A picture containing graphical user interface, text

Description automatically generated

|  |
| --- |
| **WARNING** |
| **Possession of smart phones or other electronic devices during the exam is a violation of the Academic Integrity Policy and subjects the student to dismissal from ADU.** |
| **Signature of Students: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** |

|  |  |
| --- | --- |
|  | Student Name |
|  | Student ID No. |
| CSC 308 – Operating System | Course Code and Title |
| Dr. Mohamed ELHADAD | Course Instructor |
|  | Course Section |
|  | Date and Duration of the Exam |

**Part 1- Process**

Write a program that does the following:

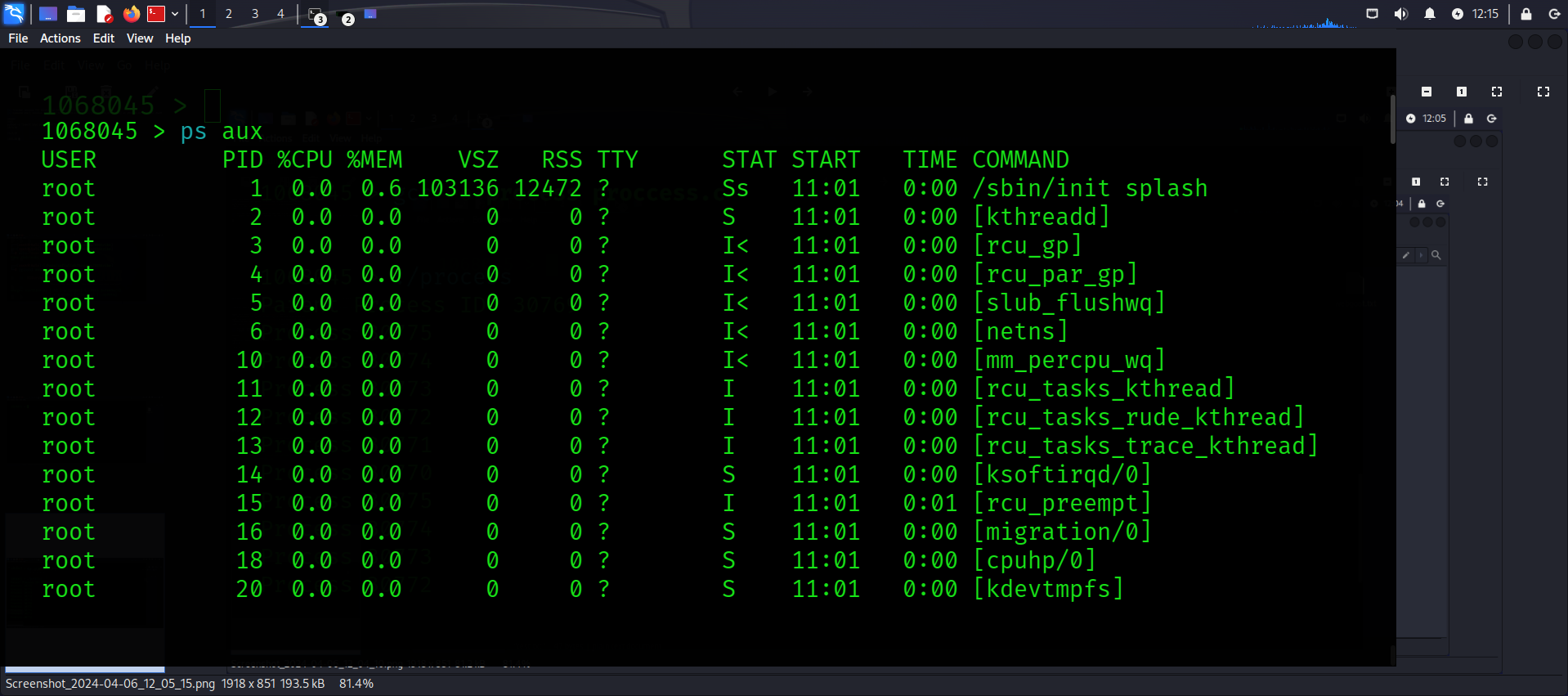
* When the program starts, it runs forever and it prints a message “process X” every 5 seconds. (X denotes the process ID of the current process, pls remember to replace X with your current process ID)
* The process creates its x child processes by using a fork and waits until they terminate, where x is the second digit of your student ID. If your student ID is 2345678, x=3: In the case when the second digit in your student ID is 0 or 1, please use the value x=4
* Each child process also runs forever and it prints a message “process X” every 5 seconds
* if a child process dies – the parent process creates it again
* run the program – report the result (screenshot)
* open a new terminal
* use ps aux to see all processes – look at the process id
* use kill -9 [pid] to kill one process
* look at the output to see whether it is restarted

Compile the Program: Use the gcc command to compile the process.c file into an executable named process. You can use the following command:

gcc -o process process.c

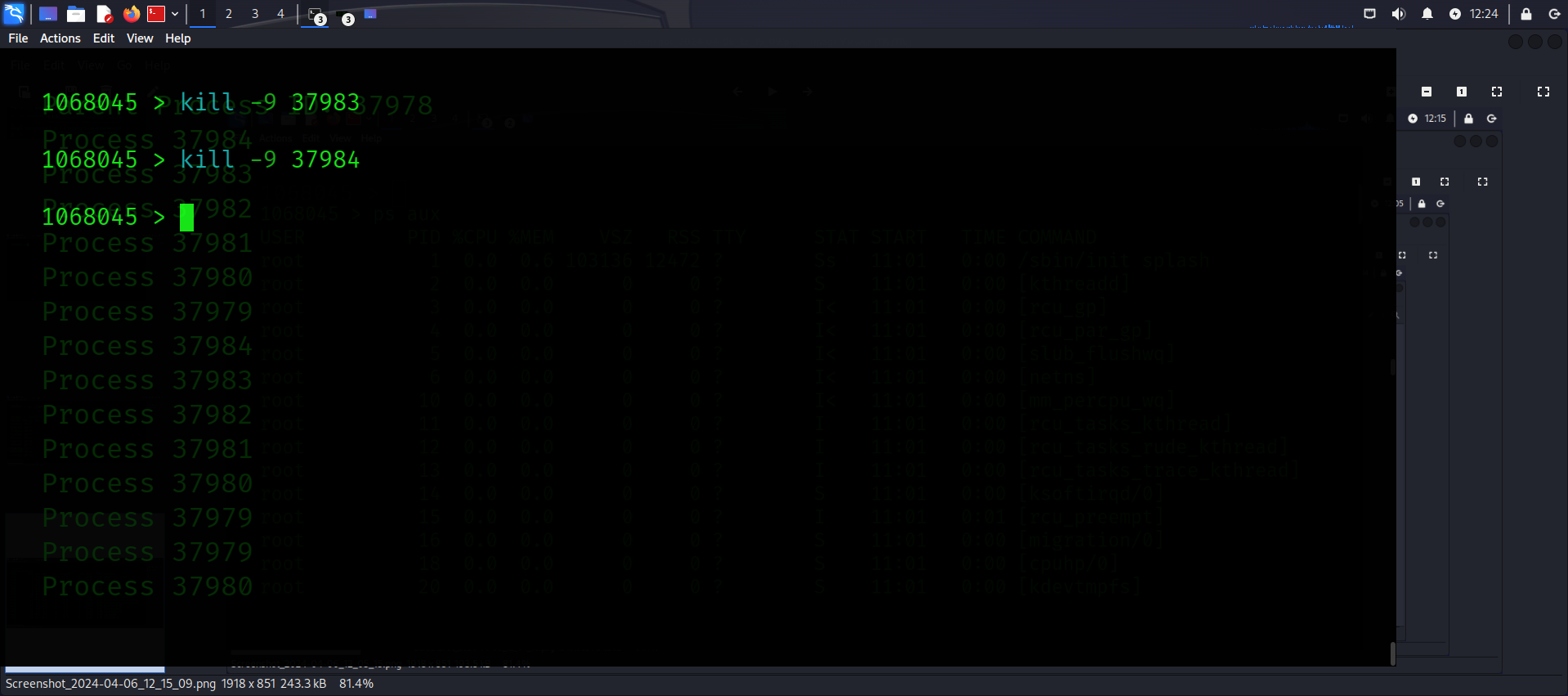
This command compiles the process.c file and creates an executable named process.

This will start the program, and it will run indefinitely, creating child processes and restarting them if they die.



View All Processes: i Use the ps aux command to view all processes. we see the process IDs (PIDs) of the process program and its child processes. Look for lines containing ./process to identify them.

Kill child processes



**Part 2- Process and Threads**

Write a program that does the following:

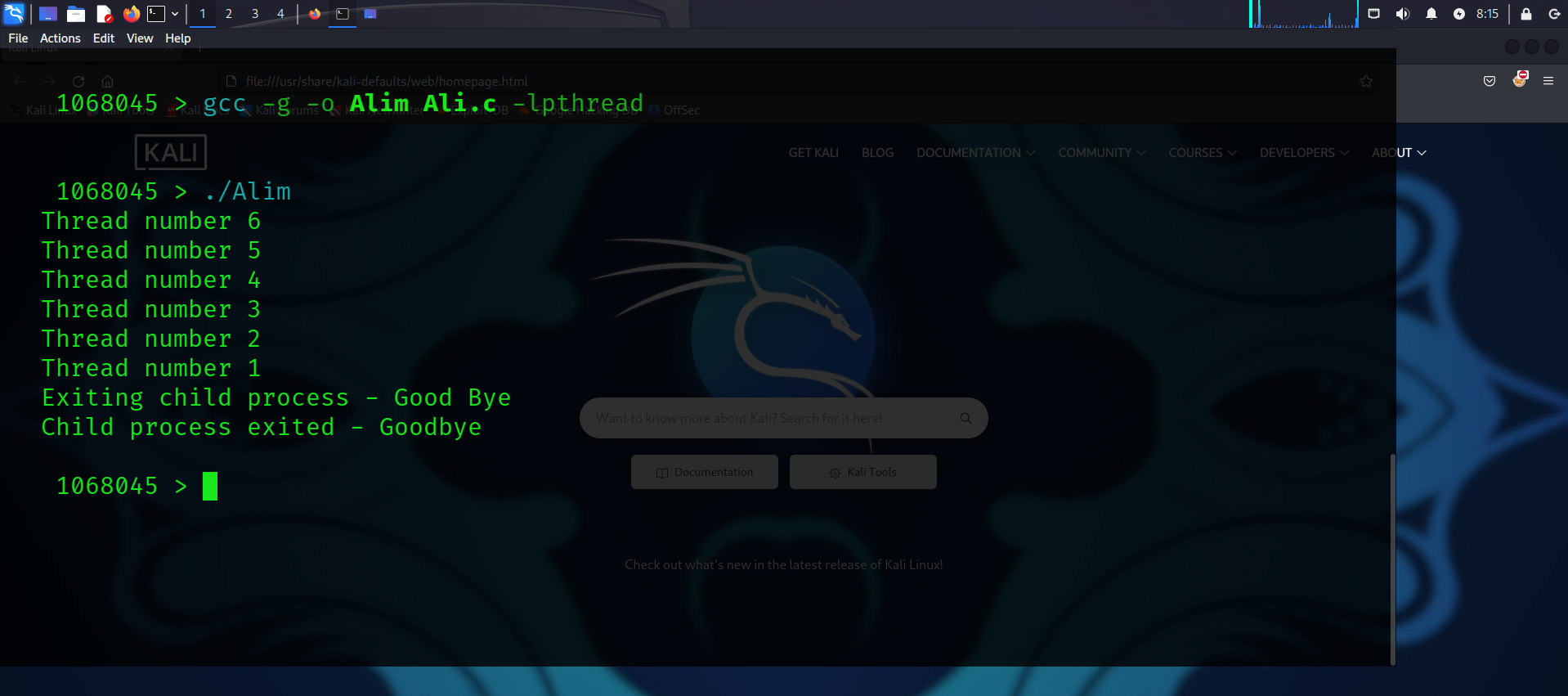
* When the program starts, it forks a new child process. The father process should wait for the child process to finish.
* The main thread of the child process should spawn y additional threads, where y is the third digit of your student ID. If your student ID is 2345678, y=4: In the case when the third digit in your student ID is 0 or 1, please use the value y=5
* Each thread should print: “Thread number X” and terminate.
* The main thread of the child process should wait for all its child threads to terminate. It should then print “Exiting child process - Good Bye” and exit.
* When the child process exists, the father should print “Child process exited- Goodbye” and exit

Compile the program

gcc -g -o Alim Ali.c -lpthread

This command compiles Ali.c and generates an executable named Alim with debugging information.

gdb ./Alim



**Methodology**

Forking a Child Process: The program starts by forking a new child process using the fork() system call. The parent process waits for the child process to finish using wait().

Thread Creation: The main thread of the child process spawns a number of additional threads based on the third digit of the student ID. If the third digit is 0 or 1, the program uses 5 threads; otherwise, it uses the third digit as the number of threads.

Thread Execution: Each thread prints a message indicating its thread number using printf() and then terminates.

Thread Synchronization: The main thread of the child process waits for all its child threads to terminate using pthread\_join().

Child Process Exit: After all threads have terminated, the main thread of the child process prints a "Exiting child process - Good Bye" message and exits.

Parent Process Exit: Once the child process exits, the parent process prints a "Child process exited - Goodbye" message and exits.

**Results**

For a student ID of 1068045, the third digit is 6. Therefore, the child process spawns 6 additional threads.

Each thread prints a message indicating its thread number.

The main thread of the child process waits for all 6 threads to terminate before printing the exit message and exiting.

The parent process waits for the child process to exit before printing its exit message and exiting.