CL2006 - Operating Systems Spring 2024
LAB # 5 Revision of Labs 1 to 4 (Common)

Please note that all labs' topics including pre-lab, in-lab and post-lab exercises are part of the theory and labsyllabus. These topics will be part of your Midterms and Final Exams of lab and theory.

Objectives:

Revision of Labs 1 to 4.

Lab # 1:

1. **Run the following commands:**

Command	Description
`pwd`	Display the current working directory.
`ls`	List files and directories.
`cd Desktop`	Change the current directory to Desktop.
`mkdir NewFolder`	Create a new directory named "NewFolder".
`rmdir NewFolder`	Remove the empty directory "NewFolder".
`cp file.txt /tmp`	Copy "file.txt" to the "/tmp" directory.
`mv file.txt newname.txt`	Rename "file.txt" to "newname.txt".
`rm oldfile.txt`	Remove the file "oldfile.txt".
`touch newfile.txt`	Create an empty file named "newfile.txt".
`nano myfile.txt`	Open the nano text editor for editing.
`cat myfile.txt`	Display the content of "myfile.txt".
`echo "Hello, Linux!"`	Display the message "Hello, Linux!".
`man ls`	Display the manual page for `ls`.
`chmod 755 script.sh`	Change the permissions of "script.sh".
`chown user:group myfile.txt`	Change the owner and group of "myfile.txt".
`ps aux`	Display information about active processes.
`kill 1234`	Terminate the process with PID 1234.
`top`	Display real-time system statistics.
`df -h`	Display disk space usage.
`du -h`	Display file and directory space usage.
`grep "pattern" file.txt`	Search for a pattern in "file.txt".
`find /home/user -name "*.txt"`	Search for ".txt" files in /home/user.
`tar -czvf archive.tar.gz folder/`	Create a compressed archive of "folder/".
`gzip file.txt`	Compress "file.txt" and replace it with "file.txt.gz".

2. **Type and save the following code as hello.cpp. Now compile it to generate assembly code using $ g++ -S hello.cpp**

```cpp
#include <iostream>

int main() {
    // Print "Hello, World!" to the console
    std::cout << "Hello, World!" << std::endl;

    // Return 0 to indicate successful completion
    return 0;
}
```

Lab # 2:

- Revised make file tutorial step by step in the lab (30 minutes)
- Explain Linux image compilation from source (10 minutes).

Lab # 3:

In-Lab

Consider the following log file data entries:

```
192.168.1.1 - - [01/Jan/2024:12:00:00 +0000] "GET /page1 HTTP/1.1" 200 1234
192.168.1.2 - - [01/Jan/2024:12:01:00 +0000] "GET /page2 HTTP/1.1" 404 5678
192.168.1.3 - - [01/Jan/2024:12:02:00 +0000] "GET /page1 HTTP/1.1" 200 9876
192.168.1.1 - - [01/Jan/2024:12:03:00 +0000] "POST /page3 HTTP/1.1" 301 2345
192.168.1.2 - - [01/Jan/2024:12:04:00 +0000] "GET /page2 HTTP/1.1" 200 8765
192.168.1.3 - - [01/Jan/2024:12:05:00 +0000] "GET /page4 HTTP/1.1" 404 3456
192.168.1.1 - - [01/Jan/2024:12:06:00 +0000] "GET /page1 HTTP/1.1" 200 6789
192.168.1.2 - - [01/Jan/2024:12:07:00 +0000] "GET /page2 HTTP/1.1" 200 1234
```

Now you are tasked with creating a shell script to analyze log files generated by a web server. The script should perform the following tasks:

- Accept a log file as input.
- Count the total number of requests (lines) in the log file.
- Determine the number of unique IP addresses that made requests.
- Identify the top 5 most frequent IP addresses and their corresponding request counts.
- Calculate the total size of data transferred (in bytes) from the log file.
- Generate a summary report containing the above information and save it to an output file.

Sample solution:

```
#!/bin/bash
# Log File Analyzer
# Accept log file as input
log_file="$1"
# Count total number of requests
total_requests=$(wc -l < "$log_file")
# Determine number of unique IP addresses
unique_ips=$(awk '{print $1}' "$log_file" | sort -u | wc -l)
# Identify top 5 most frequent IP addresses
top_ips=$(awk '{print $1}' "$log_file" | sort | uniq -c | sort -nr | head -5)
# Calculate total size of data transferred
total_size=$(awk '{sum += $10} END {print sum}' "$log_file")
# Generate summary report
echo "Log File Analysis Report" > analysis_report.txt
echo "------------------------" >> analysis_report.txt
echo "Total Requests: $total_requests" >> analysis_report.txt
echo "Unique IP Addresses: $unique_ips" >> analysis_report.txt
echo "Top 5 IP Addresses:" >> analysis_report.txt
echo "$top_ips" >> analysis_report.txt
echo "Total Size of Data Transferred: $total_size bytes" >> analysis_report.txt
echo "Report generated on: $(date)" >> analysis_report.txt
```

Lab # 4:

In-lab problems:

1. Write a program that creates two child processes. One child process should print its process ID (PID), and the other child process should print its parent process ID (PPID). The parent process should wait for both child processes to terminate before ending. **Hint: fork (), wait ()**

2. Create a program that creates a child process. The child process prints "I am a child process" 100 times in a loop. Whereas the parent process prints "I am a parent process" 100 times in a loop.

3. Develop a program that prints the current process ID (PID) and the parent process ID (PPID) of the running process. Additionally, implement a function to retrieve and print the user ID (UID) of the current user executing the program. **Hint: getpid (), getppid ()**

4. Create a program named stat that takes an integer array as command line argument (deliminated by some character such as $). The program then creates 3 child processes each of which does exactly one task from the following: i) Adds them and print the result on the screen. (done by child 1), ii) Shows the average on the screen. (done by child 2), iii) Prints the maximum number on the screen. (done by child 3)

5. Create a program that reads input from a text file named "input.txt" and writes the content to another text file named "output.txt". Ensure proper error handling for file opening, reading, and writing operations. **Hint: open (), read (), write (), close ()**

6. Invoke at least 4 commands from your programs, such as Cp, mkdir, rmdir, etc (The calling program must not be destroyed)

7. Write a program that demonstrates the usage of the fork() system call to create a child process. The child process should execute a shell command to list all files in the current directory, while the parent process waits for the child to terminate. Ensure proper synchronization between parent and child processes. **Hint: fork (), execlp ()**

8. Implement a program that utilizes the sleep()` and `alarm()` system calls. The program should initially set an alarm for 5 seconds and then enter a sleep loop. Within this loop, the program should print a message every second. When the alarm signal is received after 5 seconds, the program should terminate gracefully. **Hint: sleep (), alarm ()**