# Lab-1(4-7 March 2021)

1. Write Linux C Program for command line argument to print the output in following format.

```
Hello, welcome to Linux C programming!
No. of argument to the main Program: 5
Argument No. 0 : ./filename
Argument No. 1 : My
Argument No. 2 : name
Argument No. 3 : is
Argument No. 4 :Rahul
```

2. Write Linux C Program to initialize character, integer, String and float. Each data declaration will occupy an amount of memory in byte. Print the output in following pattern

```
Town name: Guildford population: 66773
County: Survey
Country: Great Britain
Location Latitude: 51.238598 longitude: -0.566257
Char = 1 byte int= 4 bytes float = 4 bytes
Memory used: 780 bytes
```

   Hint: Use sizeof() library function

3. Write Linux C Program to print the entire environment variable on the screen.

4. Write Linux C Program to implement mergesort (Find the Sample from Help).

# Process Creation and Management

```
top       # view top consumers of memory and CPU (press 1 to see per-CPU statistics)
who       # Shows who is logged into system
w         # Shows which users are logged into system and what they are doing
ps        # Shows processes running by user
ps -e     # Shows all processes on system; try also '-a' and '-x' arguments
ps aux | grep<user_name> # Shows all processes of one user
ps ax --tree      # Shows the child-parent hierarchy of all processes
ps -o %t -p <pid> # Shows how long a particular process was running.
          # (E.g. 6-04:30:50 means 6 days 4 hours ...)
Ctrl z <enter>    # Suspend (put to sleep) a process
fg            # Resume (wake up) a suspended process and brings it into foreground
bg            # Resume (wake up) a suspended process but keeps it running
              # in the background.
Ctrl c        # Kills the process that is currently running in the foreground
kill <process-ID> # Kills a specific process
kill -9 <process-ID> # NOTICE: "kill -9" is a very violent approach.
           # It does not give the process any time to perform cleanup procedures.
kill -l       # List all of the signals that can be sent to a proccess
kill -s SIGSTOP <process-ID> # Suspend (put to sleep) a specific process
```

1. Write a program that fork once. Find out what happens when fork returns? Also find out what the parent gets as the return value of the fork and what the child gets as the return value of the same call.

2. Write a program that, forks to create a child process. The child exits with an exit status of 10 and the parent waits for the death of the child. Display the PID & PPID of both processes and let the parent display the child's exit status. (Use fork, exit, wait)

3. Create two executables ("PGM1", "PGM2") from two different C source code files: PGM1.c and PGM2.c. These programs should behave as follows:
   a) The program "PGM1" should have 4 different versions to do the following:
      a. Call fork() and Call exec("PGM2") after fork in the child process
      b. Call exec("PGM2") before fork()
      c. Call exec("PGM2") after fork() without checking for return value
      d. Call exec("PGM2") after fork() in the parent process
   b) It does not matter what program PGM2 does, it can have any set of statements

4. Write a program that uses an "exec()" system call to execute a program given to it through command line. For example if the name of your executable is "*myexecutable*" then if you invoke as follows: "*myexecutablemycmd a b*" then "*mycmd*" will be executed with command line parameters "a" & "b" by the program "*myexecutable*" (Note: "*mycmd*" may be any standard unix program or a program created by you)

5. Execute the following code

```
/* processdemo.c */

/* Process demonstration program. Note there are no shared variables */
#include <stdio.h>
#include <stdlib.h>
constclock_t MAXDELAY = 2000000;
int x = 50; /* a global variable */
void delay(clock_t ticks) { /* a "busy" delay */
clock_t start = clock();
do
; while (clock() < start + ticks);
}
voidadjustX(char * legend, inti) {
while (1) /* loop forever */
{ printf("%s: %i\n", legend, x);
x += i;
delay(rand()%MAXDELAY);
}
}
main()
{ int c;
srand(time(NULL));
printf("creating new process:\n");
c = fork();
printf("process %i created\n", c);
if (c==0)
adjustX("child", 1); /* child process */
else
adjustX("parent", -1); /* parent process */
}
```

(a) Examine the source code and try to work out what the output from the program will be.

(b) Compile the program with e.g. gcc processdemo.c -o processdemo

(c) Run the program. Is the output what you expected?

(d) While the program is running, use psxl(in another terminal window) and identify the processes for processdemo . Which process is the parent and which is the child? How can you tell?

(e) Use the *kill* command to stop one of the clones. What happens to the other clone?

(f) Does it make any difference if the parent is killed before the child?