Operating Systems in Electronic Engineering: Exercise Guide for Python and QEMU

A Comprehensive Exploration of System Design and Functionality

Politecnico di Torino

Department of Electronic Engineering
Operating Systems

Academic Year 2023-2024

Exercise 1: Install Python on QEMU

Objective: To install Python on Buildroot for your QEMU environment, follow these steps on your host system. It is assumed that the first boot of QEMU has already been done as per another guide and that the QEMU directory is configured and ready to use.

1. **Access Buildroot Configuration:**
   - Open a terminal on your host system.
   - Navigate to your Buildroot directory.
   - Install the required libraries on Debian/Ubuntu-based systems using: `sudo apt-get install libncurses5-dev libncursesw5-dev`.
   - Run the command `make menuconfig`.
2. **Configure Buildroot for Python:**
   - In the `menuconfig` interface, go to `Toolchain` and enable `C++ support`.
   - Navigate to `Target Packages`.
   - Enter in the subsection `Interpreter Languages and Scripting`.
   - Select and enable `Python` or `Python3`.
   - Navigate to `Filesystem images` and then to the settings for the ext2 filesystem.
   - Increase the `Exact size of the filesystem` to at least 1G (`BR2_TARGET_ROOTFS_EXT2_SIZE`).
3. **Build the System:**
   - After configuring, exit the menu and save the configuration.
   - Run `make clean` to clean the previous builds.
   - Run `make` to start the build process with the new configuration.
   - In `buildroot/output/images`, modify `start-qemu.sh` to include the full path leading up to `qemu/build` before `qemu-system-riscv64`.

4. **Driver:**

   - Modify paths in the `Makefile` and `comando_make.sh` in the `'driver'` folder.
   - Execute the command in `'comando_make.sh'` to create the `'crypto-core.ko'` file.
   - Start a local HTTP server in the folder where the `.ko` file is located using `python3 -m http.server`.
   - Inside QEMU create a new folder (`mkdir myfiles`).
   - Inside the folder execute `wget http://yourIP:yourPort/crypto-core.ko`.
   - Use `chmod 777 crypto-core.ko` to change permissions.
   - Load the driver with `insmod crypto-core.ko`.

Exercise 2: Create an Interface to Interact with the Crypto-Core

Objective: Develop a Python program that interfaces with the crypto-core for easy operation handling.

Step 1: Create the C Programs

You'll create six different C programs for specific tasks:

1. Update Key Register (update_key.c)
2. Update IV Register (update_iv.c)
3. Insert New Plaintext (insert_plaintext.c)
4. Change Mode (change_mode.c)
5. Change Format (change_format.c)
6. Perform All Operations Together (perform_all_operations.c)

```c
#include "crypto_functions.h"

int main(int argc, char **argv) {
    if (argc != 2) {
        printf("Usage: %s [key]\n", argv[0]);
        return 1;
    }

    int fd = dev_file_open(DIR_KEY, O_WRONLY);
    write(fd, argv[1], strlen(argv[1]));
    close(fd);

    return 0;
}
```

Step 2: Create the Makefile

Create a Makefile to compile all your C programs with the following content:

```
CC=/path/to/compiler  # Replace with your compiler path

all: update_key update_iv insert_plaintext change_mode change_format
    perform_all_operations

update_key:
    $(CC) update_key.c crypto_functions.c -o update_key

update_iv:
    $(CC) update_iv.c crypto_functions.c -o update_iv

insert_plaintext:
    $(CC) insert_plaintext.c crypto_functions.c -o insert_plaintext

change_mode:
    $(CC) change_mode.c crypto_functions.c -o change_mode

change_format:
    $(CC) change_format.c crypto_functions.c -o change_format

perform_all_operations:
    $(CC) perform_all_operations.c crypto_functions.c -o
    perform_all_operations

clean:
    rm -f update_key update_iv insert_plaintext change_mode change_format
    perform_all_operations
```

Step 3: Compile the C Programs

Open a terminal in the directory containing your C files and Makefile, then run the following command:

```
make
```

Step 4: Create the Python Program

Create a Python script (e.g., crypto_operations.py) that provides a user interface to interact with your different C programs. The script might look like this:

```python
import subprocess

def show_menu():
    print("Select an operation:")
    print("1) Update the key register")
    print("2) Update the IV register")
    # ... more options
    print("exit) Exit the program")

def get_user_choice():
    return input("Enter your choice: ")

def call_c_program(program_name, *args):
    subprocess.run(["./" + program_name] + list(args), capture_output=True, text=True)

def main():
    while True:
        show_menu()
        choice = get_user_choice()
        if choice == 'exit':
            break
        elif choice == '1':
            # Call the function to handle this option
            # ...
        # ... other elif blocks for different choices

if __name__ == "__main__":
    main()
```