Lab: Processes, Linux Commands, and Program Execution

Outcomes

- Understand processes and context in a computing system.
- Recap essential Linux commands.
- Explore program execution.
- Analyze hardware resource usage during program execution.

Reference Links

- Basic Unix
- Unix Guru Universe (UGU)
- UNIX FAQ

Unix Commands Quick Reference

- **List files**: `ls -l`
- **Change directory**: `cd <directory>`
- **Manual for commands**: `man <command>` (e.g., `man ls`)

Task 1

Compile and Run assembly_example on an emulator:

- Copy the `assembly_example` code below and run on https://cpulator.01xz.net/?sys=arm:
    - Click `Compile and Load` to begin execution.
    - The Disassembly tab should appear automatically when code compiles successfully
    - Use `Step Into` to execute instructions step-by-step, at each step observe the changes to the Registers, Call stack, Trace and Counter windows (on the left) and as well as the Memory tab.
    - Restart the program by clicking `Restart and u`pdate the contents of registers `r0`, `r1`, and `r2` by clicking on them and entering 0. You can then step through the code again.
    - Note: You can change the format from hexadecimal to decimal unsigned

    Questions:

    - What is the name of the register pointing to the next instruction?
      Answer:
    - What changes were observed to the registers and to the memory after running the program? Note don't include any memory addresses that change on compilation.

Answers can be given as hexadecimal or unsigned integer but stick to one of them.

Answer:

assembly_example.s

Assembly code using the ARM architecture:

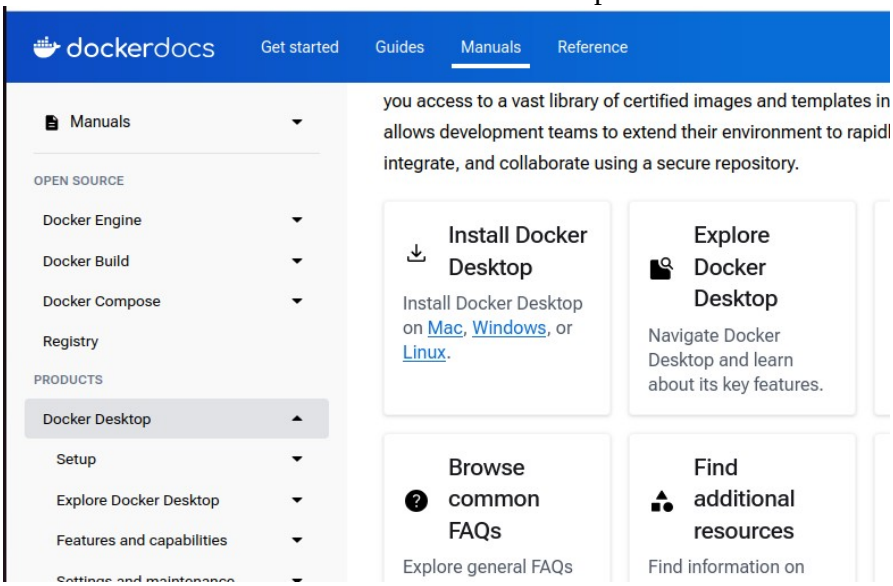
```
// assembly_example.s
.data             // Data section (not used here, but specified for structure)

.text             // Code section
.global _start    // Define the entry point of the program

_start:           // Main function
mov r1, #5        // Load the value 5 into register r1
mov r2, #26       // Load the value 26 into register r2
add r0, r1, r2    // Add r1 and r2 and store the result in r0

// Copy data to memory
ldr r3, =answer  // Assign address of answer to r3
str r2, [r3]     // Store contents of r2 to address in r3

// End program
_stop:
    b _stop

// Uninitiallized variables
.bss
answer:
.space 4    // Set aside 4 bytes for answer

.end
```

Task 2 (For Windows users)

Install docker desktop

https://docs.docker.com/desktop/

Scroll down and select Windows in Install Docker Desktop section



To check which you should install type

`%PROCESSOR_ARCHITECTURE%`

into your command line.

This will reveal whether you are running an x86_64 (AMD64 or Intel64) or ARM64 instruction set architecture so you can decide which option to choose.

Once docker is installed. Open the command line and type

`docker pull alpine`

This will pull an alpine linux image into docker so you can run linux commands in a linux container.

To start the container type

```
docker run -it --rm alpine /bin/ash
```

continue tasks .....

Note you will not be able to view man pages directly from the terminal as they are not included in the alpine image.

Observing Processes with `top`

- Open a terminal and type:

  `top`

- Observe the **S** (states/ process status) column. S, R, I etc.

  (Note: type q to exit top)

- Look up S -- the process statuses in the top manual.

  ```
  man top <enter>
  / process status <enter>
  ```

- **Copy and paste them into your lab document**

Checking System Details

Number of Processors

- In `top`, type `1` to display the number of processors.
    - How many processors are there? Submit a screenshot of the cpus with your answer.
      Answer:
- Use the following command to confirm:

  ```
  cat /proc/cpuinfo
  ```

- Count the number of tasks (processes) running. Submit a screenshot of process statuses to confirm.
  Answer:

Tasks and Multitasking

- From `top`, note the number of tasks and processors.
- Explain how the OS manages more tasks than processors using multitasking.
  Answer:

Using `ps` Instead of `top`

- Use `ps` commands to track and manage processes. How is it different from `top`?
  Answer:

Exploring `/proc` Directory

Process Details in `/proc`

- Navigate to the `/proc` directory:

  ```
  cd /proc
  ```

- What information does `/proc/<PID>` provide?
  Answer:

Manual Exploration

- Use `man proc` and explore the following:
    - Process states
    - `cmdline`
    - `stat`

Summarize your findings.
Answer:
