleriscv
Arch: riscv64
AddressSize: 64bit
LoadName: <Not found>
ElfHeader {
  Ident {
    Magic: (7F 45 4C 46)
    Class: 64-bit (0x2)
    DataEncoding: LittleEndian (0x1)
    FileVersion: 1
    OS/ABI: SystemV (0x0)
    ABIVersion: 0
    Unused: (00 00 00 00 00 00 00)
  }
  Type: Executable (0x2)
  Machine: EM_RISCV (0xF3)
  Version: 1
  Entry: 0x0
  ProgramHeaderOffset: 0x40
  SectionHeaderOffset: 0x1B10
  Flags [ (0x5)
    EF_RISCV_FLOAT_ABI_DOUBLE (0x4)
    EF_RISCV_RVC (0x1)
  ]
  HeaderSize: 64
  ProgramHeaderEntrySize: 56
  ProgramHeaderCount: 4
  SectionHeaderEntrySize: 64
  SectionHeaderCount: 14
  StringTableSectionIndex: 12
[root@dcea8e5aca3c os]#
[root@dcea8e5aca3c os]# rust-objdump -S target/riscv64gc-unknown-none-elf/debug/os

target/riscv64gc-unknown-none-elf/debug/os:   file format elf64-littleriscv
[root@dcea8e5aca3c os]# cargo run
    Finished dev [unoptimized + debuginfo] target(s) in 0.11s
    Running `target/riscv64gc-unknown-none-elf/debug/os`
Segmentation fault
[root@dcea8e5aca3c os]#
```

1.3 用户态可执行的环境

- 增加入口函数

```
@dcea8e5aca3c/mnt/os/src
1 #![no_std]
2 #![no_main]
3
4 use core::panic::PanicInfo;
5
6 #[panic_handler]
7 fn panic(_info: &PanicInfo) -> ! {
8     loop {}
9 }
10
11 #[no_mangle]
12 extern "C" fn _start() {
13     loop{};
14 }
15
-- INSERT --
15,1 A11
```

- 编译运行

```
@dcea8e5aca3c/mnt/os
root@dcea8e5aca3c os]# cargo run
   Compiling os v0.1.0 (/mnt/os)
   Finished dev [unoptimized + debuginfo] target(s) in 0.55s
   Running target/riscv64gc-unknown-none-elf/debug/os
^C
root@dcea8e5aca3c os]#
root@dcea8e5aca3c os]#
root@dcea8e5aca3c os]# cargo build
   Finished dev [unoptimized + debuginfo] target(s) in 0.12s
root@dcea8e5aca3c os]# qemu-riscv64 target/riscv64gc-unknown-none-elf/debug/os
```

- 去掉loop尝试

```
@dcea8e5aca3c/mnt/os/src
1  #![no_std]
2  #![no_main]
3
4  use core::panic::PanicInfo;
5
6  #[panic_handler]
7  fn panic(info: &PanicInfo) -> ! {
8      loop {}
9  }
10
11 #[no_mangle]
12 extern "C" fn _start() {
13     // loop();
14 }
15
-- INSERT --
14,2 All

@dcea8e5aca3c/mnt/os
root@dcea8e5aca3c os]# cargo run
   Compiling os v0.1.0 (/mnt/os)
   Finished dev [unoptimized + debuginfo] target(s) in 0.55s
   Running target/riscv64gc-unknown-none-elf/debug/os
^C
root@dcea8e5aca3c os]#
root@dcea8e5aca3c os]#
root@dcea8e5aca3c os]# cargo build
   Finished dev [unoptimized + debuginfo] target(s) in 0.12s
root@dcea8e5aca3c os]# qemu-riscv64 target/riscv64gc-unknown-none-elf/debug/os
^C
root@dcea8e5aca3c os]# cd src
root@dcea8e5aca3c src]# vim main.rs
root@dcea8e5aca3c src]# cd ..
root@dcea8e5aca3c os]# cargo build
   Compiling os v0.1.0 (/mnt/os)
   Finished dev [unoptimized + debuginfo] target(s) in 0.53s
root@dcea8e5aca3c os]# qemu-riscv64 target/riscv64gc-unknown-none-elf/debug/os
Segmentation fault
root@dcea8e5aca3c os]#
```

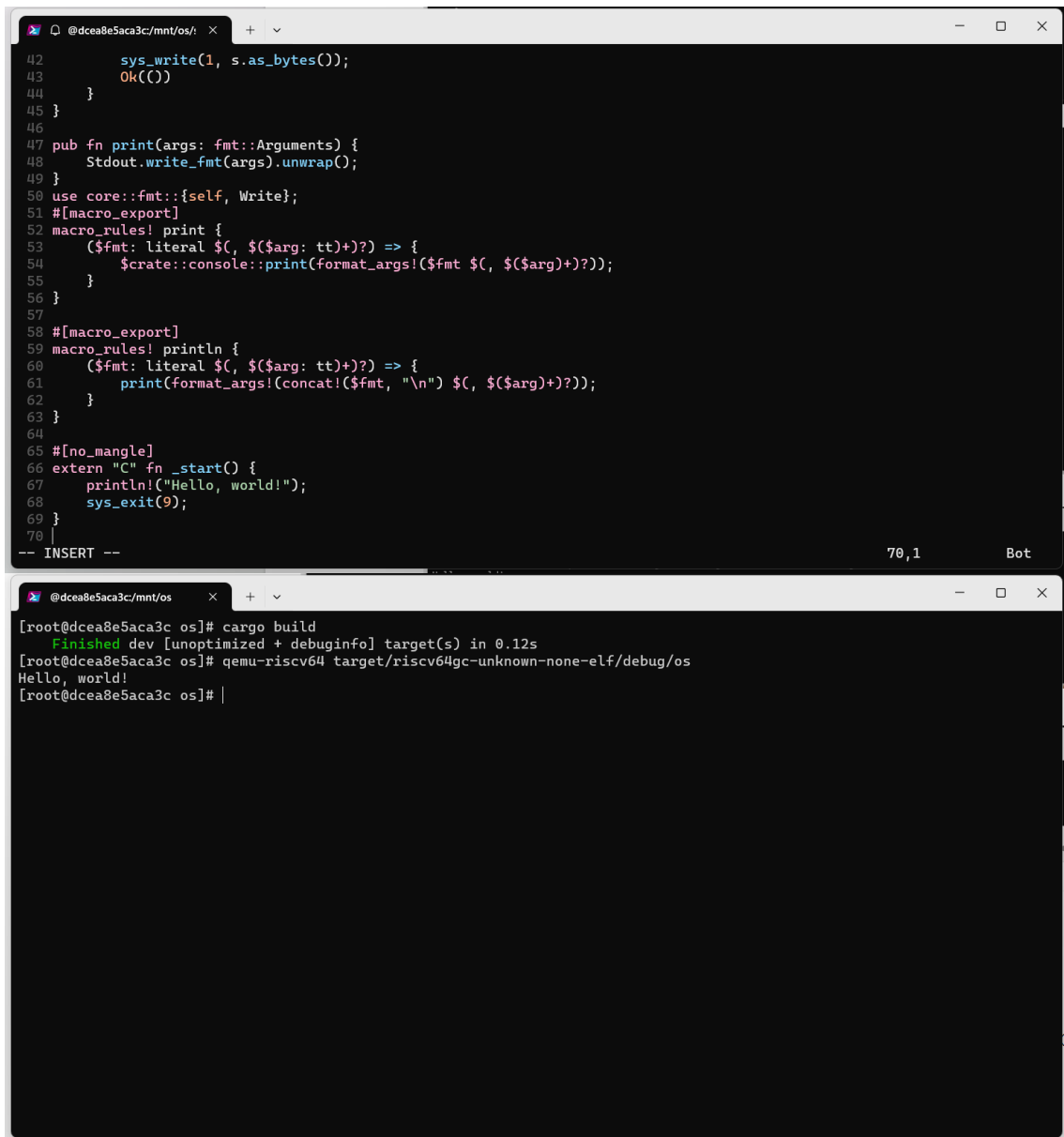
- 实现退出机制

```
@dcea8e5aca3c/mnt/os/src x + v
8     loop {}
9 }
10 use core::arch::asm;
11
12 const SYSCALL_EXIT: usize = 93;
13
14 fn syscall(id: usize, args: [usize; 3]) -> isize {
15     let mut ret: isize;
16     unsafe {
17         asm!("ecall",
18             in("x10") args[0],
19             in("x11") args[1],
20             in("x12") args[2],
21             in("x17") id,
22             lateout("x10") ret
23         );
24     }
25     ret
26 }
27
28 pub fn sys_exit(xstate: i32) -> isize {
29     syscall(SYSCALL_EXIT, [xstate as usize, 0, 0])
30 }
31
32 #[no_mangle]
33 extern "C" fn _start() {
34     sys_exit(9);
35 }
36 |
```

36, 0-1 Bot

```
@dcea8e5aca3c/mnt/os x + v
[root@dcea8e5aca3c os]# cargo build
  Compiling os v0.1.0 (/mnt/os)
  Finished dev [unoptimized + debuginfo] target(s) in 0.52s
[root@dcea8e5aca3c os]# qemu-riscv64 target/riscv64gc-unknown-none-elf/debug/os
[root@dcea8e5aca3c os]# cargo run
  Finished dev [unoptimized + debuginfo] target(s) in 0.11s
  Running `target/riscv64gc-unknown-none-elf/debug/os`
[root@dcea8e5aca3c os]#
```

- 实现输出支持



The image shows two terminal windows. The top window displays Rust source code for a program that prints 'Hello, world!'. The code includes a `main` function that calls `println!`, a `print` function, and a `start` function that calls `println!` and `sys_exit(9)`. The bottom window shows the output of running `cargo build` and `qemu-riscv64`, resulting in the output 'Hello, world!'.

```
42     sys_write(1, s.as_bytes());
43     Ok(())
44 }
45 }
46
47 pub fn print(args: fmt::Arguments) {
48     Stdout.write_fmt(args).unwrap();
49 }
50 use core::fmt::{self, Write};
51 #[macro_export]
52 macro_rules! print {
53     ($fmt: literal $(, $($arg: tt)+)? ) => {
54         $crate::console::print(format_args!($fmt $(, $($arg)+)?));
55     }
56 }
57
58 #[macro_export]
59 macro_rules! println {
60     ($fmt: literal $(, $($arg: tt)+)? ) => {
61         print(format_args!(concat!($fmt, "\n") $(, $($arg)+)?));
62     }
63 }
64
65 #[no_mangle]
66 extern "C" fn _start() {
67     println!("Hello, world!");
68     sys_exit(9);
69 }
70 |
-- INSERT --
```

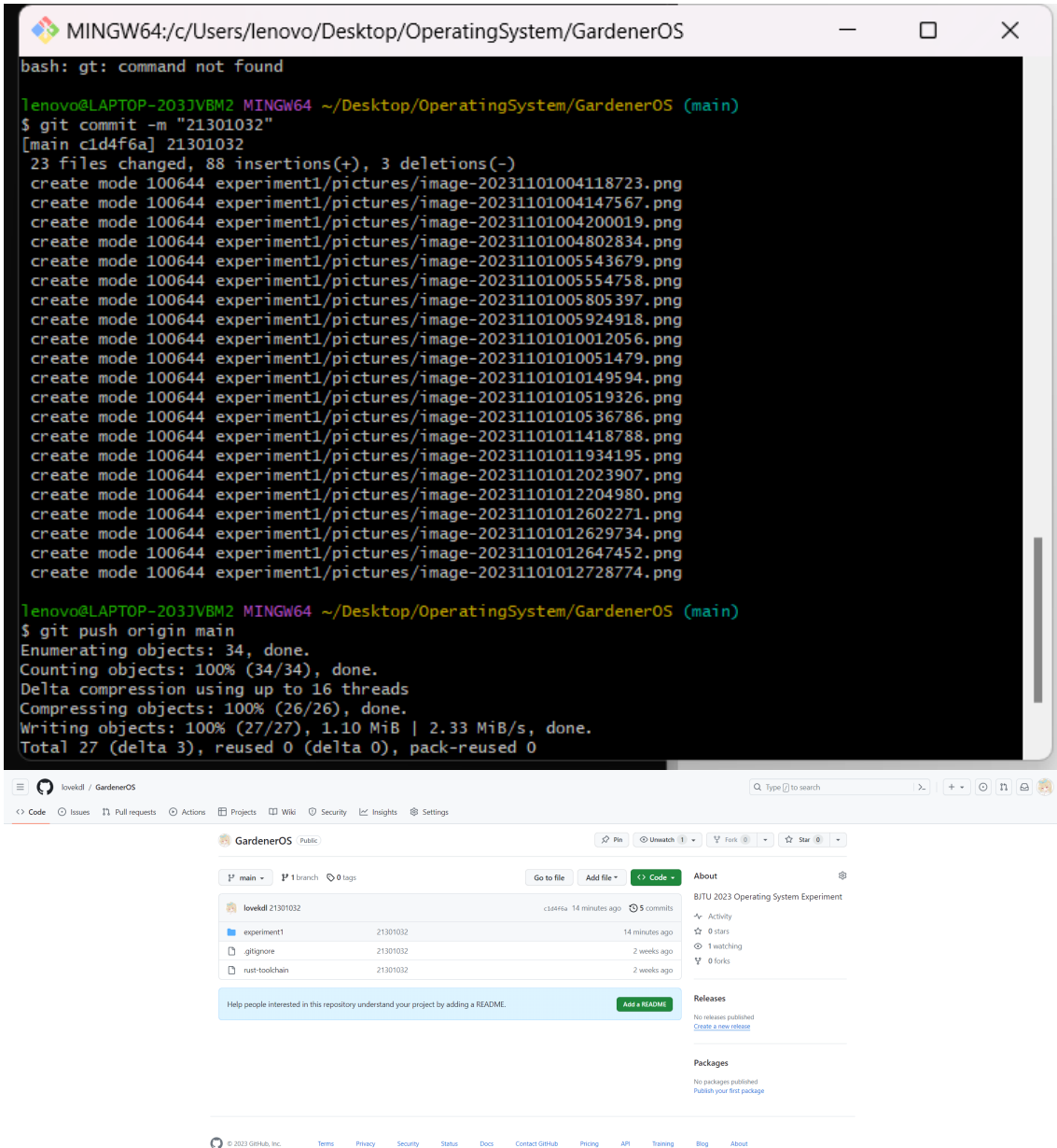
```
[root@dcea8e5aca3c os]# cargo build
Finished dev [unoptimized + debuginfo] target(s) in 0.12s
[root@dcea8e5aca3c os]# qemu-riscv64 target/riscv64gc-unknown-none-elf/debug/os
Hello, world!
[root@dcea8e5aca3c os]#
```

二、思考问题

- 为什么称最后实现的程序为独立的可执行程序，它和标准的程序有什么区别？
答：因为独立的可执行程序是源代码在经过编译处理最后形成的程序，具有独立性，不需要依赖到外部的库，通常可以直接运行，不需要额外的编译等操作。标准的程序需要依赖外部的库，文件体积相对较小，需要特定的运行环境。总的来说，标准的程序可以用于开发阶段，而独立的可执行程序移植性高，可直接运行。
- 实现和编译独立可执行程序的目的什么？
独立可执行程序移植性高，方便在环境不同的设备上部署，可以简化操作。独立可执行程序对外界环境依赖小，内部不易修改，所以安全性也很高。

三、git截图

- git提交(<https://github.com/lovekdl/GardenerOS>)



The image shows a Windows terminal window and a screenshot of the GitHub repository page for GardenerOS.

Terminal Window:

```
MINGW64:~/c:/Users/lenovo/Desktop/OperatingSystem/GardenerOS
bash: gt: command not found

lenovo@LAPTOP-203JVBM2 MINGW64 ~/Desktop/OperatingSystem/GardenerOS (main)
$ git commit -m "21301032"
[main c1d4f6a] 21301032
23 files changed, 88 insertions(+), 3 deletions(-)
create mode 100644 experiment1/pictures/image-20231101004118723.png
create mode 100644 experiment1/pictures/image-20231101004147567.png
create mode 100644 experiment1/pictures/image-20231101004200019.png
create mode 100644 experiment1/pictures/image-20231101004802834.png
create mode 100644 experiment1/pictures/image-20231101005543679.png
create mode 100644 experiment1/pictures/image-20231101005554758.png
create mode 100644 experiment1/pictures/image-20231101005805397.png
create mode 100644 experiment1/pictures/image-20231101005924918.png
create mode 100644 experiment1/pictures/image-20231101010012056.png
create mode 100644 experiment1/pictures/image-20231101010051479.png
create mode 100644 experiment1/pictures/image-20231101010149594.png
create mode 100644 experiment1/pictures/image-20231101010519326.png
create mode 100644 experiment1/pictures/image-20231101010536786.png
create mode 100644 experiment1/pictures/image-20231101011418788.png
create mode 100644 experiment1/pictures/image-20231101011934195.png
create mode 100644 experiment1/pictures/image-20231101012023907.png
create mode 100644 experiment1/pictures/image-20231101012204980.png
create mode 100644 experiment1/pictures/image-20231101012602271.png
create mode 100644 experiment1/pictures/image-20231101012629734.png
create mode 100644 experiment1/pictures/image-20231101012647452.png
create mode 100644 experiment1/pictures/image-20231101012728774.png

lenovo@LAPTOP-203JVBM2 MINGW64 ~/Desktop/OperatingSystem/GardenerOS (main)
$ git push origin main
Enumerating objects: 34, done.
Counting objects: 100% (34/34), done.
Delta compression using up to 16 threads
Compressing objects: 100% (26/26), done.
Writing objects: 100% (27/27), 1.10 MiB | 2.33 MiB/s, done.
Total 27 (delta 3), reused 0 (delta 0), pack-reused 0
```

GitHub Repository Page:

The screenshot shows the GitHub repository page for **GardenerOS**. The repository is public and has 1 branch (main) and 0 tags. The commit history shows a single commit by lovekdl at 21301032, 14 minutes ago, with 5 commits in total. The repository contains a directory named **experiment1** and two files: **gitignore** and **rust-toolchain**. The repository description is "BITU 2023 Operating System Experiment". The page also shows the "About" section, "Releases" section (no releases published), and "Packages" section (no packages published).