Assignment 1A: (6 Marks) [CO1]
Familiarity with User and System level OS Commands

Run the commands and validate the output

1) who:-It displays the information about all the users who have logged into the system currently
2) whoami:- It displays Current username, Terminal number, date and time at which user logged into the system
3) pwd:- It displays current working directory
4) date:- It displays system date and time
5) ls - It lists the files and directories stored in the current directory. To list the files in a directory use the following syntax: $ls dirname
6) mkdir – It is used to create directories by using the command: $mkdir dirname
7) clear- It clears the screen
8) cd - It is used to change the current working directory to any other directory specified
9) df - It displays currently mounted file systems.
10) rmdir - Directories can be deleted using the **rmdir** command - $rmdir dirname
11) cat – It displays the contents of a file - $cat filename
12) cp - It is used to copy a file - $ cp source_file destination_file
13) mv- It is used to change the name of a file - $ mv old_file new_file
14) rm – It is used to delete an existing file - $ rm filename
15) stat- It is used to display file or file system status - $ stat filename
16) ln - It is used to create links between files and directories.
17) tty – It prints the filename of the terminal connected to standard input.
18) uname –It prints system information
19) umask – It specifies user file creation mask, implying which of the 3 permissions are to be denied to the owner,group and others.
20) find – It searches for files in a directory hierarchy
21) sort – It sorts the lines of text files
22) ps - It displays information about the current processes.
23) chmod 777 file1 - gives full permission to owner, group and others
24) grep - It finds specific patterns in files.
25) touch - It creates an empty file or updates the timestamp of an existing file.
26) more/less - It displays file content one screen at a time, allowing you to scroll.
27) head/tail - It displays First/Last 10 lines of a File.
28) top - It dynamically displays real-time information about system statistics.
29) kill - It terminates a Process.
30) history - It displays a list of previously executed commands.
31) du - It estimates file space usage (of a file or directory).
32) ping - It tests network connectivity to a host.
33) wc - It counts lines, words, and characters in a file.
34) >/>> - It redirects the standard output of a command to a file, overwriting the file if it exists.
35) | - It takes the standard output of one command and uses it as the standard input for another command.

Assignment 1B: (4 Marks) [CO1]
Program to get and set environment variables using system calls

Problem Definition

Program to GET and SET the Environment variable and to know the use of getenv and setenv system calls

UNIX ENVIRONMENT VARIABLES

Variables are a way of passing information from the shell to programs when you run them. Programs look "in the environment" for particular variables and if they are found will use the values stored. Some are set by the system, others by you, yet others by the shell, or any program that loads another program. Standard UNIX variables are split into two categories, environment variables and shell variables. In broad terms, shell variables apply only to the current instance of the shell and are used to set short-term working conditions; environment variables have a farther reaching significance, and those set at login are valid for the duration of the session. By convention, environment variables have UPPER CASE and shell variables have lower case names.

Display the following environment variables using getenv call:

- USER (your login name)
- HOME (the path name of your home directory)
- HOST (the name of the computer you are using)
- ARCH (the architecture of the computer's processor)
- DISPLAY (the name of the computer screen to display X windows)
- PRINTER (the default printer to send print jobs)
- PATH (the directories the shell should search to find a command)

Syntax: Char *getenv(const char *name);
The getenv() function searches the environment list to find the Environment variable *name*, and returns a pointer to the corresponding *value* string.

Now, Set two new environment variables and display them.

Syntax: int setenv(const char * envname,const char * enval,int overwrite)
The setenv() function adds the variable *name* to the environment with the value *value*, if *name* does not already exist. If *name* does exist in the environment, then its value is changed to *value* if *overwrite* is nonzero; if *overwrite* is zero, then the value of *name* is not changed (and setenv() returns a success status).

Note: make sure you don't modify the already generated system's environment variable like HOME, HOST etc.

ASSIGNMENT – 2
IPC/SYNCHRONIZATION

A. SIGNAL Handling (3 Marks) [CO3]

Catch the signal ‘SIGINT’ and display “Ha Ha, Not Stopping”. Use ‘signal’ system call. Always use “perror” to check the return status of a library/system call.

B. IPC using Named Pipe (FIFO) (7 Marks) [CO5]

Using the fork system call, create a Child Process.. Transfer 1GB file from the Parent Process to Child Process using a FIFO. Now, transfer the same file from the Child Process to the Parent Process using another FIFO. Now, compare the two files(use cmp or diff command) to make sure that the same file has returned back. Also, print the time required to do this double transfer. Attach this output to the source file as a comment.

To create FIFO, you can either use a shell command or a system call.

To create a large file you can use the relevant command.

Use ‘ls –l’ command to show the FIFO and the large file. Attach this output to the source file as a comment.

Make sure that the starting comment block has all the necessary information attached.

ASSIGNMENT – 3
Total Marks - 10 [CO4]
Parallel Programming in Python

The objective of this assignment is to write a parallel program in Python which will do Matrix Multiplication between two large Square Matrices with unsigned integer elements. You have to also measure the time elapsed. The Matrix should be large enough (at least 3000 x 3000) and dynamically allocated to fit in your computer memory. Both the Matrices need to be initialized with random numbers (‘mod’ to some value).

As you increase the number of Parallel Threads (max to your number of CPUs), your timing should show improvement. Use time.perf_counter() or an equivalent high-resolution timer to note down the timing. Make sure you attach the timing to the starting comment block of your source file. Your time should not include the Matrices initialization time.

You are not allowed to use Python’s threading module. You can import the package “multiprocessing” for this purpose. Also, use proper command to show CPU utilization(cpustat or sar) and attach these outputs to the starting comment block of your source file.

This program should take four command line arguments. The first argument is the dimension of the Square Matrix.

The second argument is about the number of Parallel Threads. For example, the value of the second argument will be 1, 2, 3, 4 etc.; signifying how the workload will be split among all the CPUs.

The third argument will be the value of the ‘mod’ which will be used to initialize all the elements of two input square matrices.

The fourth (last) argument will be the print_switch. If its value is ‘1’, both the Input and Output Matrices and as well as the Result Matrix will be printed on the screen. In case the value is ‘0’, Matrices won’t be printed on the screen.

ASSIGNMENT – 4
mmap and page fault

Total Marks - 10 [CO2]

The objective of this programming assignment is to use mmap() call and observe page-fault using the ‘sar’ command.

A big file (about 8GB) should be created using the ‘fallocate’ command. This big file should be written with a single byte value (say X) at a specific offset (say F). Both the values and the offset should be generated using a random function. Please do remember this random function should generate a quantity anywhere between 0 and 8G for the value of F and between 0-255 for the value of X.

The above big file should also be mapped in the virtual address space using mmap() call. Once it is mapped, the data should be read from the same specific offset (F). Now, if the data read is X`; then verify that X and X` are the same. In case of verification failure, an error message is to be printed and also the program should terminate. Note that, the offset value F can be anywhere between 0 and 8G and you should display the offset as hex number format.

This sequence of writing and reading data to/from a specific offset and also verification should be put in a while loop to go forever.

In another terminal execute the command ‘sar –B 1’ to observe for the page fault. This command should be started before the above program is put under execution. So, one can observe that the page faults are increasing, once the above program starts executing.

The output of the program and the ‘sar’ command should be pasted as a comment at the beginning of the program file as indicated by the guidelines.

ASSIGNMENT – 5
Total Marks - 10 [CO4 & CO5]
Thread, Synchronizations & Shared Memory

Consider a main process which creates three threads Th1, Th2, and Th3. The main process also creates two random quantities (X, Y), both less than 10. These two values will be placed by the main process in the shared memory (One variant of IPC Primitive) that is accessible by all the three threads Th1, Th2 and Th3. The shared memory will be created by the main process using shmat/shmget calls.

For each pair of values (X,Y), it is required that some computations should be done by various threads. The thread Th1 will compute A (X*Y), the thread Th2 will compute B (2*X+2*Y+1) and Th3 computes C (B/A). All these values are kept in the shared memory in a tabular fashion as shown below.

The number of (X,Y) pairs will be taken as an argument from the CLI. It is the responsibility of the main process to populate required numbers of (X,Y)s in the shared memory. The program will only exit when all A,B,C are computed for all given (X,Y) values. Before exiting, all (X,Y)s, As, Bs and Cs should be displayed as per the format shown below.

Whenever, the threads complete one phase of computations (A, B and C), they will go for another pair of (X,Y) values; but they will start all together. This can be achieved by proper synchronization.

Use the proper shell command(ipcs or simmilar) to display the Shared Memory Status/Info/Statistics and attach this sample output as a part of the starting comment block. This statistics should show the Shared Memory which your program has created.

Example:—
Input: *N, number of random pairs*

Output Format:

```
Pairs(X,Y)   |    A    |    B    |    C
  ------------------------------------------------
   (1, 2)    |    2    |    7    |    3.5
  ------------------------------------------------
   (4, 1)    |    4    |    11   |    2.75
 .........
```

ASSIGNMENT – 6
Total Marks - 10 [CO4]
Parallel Programming using THREAD

The objective of this assignment is to use thread programming technique to write a parallel program which will do Matrix Multiplication between two large Square Matrices with unsigned character elements. You have to also measure the time elapsed. The Matrix should be large enough (at least 3000 x 3000) and dynamically allocated to fit in your computer memory. Both the Matrices need to be initialized with random numbers (‘mod’ to some value).

As you increase the number of threads(max to your number of CPUs), your timing should show improvement. Use ‘gettimeofday’ to note down the timing. Make sure you attach the timing to the starting comment block of your source file. Your time should not include the Matrices initialization time.

You have to use various ‘Pthread’ library calls to do this assignment properly. Use the proper process listing command (ps) to show (and attach it to the starting comment block of your source file) that your threads are running in the system. Also, use proper command to show CPU utilization(cpustat or sar) and attach these outputs to the starting comment block of your source file.

This program should take four command line arguments. The first argument is the dimension of the Square Matrix.

The second argument is about the number of threads. For example, the value of the second argument will be 1, 2, 3, 4 etc.; signifying total no of threads which will be created.

The third argument will be the value of the ‘mod’ which will be used to initialize all the elements of two input square matrices.

The fourth (last) argument will be the print_switch. If its value is ‘1’, both the Input and Output Matrices and as well as the Result Matrix will be printed on the screen. In case the value is ‘0’, Matrices won’t be printed on the screen.

ASSIGNMENT – 7
Total Marks - 10 [CO4]
Avoiding DEADLOCK using Thread Programming

The objective of this assignment is to avoid deadlock. For this purpose define three global variables (Total_1, Total_2 and Total_3) and initialize all of them to 100000. You should also have three mutexes to protect these three global variables. You need to create four threads also (Th1, Th2, Th3 and Th4).

The function of thread Th1 is to generate a random quantity (not more than 10) and subtract that quantity from Total_1 and add that quantity to the Total_2 or Total_3 (deciding randomly). Likewise, The function of thread Th2 is to generate a random quantity (not more than 20) and subtract that quantity from Total_2 and add that quantity to the Total_1 or Total_3 (deciding randomly). Lastly, The function of thread Th3 is to generate a random quantity (not more than 30) and subtract that quantity from Total_3 and add that quantity to the Total_1 or Total_2 (deciding randomly). PLease remember, while manipulating the Totals, the concerned thread should lock all the relevant mutexes and then unlock it after changing the desired Totals.

The function of thread Th4 is to keep displaying the individual Totals and Grand Total (sum of Total_1, Total_2 and Total_3). Make sure the displayed values are consistent.

Please note that this program should run forever.

The order of locking and unlocking the Mutexs are very important, because that's what avoids a Deadlock or creates one. Once you correctly identify the order for avoiding Deadlock, you should upload the program. Also, include that information (as a comment in your source file) regarding the order for creating Deadlock.

Make sure there are enough printf in your program so that it can be clearly understood that there is no Deadlock.