OS Lab Report – Week 1

PES1201800366

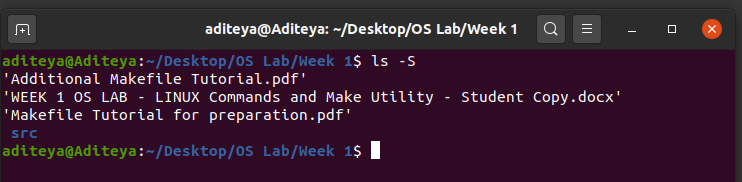
Aditeya Baral

1. Exercise 1 – Linux Commands

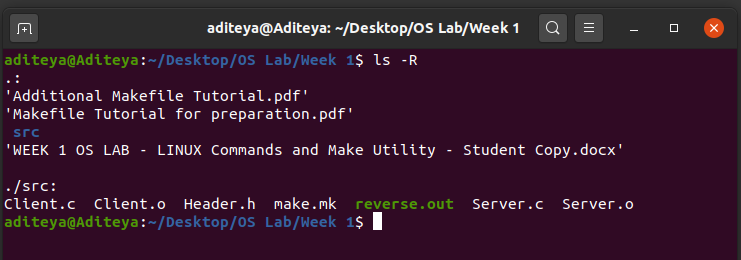
* Execute and familiarize with Linux environment and commands. Getting used to basic commands on Linux Operating System – Process creation, Process monitoring, Process states, Linux File system tree, Linux File system commands.
* Basic LINUX commands (ANY 5) executed in the lab should be submitted in the following way:
  + Command: What does the command do?
  + Any two options (i.e. flags or arguments) regarding the command
  + Outcome of the command

1.1 ls

* The **ls** command is a command-line utility for listing the contents of a directory or directories given to it via standard input and writes results to standard output.
* Flags can be provided to the command:
  + **-S**: this flag is used to sort the listed files and directories in decreasing order of size

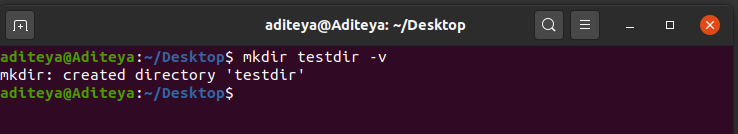


* + **-R**: this flag is used to display a recursive listing of all files and directories by traversing the entire path

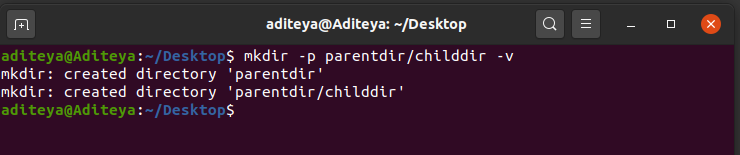


1.2 mkdir

* The **mkdir** command is a command line utility used to create a directory or even a directory structure given to it via standard input.
* Flags/Arguments can be provided to the command:
  + **dirname**: this argument specifies the name of the directory to be created

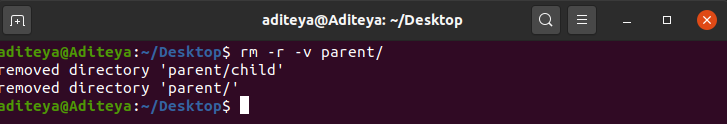


* + **-p**: this flag is used to create an entire directory path (with parent and child directories) if the parent directory does not exist

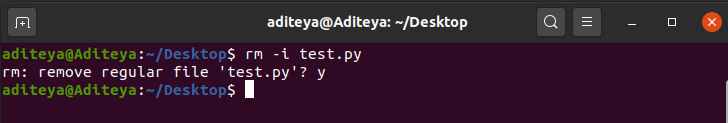


1.3 rm

* The **rm** command is a command line utility used to delete files and folders given to it via standard input
* Flags/Arguments can be provided to the command:
  + **-r**: This flag is used to recursively delete a directory and all its contents since rm does not delete directories by default

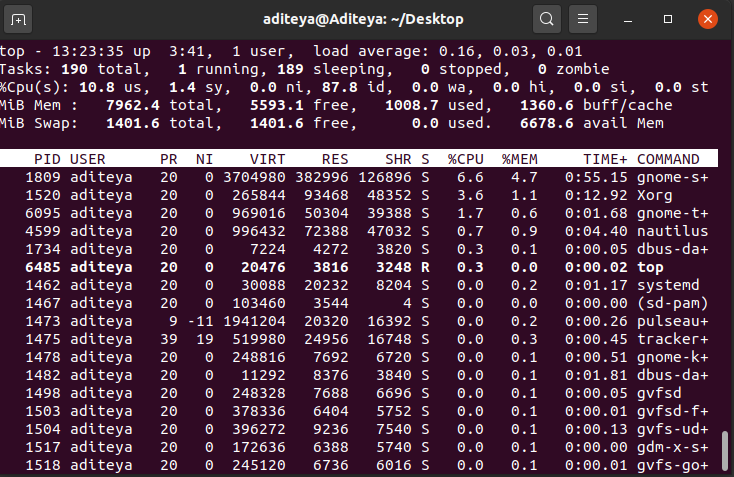


* + **-i**: This flag confirms with and asks the user before deleting each file



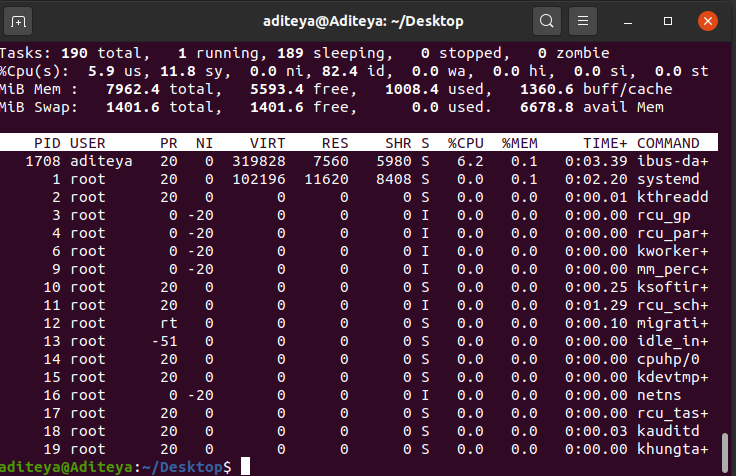
1.4 top

* The **top** command is used to show all the running processes on the system. It is like **ps** but provides an additional functionality of updating the processes at regular intervals.
* Flags can be provided to the command:
  + **-u**: This flag is used to show all the processes associated with a user.



top -u aditeya

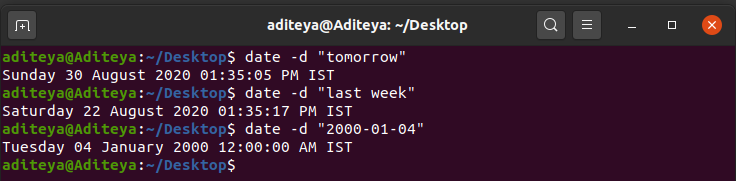
* + **-n**: This flag is used to exit the top command after a certain number of update intervals



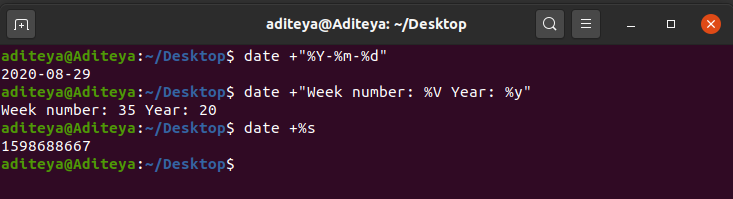
top -n 1

1.5 date

* The date command is a command line utility that is used to manipulate datetimes on Linux. It can be used to set dates, and perform operations using them.
* Flags/Arguments can be provided to the command:
  + **-d**: This flag is used to operate on a single date, which can also be provided in human readable format



* + **+format**: This argument is a format string and is used to return results in the date format specified



2. Exercise 2 – Makefile and Reversal of Array

* Write a C program to display an array in reverse using index. Create Makefile (ex: make.mk below) and other files as shown below:
  + Client.c – contains main function to collect input on array elements from the user and calls reverse\_array function
  + Server.c – contains reverse\_array function and prints the reversed array (use a separate function to print the reversed array)
  + Header.h – contains function prototypes
  + make.mk – contains targets and their dependencies
* Main program and all sub programs (dependency files, header file and Makefile) should be submitted. Steps to execute make and output of the program should be submitted.

2.1 Code

2.1.1 Client.c

1. #include "Header.h"
2. int main()
3. {
4. int n;
5. printf("Enter size of Array: ");
6. scanf("%d", &n);
8. int \*a = (int\*)calloc(n, sizeof(int));
10. printf("Enter elements: ");
11. for(int i = 0; i<n; i++)
12. scanf("%d", &a[i]);
14. printf("\nInput Array:\n");
15. display\_array(a, n);
17. printf("\nReversed Array:\n");
18. reverse\_array(a, n);
20. free(a);
21. return 0;
22. }

 2.1.2 Server.c

1. #include "Header.h"
2. void reverse\_array(int \*a, int n)
3. {
4. int temp, start = 0, end = n - 1;
5. while (start < end)
6. {
7. temp = a[start];
8. a[start] = a[end];
9. a[end] = temp;
10. start++;
11. end--;
12. }
13. display\_array(a, n);
14. }
16. void display\_array(int \*a, int n)
17. {
18. for (int i = 0; i < n; i++)
19. printf("%d ", a[i]);
20. printf("\n");
21. }

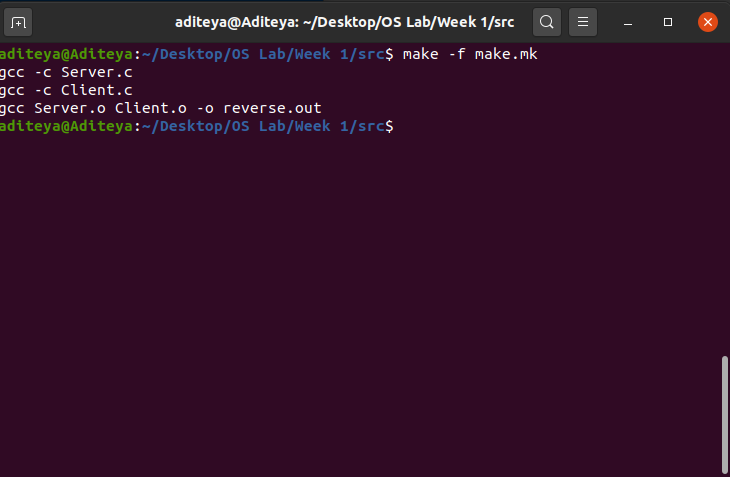
2.1.3 Make.mk

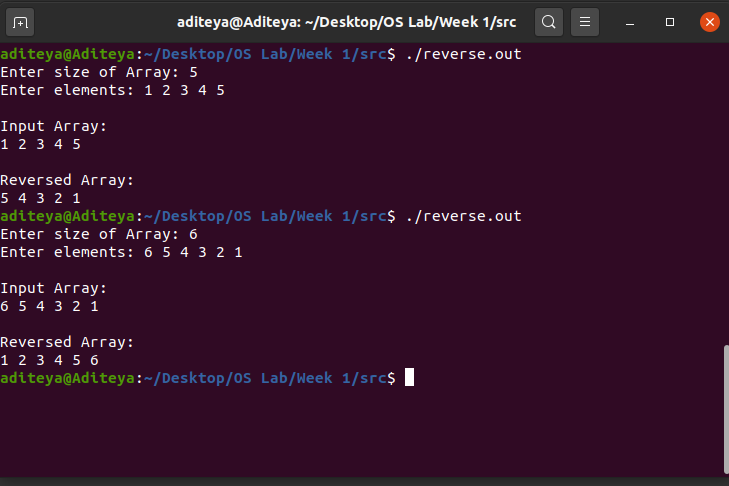
1. reverse.out: Server.o Client.o
2. gcc Server.o Client.o -o reverse.out
3. Server.o: Server.c Header.h
4. gcc -c Server.c
5. Client.o: Client.c Header.h
6. gcc -c Client.c
7. clean:
8. rm -rf \*.o
9. rm -rf \*.out

2.1.4 Header.h

1. #include <stdio.h>
2. #include <stdlib.h>
3. void display\_array(int \*, int);
4. void reverse\_array(int \*, int);

2.2 Screenshots





3. Exercise 3 – Questions

Answer the following questions (Brief answers only).

1. **Why do we use Makefile?**
   * A makefile is used to automate the software building procedure and other complex tasks which involve a lot of dependencies and requirements by issuing a set of rules or commands required which state when and how to compile a given source program. The makefile is read by the make utility, which understands the project structure and instructions and then performs and executes the commands stated in the makefile.
2. **Is Makefile a shell script?**
   * No, the makefile is not a shell script. The makefile is read by the make command, and not the shell program itself. Make however uses the shell the execute the commands listed for each dependency but is distinguished by its ability to use the modification information such as the edit time to select which dependencies to recompile.
3. **What does “clean” do in Makefile?**
   * “clean” is a phony target which is sometimes included in a makefile. A phony target can have any name and does not correspond to any target file. It is usually used to execute a list of commands before the execution of the target dependencies. “clean” is a conventional phony target name used to remove the outdated object files and executables before the source code is recompiled.
4. **How does make learn about the last modified files to be complied?**
   * make learns about modified files and other dependencies by inspecting the system’s modification meta information. If a certain target file’s modification time is older than any of its dependencies, it realises that the file has to be rebuilt or recompiled and follows through by executing the set of commands required to build the target file.
5. **What does Cflags in Makefile mean?**
   * Cflags is a macro defined in a makefile to specify the options that must be passed to the compiler during compilation of dependency files to create the target file. Macros are defined to create shortcuts to code in a makefile and serve to help programmers avoid repeating long text entries. Cflags is a conventional macro name used that is expanded into the full option string when make encounters it while executing the required commands.
6. **Why do we use -f option with make command?**
   * The make command reads or takes in a makefile and executes the required commands for each target file. By default, it looks for a makefile named “makefile.mk” but can also take a filename as an argument. This filename is provided to the make command utility via the -f option. Syntax: make -f <filename>.mk