Lab BSY PROC-1

Introduction & Prerequisites

This laboratory is to learn how to:

- List and interact with processes
- Control jobs in background
- Create processes and threads
- Identify kernel threads

The following resources and tools are required for this laboratory session:

- A ZHAW VPN session
- Any modern web browser
- Any modern SSH client application
- OpenStack Horizon dashboard: https://ned.cloudlab.zhaw.ch
- OpenStack account details: please contact the lab assistant in case you already have not received your access credentials.
- Username to login with SSH into VMs in ned.cloudlab.zhaw.ch OpenStack cloud from your laptops
    - `Ubuntu`

Time

The entire session will take 90 minutes.

Assessment

No assessment foreseen

Task 1 – Programs and Processes

Subtask 1.1 – Processes Basics

Get an overview about the running processes on your Lab Instance, using a non-interactive and an interactive tool on the command line.

Using "top", explain the elements of the "Summary Display"

Which process has PID 1 and PID 2? Who is the owner of these processes?

What is the PID of your terminal? Who is the owner?

Show the Environmental Variables of your terminal. Add a variable named Name and assign your surname to it.

Study the "tail" program and find out how to monitor the evolution of a file in real-time. Use this tool to monitor system information available in the following file: /var/log/syslog

Start a second terminal and observe the output of your monitoring session. What do you notice?

On the second terminal, what is the PID and the state of your monitoring process (tail)?

Show the process hierarchy of your observer process (tail)

Terminal (kill) the terminal that runs the tail command and explain what happens.

Subtask 1.2 – Job Control

Study the "Job Control" section of the bash man page. Start three observer processes and put all of them into the background. Display the list of background jobs and explain the output.

Move job number 2 into foreground and press Ctrl + z. List again the job list. What is the difference now?

`Return each job to foreground and terminate it (Ctrl + c)`

Subtask 1.3 – Process Creation

Write a Hello World in c and create a process that executes your program.

Revisit the layout of a binary and print the sizes of sections “Text”, “Data” and “BSS”. What is the difference between the “Berkley” and the “Gnu” format?

Use objdump and display all section headers. Revisit the sections above and study the elements of each section. Disassemble the “Text” section.

Write a program that when running as a process, creates another process by “forking” (man fork) itself. Use the library function “sleep()” (man sleep) to put both processes to sleep for 30 seconds before they terminate. Check the memory usage of both processes and explain. You may find “getpid()” useful, see man getpid.

Subtask 1.4 – Zombie

Develop a program that creates a “Zombie Process”. Technically, when a child process terminates, the process’s parent is notified and supposed to execute the wait() (or waitpid()) system call to read the dead process’s exit status and other information. After wait() is called, the zombie process is completely removed by the OS.

Subtask 1.5 – Zombie or no Zombie

Modify your program such that the parent terminates before the child and explain what happens.

Task 2 – Multi-Threading

Study the following code and explain what happens, step-by-step. How can you (developer) define what should be shared between parent and child?

```
#define STACK_SIZE (1024 * 1024)  /* Stack size for cloned child */

static int childFunc(void *arg) {
        //do something
         return 0;
}

Int main(int argc, char *argv[]) {
        char *stack;
        char *stackTop;
        pid_t pid;

        if (argc < 2) {
                fprintf(stderr, "Usage: %s <child-hostname>\n", argv[0]);
                exit(EXIT_SUCCESS);
        }
        stack = mmap(NULL, STACK_SIZE, PROT_READ | PROT_WRITE,
                MAP_PRIVATE | MAP_ANONYMOUS | MAP_STACK, -1, 0);

        stackTop = stack + STACK_SIZE;

        pid = clone(childFunc, stackTop, CLONE_NEWUTS | SIGCHLD, argv[1]);

        if (waitpid(pid, NULL, 0) == -1)
                exit();

        printf("child has terminated\n");
        exit(EXIT_SUCCESS);
}
```

Task 3 - Kernel Threads

Which process has PID #2?

Identify any Kernel Thread and find out on which CPU it is running. Recall, the /proc directory is providing information about running processes of all kinds.

Select any thread called “Kworker” and explain the state by querying it with the ps command.

Cleanup

IMPORTANT: At the end of the lab session:

- **Delete** all - unused / no longer needed - OpenStack VMs that were created by your team.

Additional Documentation

OpenStack Horizon documentation can be found on the following pages:

- User Guide: https://docs.openstack.org/horizon/latest/